High Performance Computing
and modern Architectures

Comparison analysis of parallel sorting algorithms

Shushkova Varvara

Skoltech 2021

Sorting algorithms

Merge Sort

Divide the unsorted list into *n* sublists, (a list of one element is considered sorted). Repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining.

- Linear Complexity $O(nlogn)$
- Parallel Complexity $O(\frac{n}{p}log(n))$

Bucket Sort

1. Set up an array of initially empty buckets
2. Scatter: Go over the original array, putting each object in its bucket.
3. Sort each non-empty bucket.
4. Gather: Visit the buckets in order and put all elements back into the original array.

Results

Merge Sort

- Vector: 100000000
- Number of threads: 2

Sequential version

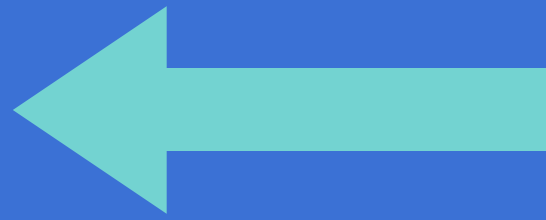
101

MPI

82.75

OMP

70



Bucket Sort

- Vector: 100000000
- Number of threads: 2

Results

Sequential version

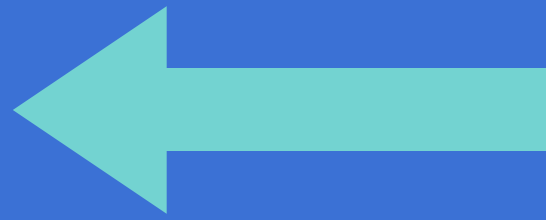
108.596

MPI

87.76

OMP

72



Bucket Sort

Results

Sort	Time
Seq Merge	158
OMP	128

Experiments

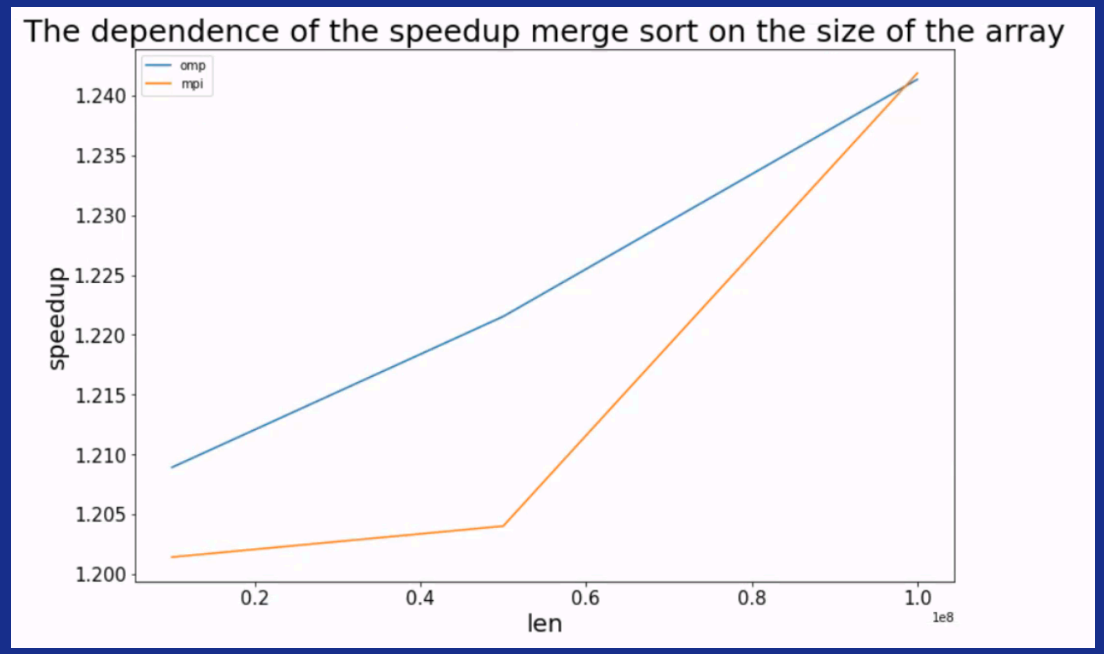
1. Sequential Merge
2. OMP Merge

- **Vector: 100000000**
- **Number of threads: 2**

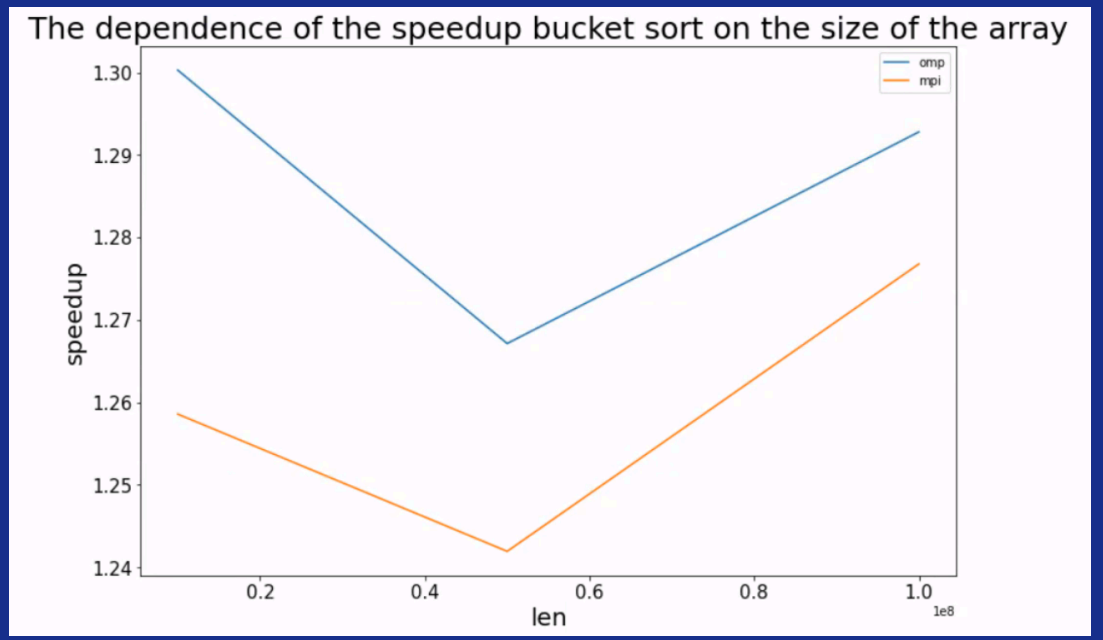
Results Comparison

Vector size

Merge Sort



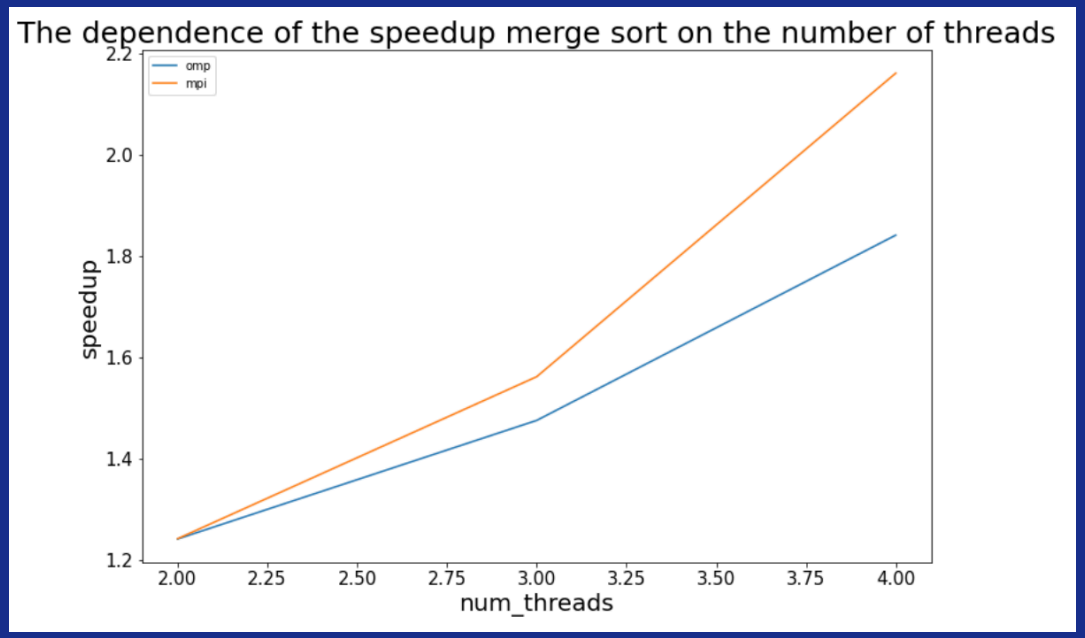
Bucket Sort



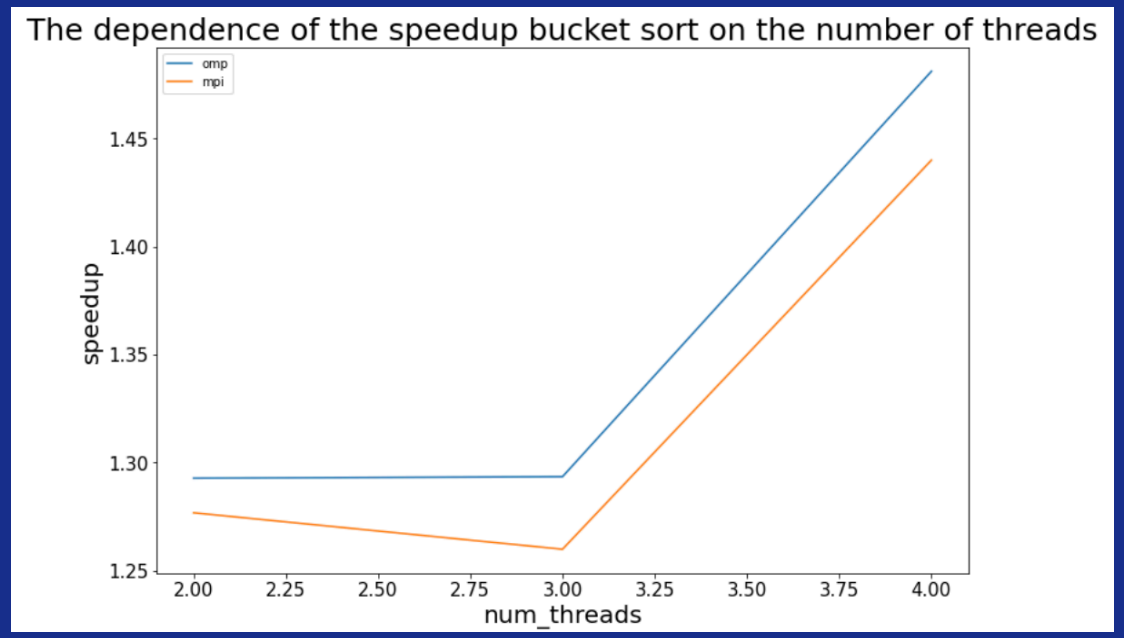
Results Comparison

Number of threads

Merge Sort



Bucket Sort



Conclusion

Two versions of parallel algorithms are implemented

- ✓ a **comparison** of results is made
- ✓ **problems** with memory problems have been reached and analysed
- ✓ algorithm **optimization** is performed

Conclusion

- ▲ not enough memory on my computer
- ▲ MPI needs more memory then OpenMP
- ▲ OMP is faster for my implementation

High Performance Computing
and modern Architectures

Comparison analysis of parallel sorting algorithms

Shushkova Varvara

Skoltech 2021