

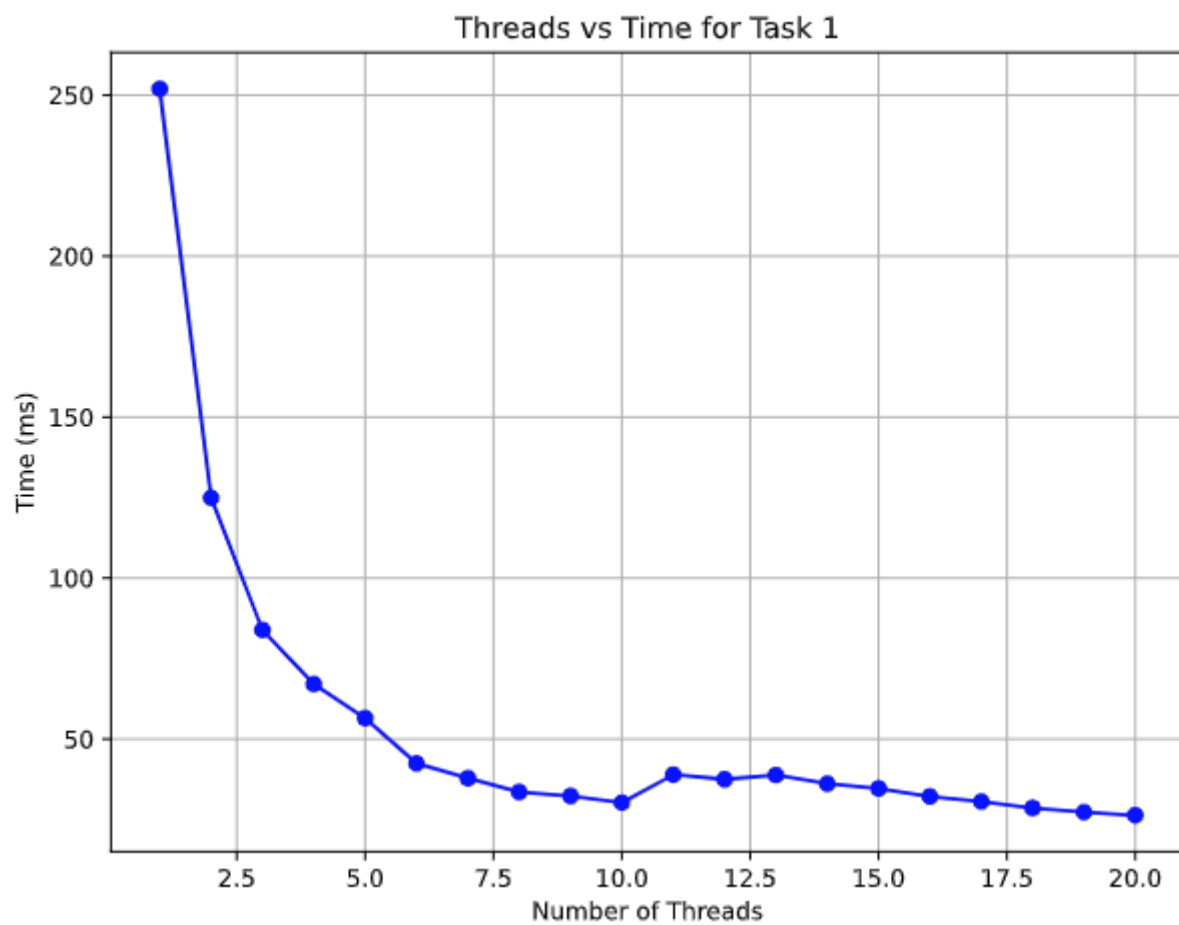
Assignment 3

Github Link

- **HW02:** <https://github.com/hemanth-nakshatri/repo759/tree/main/HW03>
- **Parent Repo:** <https://github.com/hemanth-nakshatri/repo759>

Question 1

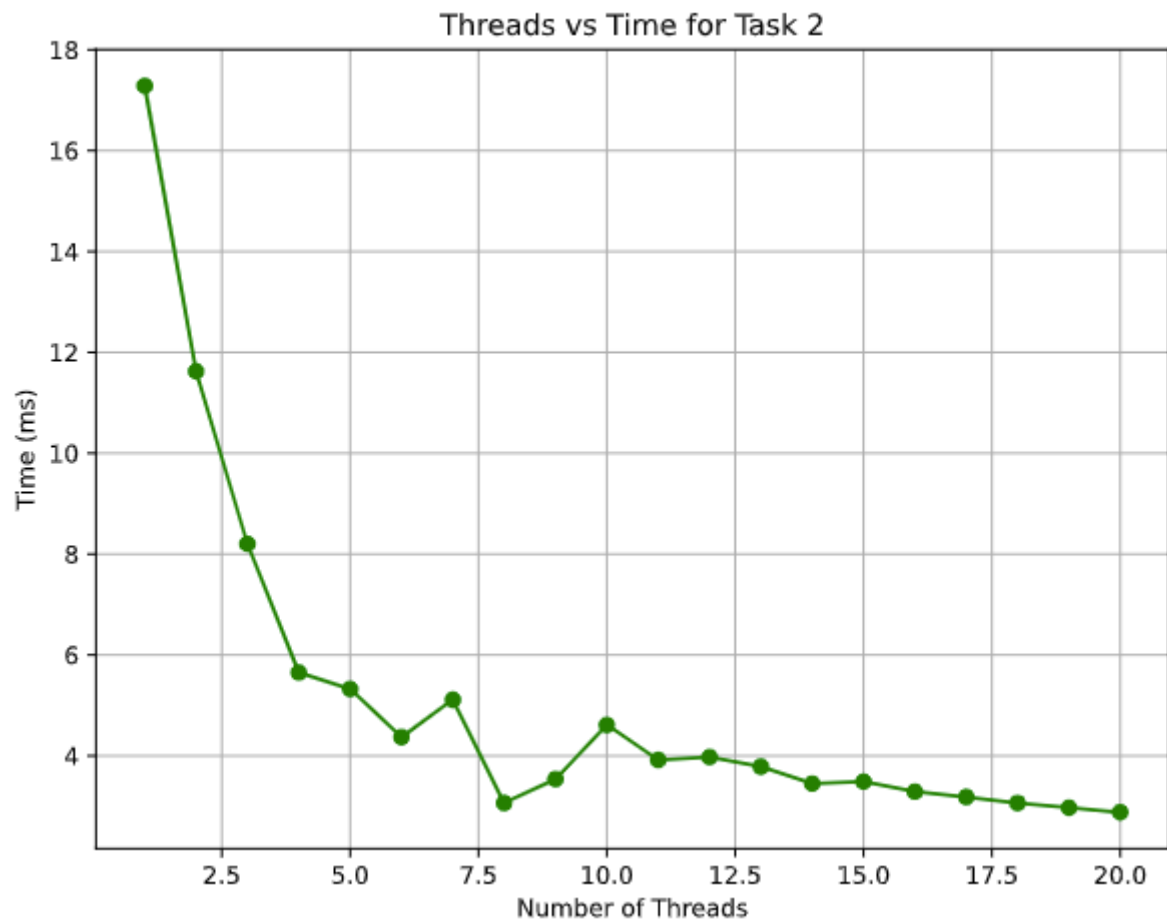
- `matmul` function was parallelized using `#pragma omp parallel for` directive.
- `task1.pdf` results are shown below.



Question 2

- `convolve` function was parallelized using OpenMP in a similar way.

- `task2.pdf` results are shown below.

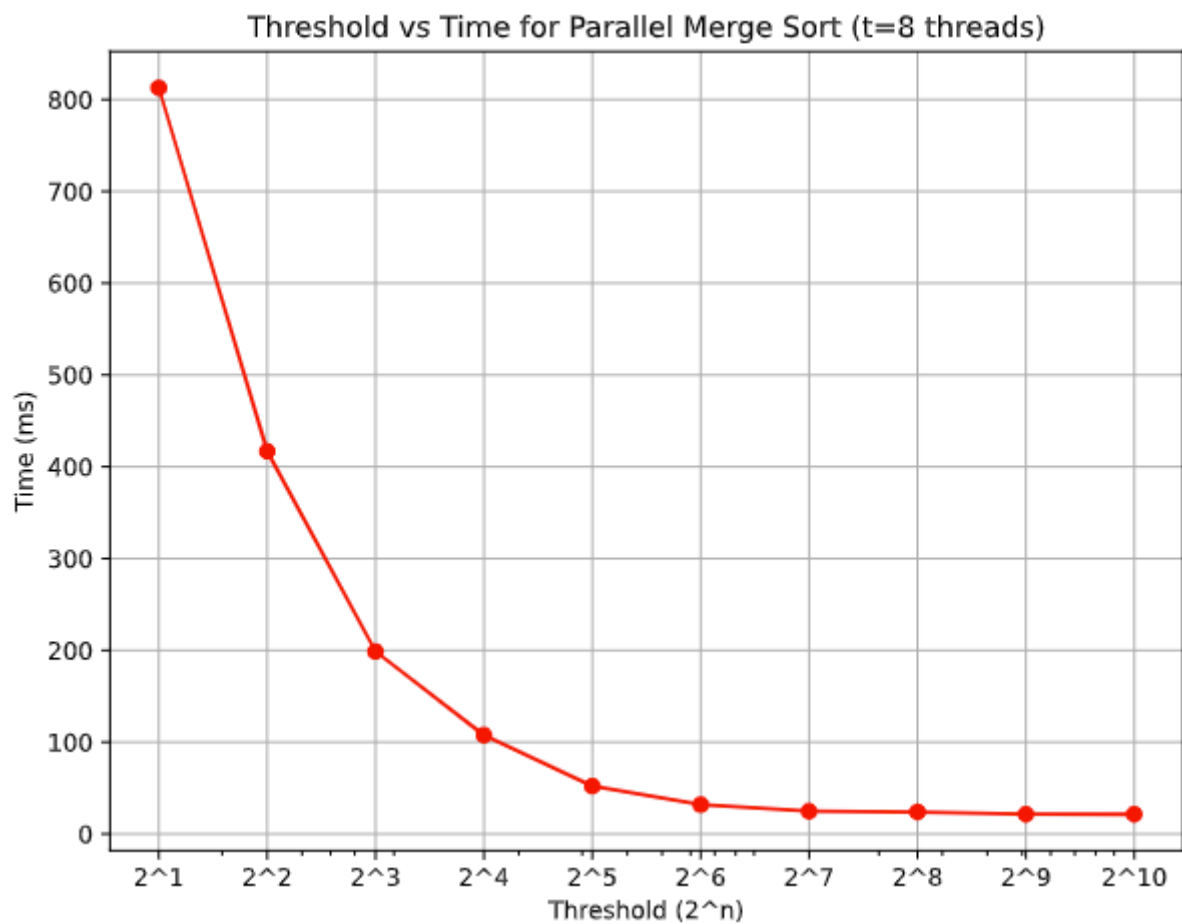


- We see a steep decrease in compute time until about approximately 10 threads. After that, it looks like it stagnates. This could be due to a few factors:
 - **Overhead of thread management:** Creation and management of more threads can introduce significant overhead, especially for small tasks.
 - **Limited workload for each thread:** With every thread added, each thread is responsible for a smaller portion of the workload. This potentially makes the cost of the other overhead more significant.

Question 3

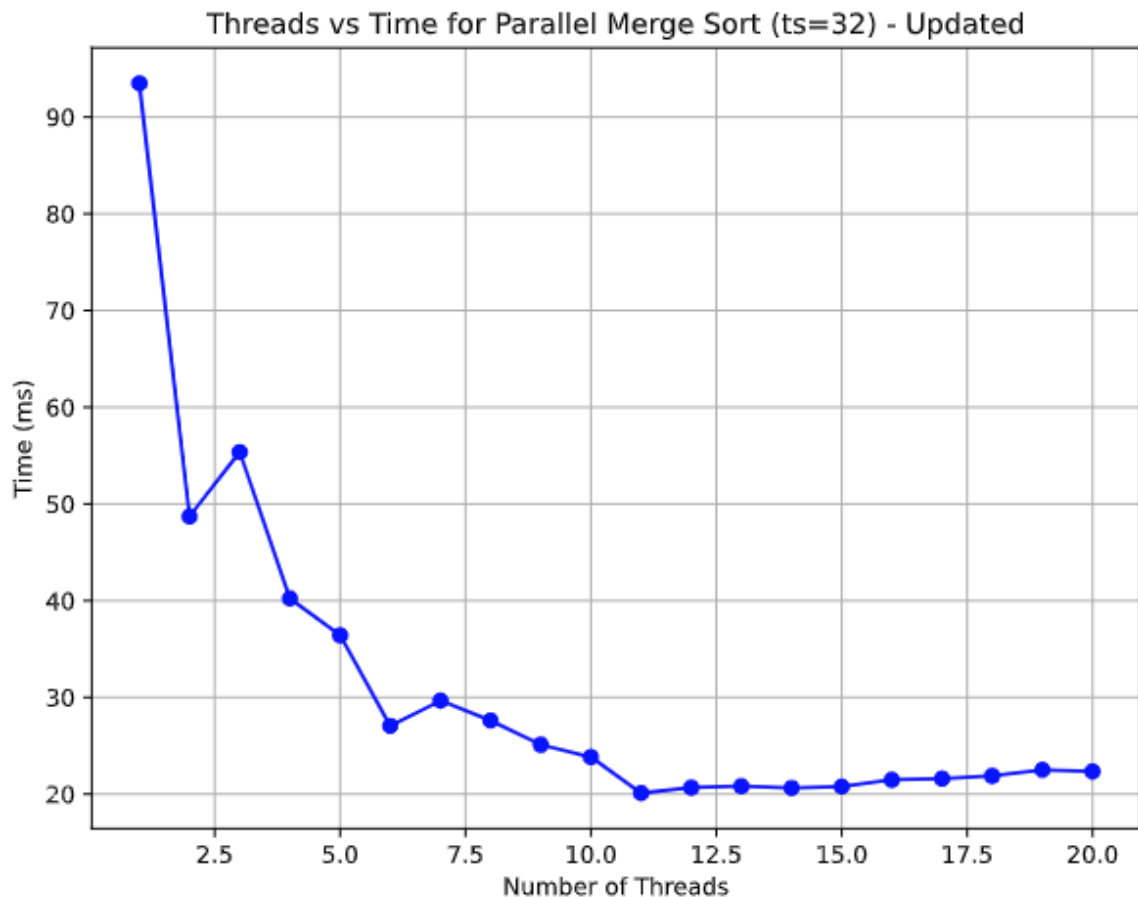
- `merge sort` function was implemented and parallelized using OpenMP.

- `task3_ts.pdf` results are shown below.



- `ts = 2^6` was chosen since that is the threshold where the time curve starts to flatten out without any significant decrease in time.

- [task3_t.pdf](#) results are shown below.



ChatGPT and other LLM Main Help

Prompt 1

Prompt: Give me a step by step pseudo code for merge sort algorithm that I can implement in C++

ChatGPT: Certainly! Here's a step-by-step pseudocode for the Merge Sort algorithm that you can easily translate into C++:

Merge Sort Pseudocode:

1. Main Function - mergeSort(arr, left, right) Input: An array arr[], left (starting index), and right (ending index). Output: A sorted version of the input array.

```
function mergeSort(arr, left, right)
    if left >= right:
        return

    // Step 1: Find the middle point to divide the array into two halves
    mid = (left + right) / 2

    // Step 2: Call mergeSort for the first half
    mergeSort(arr, left, mid)

    // Step 3: Call mergeSort for the second half
```

```
mergeSort(arr, mid + 1, right)

// Step 4: Merge the two halves
merge(arr, left, mid, right)
```

2. Merge Function - merge(arr, left, mid, right) Input: An array arr[], left (starting index), mid (middle index), and right (ending index). Output: Merges two sorted halves into a single sorted array.

```
function merge(arr, left, mid, right)
    // Step 1: Find the sizes of the two subarrays to be merged
    n1 = mid - left + 1
    n2 = right - mid

    // Step 2: Create temporary arrays
    L[n1], R[n2]

    // Step 3: Copy data to temporary arrays L[] and R[]
    for i = 0 to n1 - 1
        L[i] = arr[left + i]

    for j = 0 to n2 - 1
        R[j] = arr[mid + 1 + j]

    // Step 4: Merge the two subarrays back into arr[]
    i = 0, j = 0, k = left

    while i < n1 and j < n2
        if L[i] <= R[j]:
            arr[k] = L[i]
            i = i + 1
        else:
            arr[k] = R[j]
            j = j + 1
        k = k + 1

    // Step 5: Copy any remaining elements of L[] (if any)
    while i < n1
        arr[k] = L[i]
        i = i + 1
        k = k + 1

    // Step 6: Copy any remaining elements of R[] (if any)
    while j < n2
        arr[k] = R[j]
        j = j + 1
        k = k + 1
```

Prompt 2

Prompt: This is the output of my terminal when I run my code. Give me a python code to plot the last line of each output parts vs the number of threads. Output of terminal: </My terminal output/>

ChatGPT: Sure! Here is a Python code to plot the last line of each output part (time taken) vs the number of threads:

```
import matplotlib.pyplot as plt

# Data: thread counts and corresponding times in milliseconds
threads = list(range(1, 21))
times = [
    251.983, 124.94, 83.8743, 67.0746, 56.4641, 42.4258, 37.8214, 33.5774,
    32.3255, 30.2553, 38.9272, 37.4728, 38.7733, 36.1111, 34.6684, 32.1828,
    30.5802, 28.607, 27.2935, 26.2977
]

# Plotting thread count vs time
plt.figure(figsize=(8, 6))
plt.plot(threads, times, marker='o', linestyle='-', color='b')
plt.title('Threads vs Output')
plt.xlabel('Number of Threads')
plt.ylabel('Time (ms)')
plt.grid(True)

# Display the plot
plt.show()
```

Prompt 3

Prompt: Here is one of my slurm batch scripts. Rewrite this script to run a different task.

Slurm batch script:

```
#!/usr/bin/env bash #SBATCH --job-name=hw3_task1 # Job name #SBATCH --partition=instruction # Partition
(queue) name #SBATCH --time=00:30:00 # Run time (hh:mm:ss) #SBATCH --cpus-per-task=20 # Request 20
threads (OpenMP will use this) #SBATCH --nodes=1 # Run on a single node #SBATCH --
output=task1_output.out # Standard output log #SBATCH --error=task1_error.err # Standard error log

echo "Running task1 for matrix multiplication using OpenMP"
```

```
g++ task1.cpp matmul.cpp -Wall -O3 -std=c++17 -fopenmp -o task1
```

```
for t in {1..20}; do echo "Running with t=$t threads" ./task1 1024 $t done
```

The task is for running task 3. I compile the code using the command `g++ task3.cpp msort.cpp -Wall -O3 -std=c++17 -fopenmp -o task3`. Also the loop should be from 2^1 , 2^2 ... 2^{10} .

The run command is of format: `./task3 1000000 t ts`

t = 8 for task 3.1

Do one more script for task 3.2. Same compile command and loop threads ts from 1 to 20 assume $ts = 32$ for task 3.2

ChatGPT: Certainly! Here is a revised slurm batch scripts.

1. task3_1_slurm.sh:

```
#!/usr/bin/env bash
#SBATCH --job-name=hw3_task3          # Job name
#SBATCH --partition=instruction        # Partition (queue) name
#SBATCH --time=01:00:00              # Run time (hh:mm:ss)
#SBATCH --cpus-per-task=20           # Request 20 threads (OpenMP will use this)
#SBATCH --nodes=1                    # Run on a single node
#SBATCH --output=task3_output.out     # Standard output log
#SBATCH --error=task3_error.err       # Standard error log

# Print task details
echo "Running task3 for parallel merge sort using OpenMP"

g++ task3.cpp msort.cpp -Wall -O3 -std=c++17 -fopenmp -o task3

for ts_exp in {1..10}; do
    ts=$((2 ** ts_exp)) # Compute 2^ts_exp
    echo "Running with t=8 threads and threshold=2^$ts_exp ($ts)"
    ./task3 1024 8 $ts # Sorting array of size 1,000,000
done
```

2. task3_2_slurm.sh:

```
#!/usr/bin/env bash
#SBATCH --job-name=hw3_task3_final    # Job name
#SBATCH --partition=instruction        # Partition (queue) name
#SBATCH --time=01:00:00              # Run time (hh:mm:ss)
#SBATCH --cpus-per-task=20           # Request 20 threads
#SBATCH --nodes=1                    # Run on a single node
#SBATCH --output=task3_final_output.out # Standard output log
#SBATCH --error=task3_final_error.err  # Standard error log

echo "Running task3 for parallel merge sort with ts=32 using OpenMP"

g++ task3.cpp msort.cpp -Wall -O3 -std=c++17 -fopenmp -o task3

# Run task3 with ts=128 (2^6) for t=1 to t=20 threads
for t in {1..20}; do
    echo "Running with t=$t threads and ts=32"
    ./task3 1000000 $t 32 # Sorting array of size 1,000,000 with ts=32
done
```