National University of Computer & Emerging Sciences
Karachi Campus

Parallel Programming - Comparison of sorting Algorithms using Pthreads vs. OpenMP vs. serial

Project Report
Operating Systems
BSR-4C/BCS-4N

Members:
K225018 Aheed Khan
K225024 Muhammad Hadi Shahid

1. Introduction:

The objective of this project is to compare the performance of various sorting algorithms when implemented using different parallel programming paradigms. Specifically, we aim to evaluate the efficiency of sorting algorithms such as Merge Sort, Quick Sort, and Binary Search implemented using Pthreads, OpenMP, and serial execution.

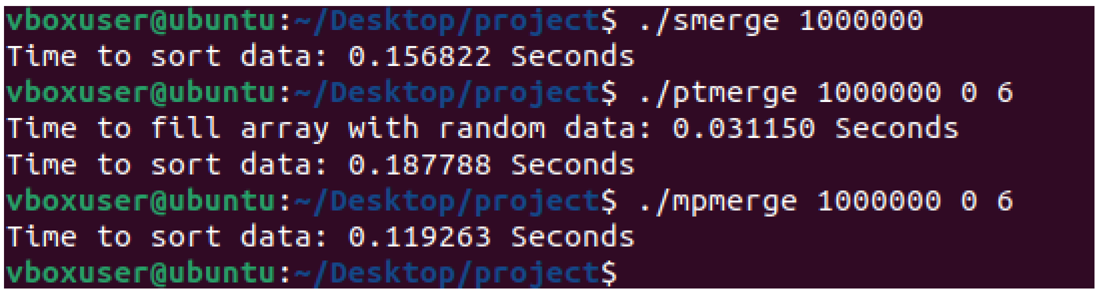
Sorting algorithms are fundamental to computer science and are extensively used in various applications. With the advent of multi-core processors and parallel computing, there is a growing interest in implementing sorting algorithms in parallel to improve performance and scalability.

2. Objectives:

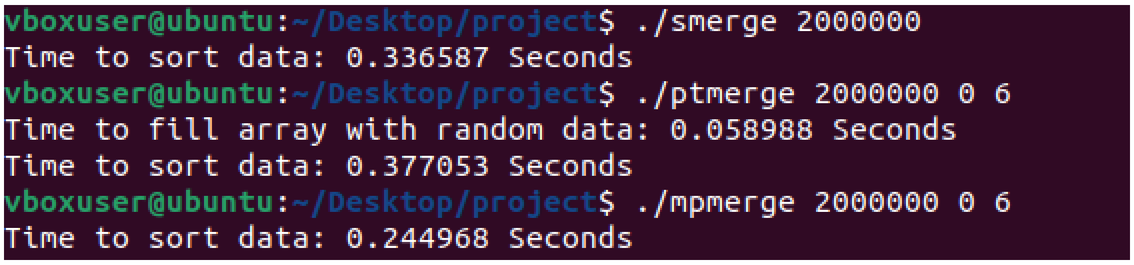
- Implement Merge Sort, Quick Sort, and Binary Search algorithms using Pthreads, OpenMP, and serial execution.
- Evaluate the performance of each algorithm under different input sizes and configurations.
- Compare the efficiency and scalability of sorting algorithms across the three parallel programming methods.
- Conclude the result of our findings.

3.Algorithms and their findings:

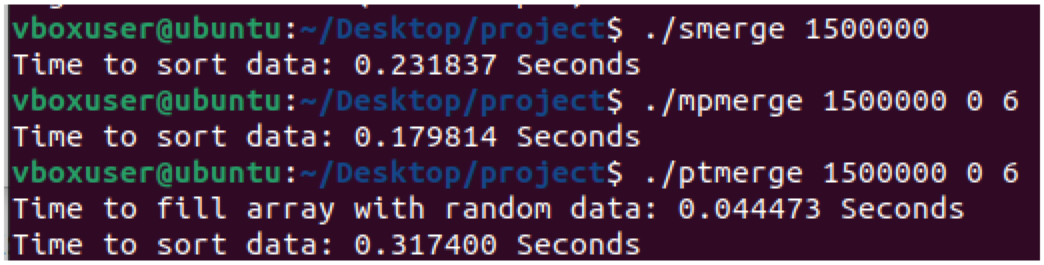
1.Merge Sort:



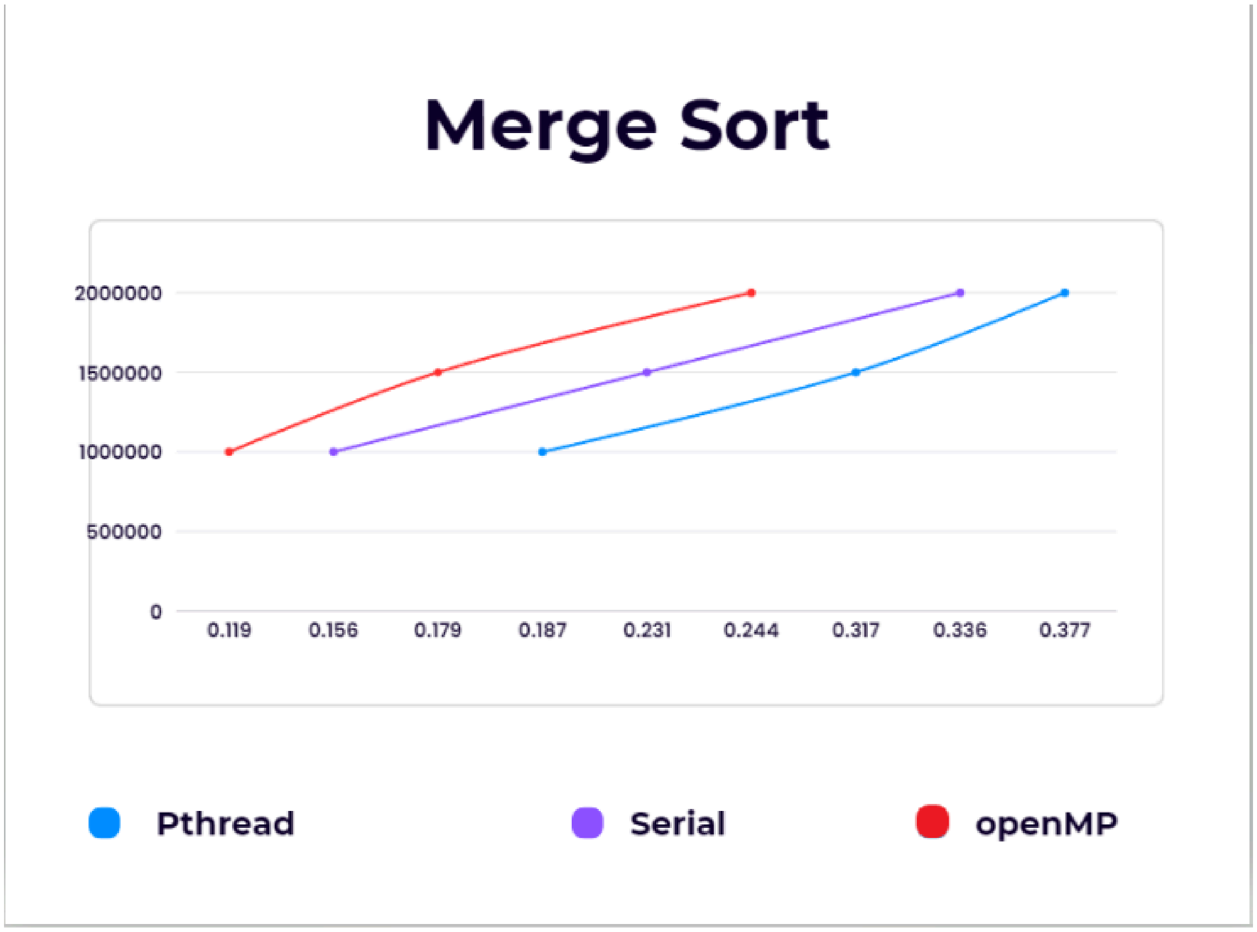
```
vboxuser@ubuntu:~/Desktop/project$ ./smerge 1000000
Time to sort data: 0.156822 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./ptmerge 1000000 0 6
Time to fill array with random data: 0.031150 Seconds
Time to sort data: 0.187788 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./mpmerge 1000000 0 6
Time to sort data: 0.119263 Seconds
vboxuser@ubuntu:~/Desktop/project$
```



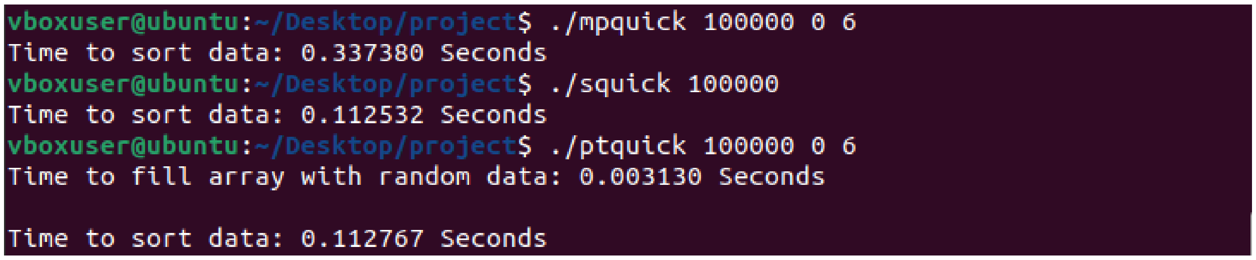
```
vboxuser@ubuntu:~/Desktop/project$ ./smerge 2000000
Time to sort data: 0.336587 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./ptmerge 2000000 0 6
Time to fill array with random data: 0.058988 Seconds
Time to sort data: 0.377053 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./mpmerge 2000000 0 6
Time to sort data: 0.244968 Seconds
```



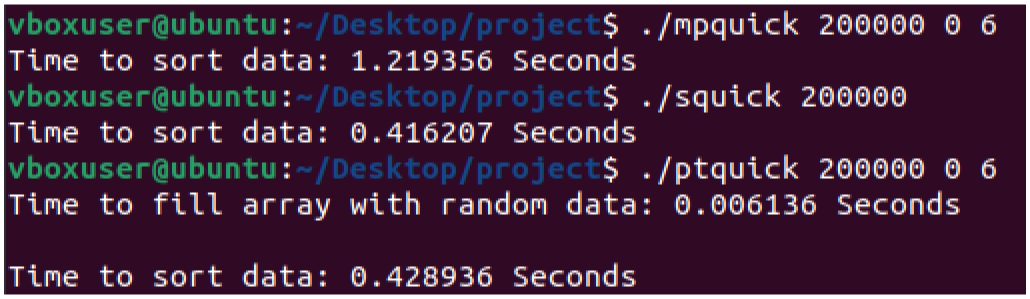
```
vboxuser@ubuntu:~/Desktop/project$ ./smerge 1500000
Time to sort data: 0.231837 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./mpmerge 1500000 0 6
Time to sort data: 0.179814 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./ptmerge 1500000 0 6
Time to fill array with random data: 0.044473 Seconds
Time to sort data: 0.317400 Seconds
```



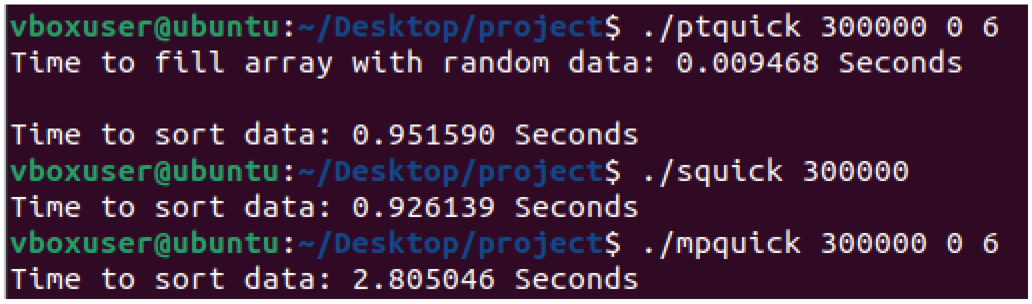
2.Quick Sort:



```
vboxuser@ubuntu:~/Desktop/project$ ./mpquick 100000 0 6
Time to sort data: 0.337380 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./squick 100000
Time to sort data: 0.112532 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./ptquick 100000 0 6
Time to fill array with random data: 0.003130 Seconds

Time to sort data: 0.112767 Seconds
```

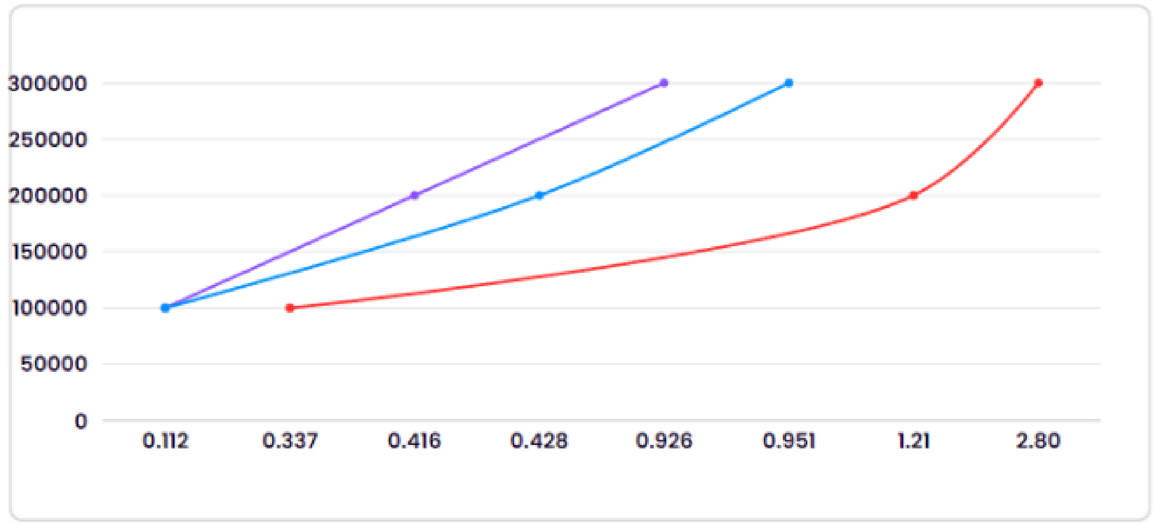


```
vboxuser@ubuntu:~/Desktop/project$ ./mpquick 200000 0 6
Time to sort data: 1.219356 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./squick 200000
Time to sort data: 0.416207 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./ptquick 200000 0 6
Time to fill array with random data: 0.006136 Seconds

Time to sort data: 0.428936 Seconds
```



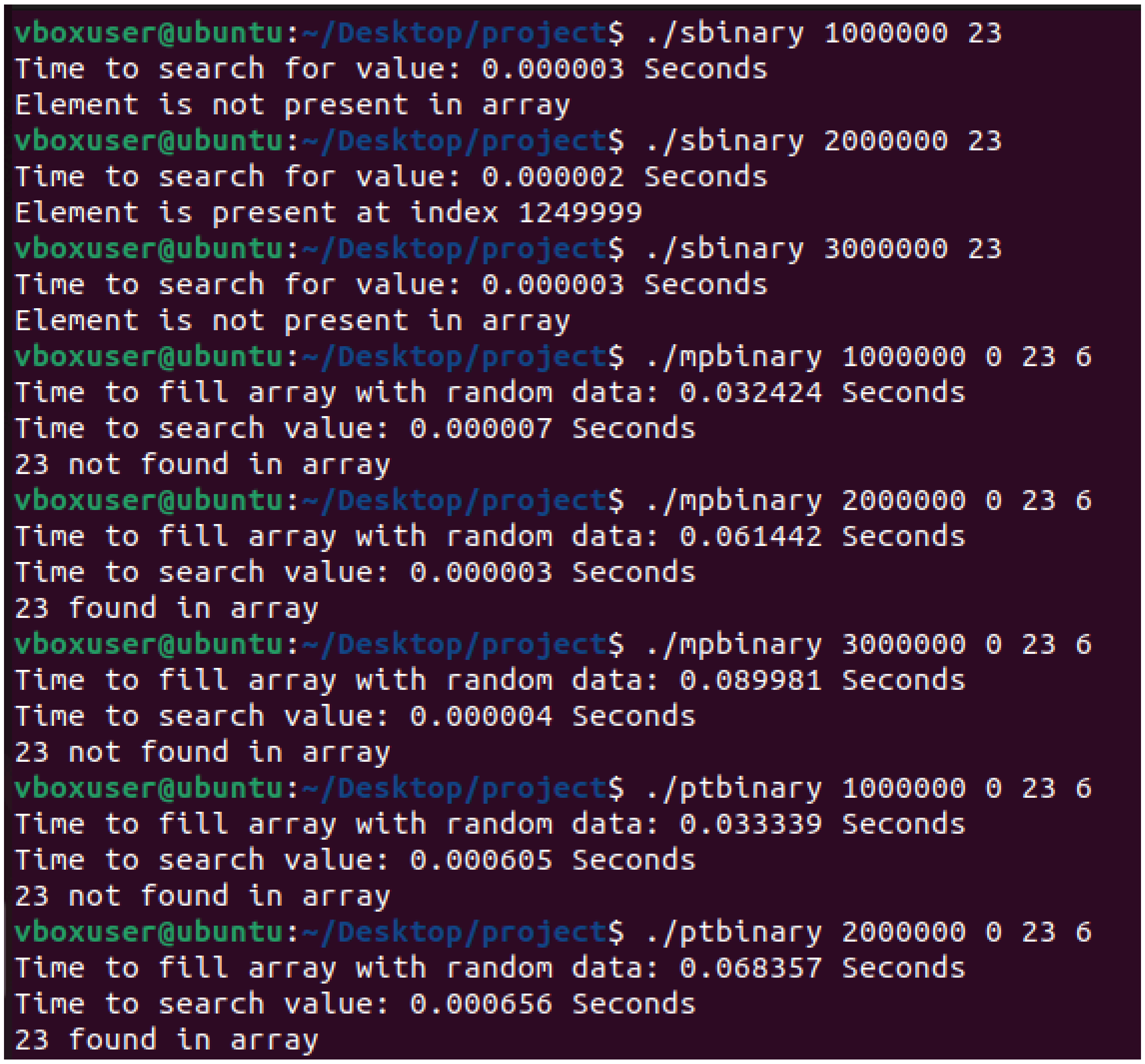
```
vboxuser@ubuntu:~/Desktop/project$ ./ptquick 300000 0 6
Time to fill array with random data: 0.009468 Seconds

Time to sort data: 0.951590 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./squick 300000
Time to sort data: 0.926139 Seconds
vboxuser@ubuntu:~/Desktop/project$ ./mpquick 300000 0 6
Time to sort data: 2.805046 Seconds
```

Quick Sort

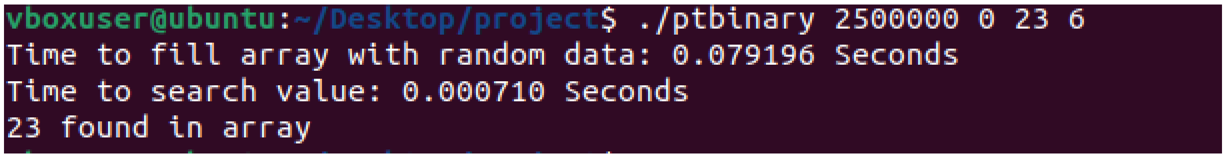


● Pthread ● Serial 🟥 openMP

3.Binary Search:

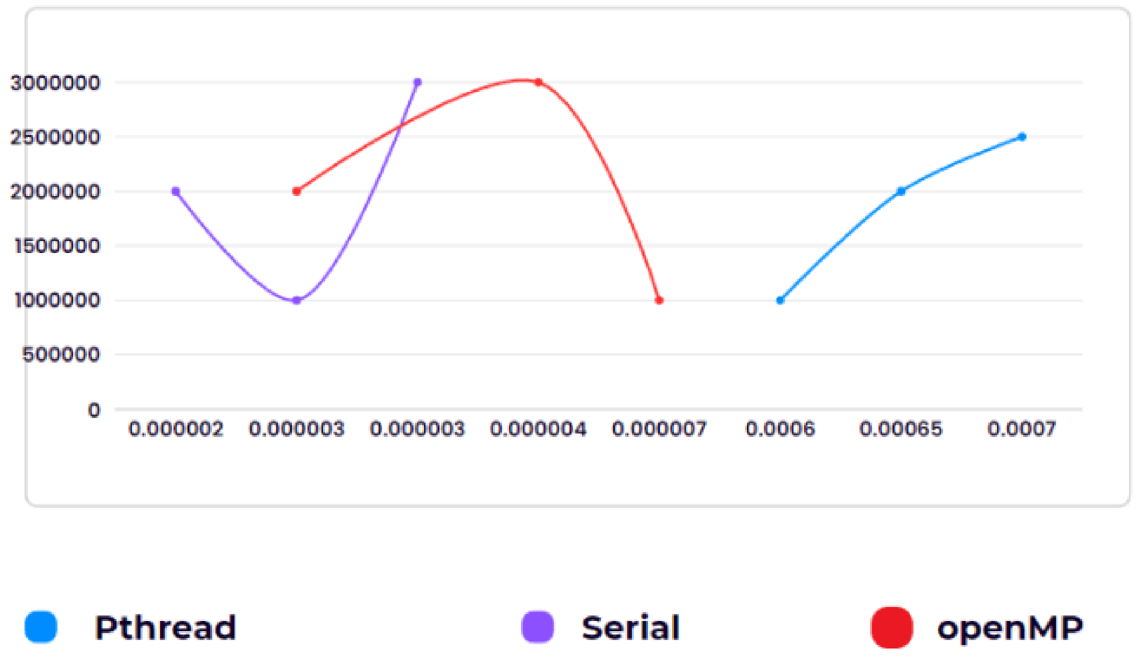


```
vboxuser@ubuntu:~/Desktop/project$ ./sbinary 1000000 23
Time to search for value: 0.000003 Seconds
Element is not present in array
vboxuser@ubuntu:~/Desktop/project$ ./sbinary 2000000 23
Time to search for value: 0.000002 Seconds
Element is present at index 1249999
vboxuser@ubuntu:~/Desktop/project$ ./sbinary 3000000 23
Time to search for value: 0.000003 Seconds
Element is not present in array
vboxuser@ubuntu:~/Desktop/project$ ./mpbinary 1000000 0 23 6
Time to fill array with random data: 0.032424 Seconds
Time to search value: 0.000007 Seconds
23 not found in array
vboxuser@ubuntu:~/Desktop/project$ ./mpbinary 2000000 0 23 6
Time to fill array with random data: 0.061442 Seconds
Time to search value: 0.000003 Seconds
23 found in array
vboxuser@ubuntu:~/Desktop/project$ ./mpbinary 3000000 0 23 6
Time to fill array with random data: 0.089981 Seconds
Time to search value: 0.000004 Seconds
23 not found in array
vboxuser@ubuntu:~/Desktop/project$ ./ptbinary 1000000 0 23 6
Time to fill array with random data: 0.033339 Seconds
Time to search value: 0.000605 Seconds
23 not found in array
vboxuser@ubuntu:~/Desktop/project$ ./ptbinary 2000000 0 23 6
Time to fill array with random data: 0.068357 Seconds
Time to search value: 0.000656 Seconds
23 found in array
```



```
vboxuser@ubuntu:~/Desktop/project$ ./ptbinary 2500000 0 23 6
Time to fill array with random data: 0.079196 Seconds
Time to search value: 0.000710 Seconds
23 found in array
```

Binary Seach



4. Conclusion:

Our findings conclude that in tasks as simple as sorting and searching serial code works faster than parallel because of multiple reasons.Complex tasks that require parallel calling of computer resources take advantage of multi-threading properly.In simple tasks , creating thread adds overhead and excessive function calls also increase time complexity.Merge sort proved to be faster hence in that algorithm after division of the array every thread can work independently.There is also the problem that all this testing is done on a virtual machine which cannot take full advantage of a computer's resources.We also found that decreasing the number of threads also increases the speed of the algorithm , since fewer threads create less overhead.