**Bubble Sort in Parallel**

Jacob Ciurca

Computer Science 320 Salisbury University

**Abstract:**

Parallel computing is when you carry out multiple tasks on a computer and perform them simultaneously. With Parallel computing you can take a large problem and divide it into smaller ones to be solved simultaneously using multiple processors. In order to create a parallel program, you must take one large problem and divide it into a sub-set of smaller problems then run that sub-set of problems simultaneously on multiple processors. Our objective is to take the sequential sorting algorithm bubble sort and implement it in parallel. In order to implement bubble sort in parallel I used Openmp software for C++. This allows you to take the inner for loop of bubble sort and have each iteration of the for loop run by a different processor. Anastasio (2004) Bubble sort run sequentially has an average run time of big O n squared, however when run in parallel if you were to have a hypothetical infinite number of processors, the average run time is reduced to big O n meaning it is exponentially faster when run in parallel compared to sequentially.

**I. Introduction:**

Historically, computer programs are made to be run sequentially. This means that the computer performs one task at a time in a set order until all tasks are done. Even though a computer can perform millions of tasks every second, programs have gotten so large that even while performing millions of tasks per second the program will still take a long time to complete. Computers use the logical processors in the CPU or central processing unit to carry out these tasks. When a computer runs a program sequentially, it uses only one logical processor to perform the program. This means that every other logical processor in your computer is not being used wasting potential processing ability. Back when computers were a new thing, it was uncommon for a CPU to have more than one logical processor but in today’s time, almost all computers are made to be run in parallel with multiple CPU cores which can hold up to 2 logical processors. Even the most basic computers today have more than one logical processor. If we were to run the program in parallel, we would make use of the additional previously unused processors making the program both faster and make better use of the computer’s processing ability. Our goal is to take the sequential sorting algorithm bubble sort and implement it in parallel, then analyze our algorithm speedup.

**II. Bubble Sort Introduction:**

Bubble sort is a sorting algorithm used in computer science that has references dating back to 1962. Bubble sort works by taking an array of unsorted elements and checks if adjacent elements are in their proper place, swapping them if necessary. This continues until every element is in its proper place meaning the program would stop there.

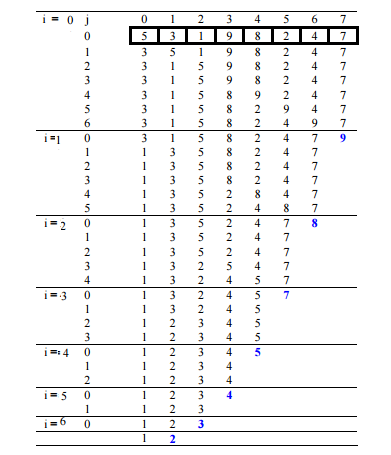


Figure 1. Sequential Bubble Sort Visual Representation. Bubble Sort. (2020, September 30). from https://www.geeksforgeeks.org/bubble-sort/

This algorithm is considered one of the most inefficient sorting algorithms because you are only able to move an element one space each time you do an iteration of the inner for loop. Anastasio (2004) The time complexity of this algorithm is big o n squared or quadratic time complexity. Any algorithm that has a nested for loop with each loop having n iterations is quadratic. For a quadratic algorithm, every time you double the size of n, the run time increases by n\*n.

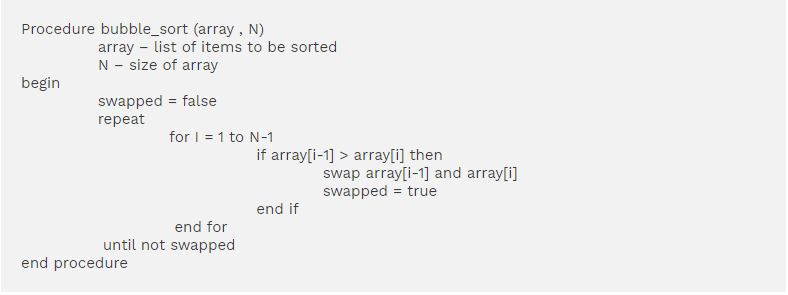


Figure 2. Sequential bubble sort pseudocode. Reprinted from Bubble Sort In C++ With Examples. (2020, November 13) from https://www.softwaretestinghelp.com/bubble-sort/

**III. Creating a Parallel Program:**

In order to create a parallel program, you must take a large problem and divide it into a set of smaller problems for the computer to run in parallel. For bubble sort you would take each iteration of the inner for loop and have them run in parallel thereby decreasing the run time of the sorting algorithm. In order to properly run bubble sort in parallel you must modify the algorithm to an existing algorithm called brick sort. Brick sort works the same way as bubble sort with the only difference being an even and odd phase of the program. What this even and odd phase does is take elements that are indexed at either an odd number or an even number depending on what phase the algorithm is on and compare them with its adjacent element. This is necessary for the algorithm to be run in parallel because this stops the loop from comparing elements more than once for each pass of the inner loop which would make the array sort incorrectly. This is called the race condition, which is when a system tries to perform multiple operations at the same time, but because of how the code works, it must be done in proper sequence in order to be completed correctly.

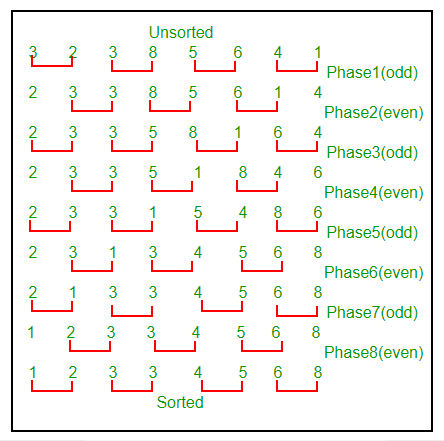


Figure 3. Brick sort algorithm visual representation. Odd-Even Sort / Brick Sort. (2018, April 14). from https://www.geeksforgeeks.org/odd-even-sort-brick-sort/

**IV. OpenMP Software:**

For our parallel program we are going to use C++ OpenMP on Microsoft visual studio to implement our parallel algorithm. OpenMP has several built-in functions that allows a programmer to do several things. One of the functions omp\_get\_max\_threads returns an integer value that is equal to the maximum number of threads you are able to run in parallel on your machine which is equal to the number of logical processers in your CPU. This function is useful because not all computer processors are created equal and while some machines can have as little as one logical processor, there are some that can have as many as 128. Another OpenMP function that will be useful in our program is omp\_set\_num\_threads. This function accepts an integer value and sets the number of threads to be used for parallel computing. This means that if you wanted to only use 2 threads when running your program in parallel you would use omp\_set\_num\_threads and set the accepted integer value as 2. This function also modifies the returned value of the omp\_get\_max\_threads function. This means that if you were to originally have a maximum thread count of 8 and then set the omp\_set\_num\_threads function to 2, the omp\_get\_max\_threads function would then return 2 instead of the original 8. This statement also has the ability to take an integer value greater than your maximum number of available concurrent threads but because your computer can’t run more than the maximum number of threads available, your maximum number of threads stays equal to the number of logical processors in your computer. When you want to run a section of code in parallel with OpenMP you would use the statement #pragma omp parallel to tell the machine you want to run the below code in parallel. This is different if you wanted to run a for loop in parallel as the statement would be #pragma omp parallel for with a default statement and a shared statement. The default would be set to none as we don’t have any defaults in this program. The shared would be set to the array and the even odd counter which would mean the full statement would be #pragma omp parallel for default(none), shared(arr,first). For our entire program we would use the omp\_get\_max\_threads to tell the user how many threads are available to run in parallel. We then ask the user how many threads they would like to use for the parallel program with the user inputting an integer from 1 to the maximum number of threads available setting the number of threads for use with the omp\_set\_num\_threads. We then create the nested for loop with our parallel statement of #pragma omp parallel for right before the second for loop but after the first because we only want the inner for loop to run in parallel.

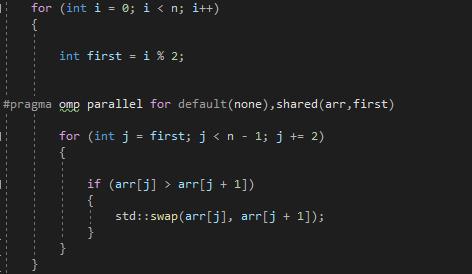


Figure 4. Parallel Implementation of Bubble Sort

**V. Algorithm Speedup Analysis**

Anastasio (2004) As we stated before, bubble sort run sequentially has a time complexity of O n squared meaning it is a quadratic sorting algorithm. When you run bubble sort in parallel the average time complexity is O(n^2/p) where p is the number of processors used. This can be reduced all the way to O(n) with enough processors. In order to reach this level of speedup we would have to have a hypothetical infinite number of processors meaning that when you run the inner for loop you are doing all n iterations in the same amount of time it would take you to run 1 iteration of the inner for loop. This means that as the size increases by n, the run time of the parallel algorithm also increases by n. This makes parallel bubble sort faster than any of the sequential sorting algorithms available such as quicksort which has a time complexity of O n\*log(n). When the parallel algorithm was run you could immediately see the speedup with large arrays. With an array size of 120 thousand you and a thread count of 2, the sorting algorithm run in parallel is 5 seconds faster than sequentially. With a thread count of 5, the algorithm runs 6.2 seconds faster, and with a thread count of 8, the algorithm runs 7.6 seconds faster than sequentially. This speedup would continue until the number of processors would be equal to the size of the array or n in this case. At this point the algorithm would have a linear time complexity meaning that if the size of the array was doubled, then the run time of the algorithm would also double.

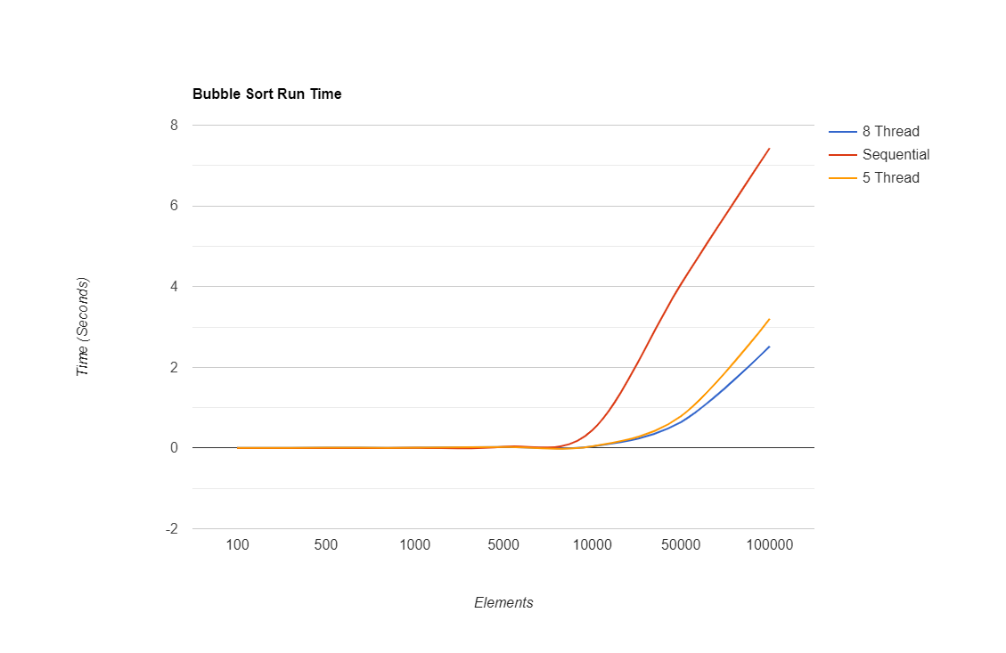


Figure 5. Algorithm Speedup based on Number of Concurrent Threads

**VI. Parallel Bubble Sort Drawbacks**

There are some drawbacks to be aware of when running bubble sort in parallel. When you run a multi-threaded program, you must link the threads back together in order to finish the program. This thread linking takes time to complete, and with a smaller sized array where sorting it sequentially only takes a fraction of a second already, this thread linking can make the parallel algorithm run slower than if it were to run sequentially. However, this is only an issue with smaller arrays and once the array becomes larger with tens of thousands of elements the difference in time between the parallel and sequential sorting becomes so great, that the added time caused by thread linking doesn’t make a difference. Another drawback is that while the time complexity of bubble sort decreases, the work complexity of the algorithm does not. This means that although the amount of time spent sorting is decreased, the amount of work done to sort the algorithm does not change. You are simply spreading out the amount of work to be done throughout different threads and running them at the same time. If you were looking to make the algorithm more efficient and not just faster, then this would not be the way to do it.

**VII. Real World Application of Parallel Bubble Sort**

For a real-world application for parallel bubble sort I found research done by a professor in quantum field theory at the Oregon Institute of Technology that they used bubble sort to determine if sums of operators found in their work would cancel each other out. These operators were found to cancel each other out if they differed by an odd number of swaps and they were able to do the work by hand with a small amount of operators but when they had 100 different operators with 10 thousand permutations it became impossible to do this by hand. That’s where bubble sort came in since with bubble sort, they were able to swap the elements as well as keep track of the number of times the elements swapped.

**IV. Conclusion**

In conclusion, parallel programming is a type of programming that makes better use of your computer’s hardware by using multiple processors to do more than one task at a time, decreasing the amount of time it takes to perform the program. Bubble sort is a sorting algorithm that takes elements in an array and sorts them by swapping one element at a time until the array is sorted making the algorithm very slow and inefficient. We made use of OpenMP software for C++ to create a parallel program that ran bubble sort in parallel increasing the speed at which it ran. Our program was shown to dramatically decrease the run time of the algorithm with the algorithm going from a quadratic function to a linear function if we were to use a hypothetical infinite number of processors. This makes bubble sort in parallel vastly superior to its sequential counterpart as you are greatly reducing the amount of time spent sorting the array as well as making better use of your computer’s hardware.

**References**

Anastasio, T. A. (2004, January 07). Asymptotic Analysis. Retrieved December 07, 2020, from http://faculty.salisbury.edu/~ealu/COSC320/Lectures/asymptotic.pdf

Bubble Sort In C++ With Examples. (2020, November 13). Retrieved December 08, 2020, from <https://www.softwaretestinghelp.com/bubble-sort/>

Bubble Sort. (2020, September 30). Retrieved December 08, 2020, from <https://www.geeksforgeeks.org/bubble-sort/>

Introduction to Parallel Computing Tutorial. (n.d.). Retrieved December 10, 2020, from https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial

Odd-Even Sort / Brick Sort. (2018, April 14). Retrieved December 08, 2020, from https://www.geeksforgeeks.org/odd-even-sort-brick-sort/

Saadeh, R., & Qatawneh, M. (2019, June). Performance Evaluation of Parallel Bubble Sort Algorithm on Supercomputer IMAN1. Retrieved November 20, 2020, from <https://www.researchgate.net/publication/334131761_Performance_Evaluation_of_Parallel_Bubble_Sort_Algorithm_on_Supercomputer_IMAN1>