CS 581: High Performance Computing
Emory University
Spring 2025
Prof. Tianshi Xu

Homework # 2

Mitchell Scott
(mtscot4)

1 Environment

1. **List the environment you used to run the code (Operating System, CPU, RAM, etc.)** .

   Solution. The local environment where I was running the code was on a 2022 MacBook Pro with Apple M2 chip, 8 GB of RAM running macOS Sequoia 15.3.1.

2 Sequential Code

1. The results, which are averages of five runs where the fastest and slowest are removed, are seen in Tab. 1.

Trial	`small.bin`	`medium.bin`	`large.bin`	`huge.bin`
1	6.5834e-05 (**high**)	0.000828625 (**low**)	1.23898 (**low**)	6.73952
2	3.0208e-05	0.00109362 (**high**)	1.24350	6.69056 (**low**)
3	3.0875e-05	0.0009965	1.32917 (**high**)	6.79279
4	2.8542e-05 (**low**)	0.000983875	1.25815	6.71300
5	2.9375e-05	0.000959958	1.26216	6.82485 (**high**)
Avg	**3.01527e-5**	**0.000980111**	**1.25460**	**6.748437**

Table 1: Sequential Code for different file sizes. All are reported in seconds.

3 Parallel Code

3.1 Run Time

3.1.1 `small.bin`

The results for `small.bin` are given in table 2.

3.1.2 `medium.bin`

The results for `small.bin` are given in table 3.

Trial	1 thread (s)	2 thread (s)	4 thread (s)	8 thread (s)
1	1.1875e-4	1.60708e-4	0.00127112	0.001822 (**low**)
2	1.22291e-4	1.74000e-4	0.000373667	0.00222171
3	8.9958e-05 (**low**)	1.66537e-4	0.000272833 (**low**)	0.002149
4	1.5875e-4	1.42042e-4 (**low**)	0.00167750 (**high**)	0.00444679 (**high**)
5	1.67583e-4 (**high**)	1.82500e-4 (**high**)	0.000644041	0.00199421
Avg	**1.33264e-4**	**1.67082e-4**	**7.62943e-4**	**0.00212164**

Table 2: Time (s) to sort data in `small.bin` dataset for varying number of threads.

Trial	1 thread (s)	2 thread (s)	4 thread (s)	8 thread (s)
1	0.00123242 (**high**)	0.00364746 (**high**)	0.00293096 (**high**)	0.00612125
2	0.00110038	0.00269762	0.00197646	0.00327346 (**low**)
3	0.00122571	0.00240858	0.00184267 (**low**)	0.0141560 (**high**)
4	0.000960625 (**low**)	0.00208879 (**low**)	0.00202196	0.00400771
5	0.00100529	0.00277246	0.00209833	0.00410279
Avg	**0.00111046**	**0.00262622**	**0.00203225**	**0.00474392**

Table 3: Time (s) to sort data in `medium.bin` dataset for varying number of threads.

3.1.3 `large.bin`

The results for `large.bin` are given in table 4.

Trial	1 thread (s)	2 thread (s)	4 thread (s)	8 thread (s)
1	1.14397	0.908378 (**low**)	0.529193 (**low**)	0.534565(**low**)
2	1.13633	0.952551 (**high**)	0.534829	0.696034 (**high**)
3	1.17768 (**high**)	0.909059	0.532734	0.586798
4	1.1341	0.914919	0.533748	0.576026
5	1.13208 (**low**)	0.91012	0.543836 (**high**)	0.607107
Avg	**1.13813**	**0.911366**	**0.533770**	**0.589977**

Table 4: Time (s) to sort data in `large.bin` dataset for varying number of threads.

3.1.4 `huge.bin`

The results for `huge.bin` are given in table 5.

3.2 Speedup

In terms of the speedup, first we need to recall that

$$\text{Speedup} = \frac{T_{\text{sequential}}}{T_{\text{parallel}}} \tag{1}$$

So simply by dividing our times from the result in section 1, the speedup is seen in Tabs. 2, 3, 4 and 5, we arrive at the speedup, which is seen in Tab. 6.
This data is also presented in Fig. 1. As we can see from both the figure and the table, we have no great speedup for any number of processors with the `small` and the `medium`

Trial	1 thread (s)	2 thread (s)	4 thread (s)	8 thread (s)
1	5.99075 (**high**)	4.24292 (**low**)	2.60889 (**low**)	2.49775
2	5.96154	4.34822	2.76255 (**high**)	2.54355 (**high**)
3	5.88652 (**low**)	4.27647	2.61821	2.34084
4	5.94369	4.40685 (**high**)	2.69274	2.25988 (**low**)
5	5.98632	4.27752	2.71596	2.42781
Avg	**5.96385**	**4.30074**	**2.67564**	**2.38880**

Table 5: Time (s) to sort data in `huge.bin` dataset for varying number of threads.

# Threads	`small`	`medium`	`large`	`huge`
1	0.2263	0.8826	1.1023	1.1316
2	0.1805	0.3732	1.3766	1.5691
4	0.0395	0.4823	2.3504	2.5222
8	0.0142	0.2066	2.1265	2.8250

Table 6: Speedup of different data sets for varying number of threads.

dataset. This is due to the parallel overhead and message transferring, which isn't worth it at this size. However, for the `large` and the `huge` dataset, we see advantages for using the parallel code. Another note is the large dataset is only large enough for 4 processors to make sense to be used. It is large enough for some speedup but not too large to justify the extra communication costs that 8 have.

3.3 Data Splits

Ideally, we would split the data in such a way that all processors get the same amount of data to sort locally, but since we don't know the distribution of the data, we cannot *a priori* know the best splitting vector. Instead, we

4 Implementation

1. **Explain the Algorithm you used.**

   Solution.

2. **What is the performance bottleneck of your parallel implementation?**

   Solution.

Acknowledgements

I would like to acknowledge that I worked with fellow CS 581 students Emma Hart in OH on 2-4-25. Additionally I did use the Github Copilot to expedite my code (of course I checked the work it produced before I ran it). Lastly, I use resources from the University of Michigan EECS department to figure out how to structure my PThread code.

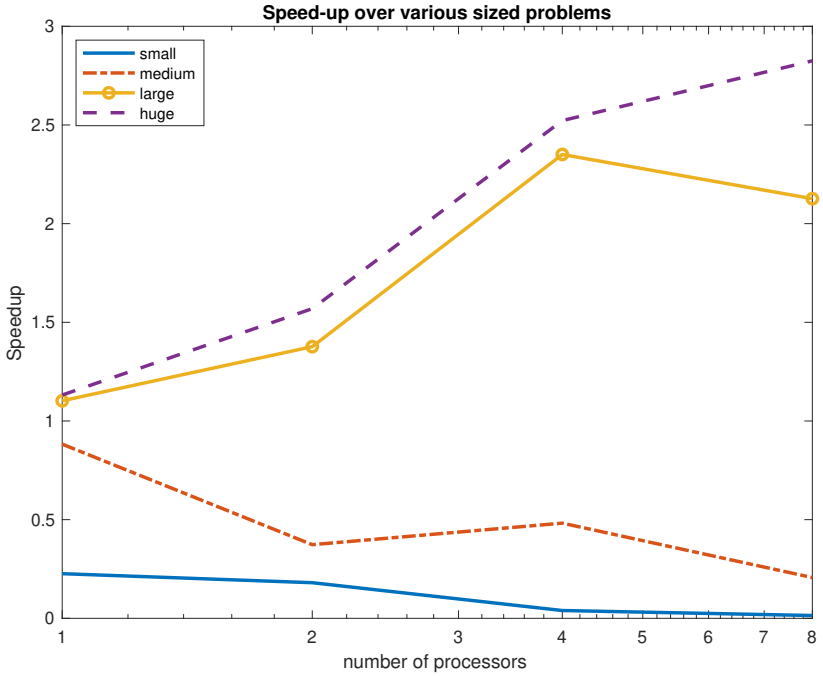

Figure 1: Comparing the speedup curves for different datasets.

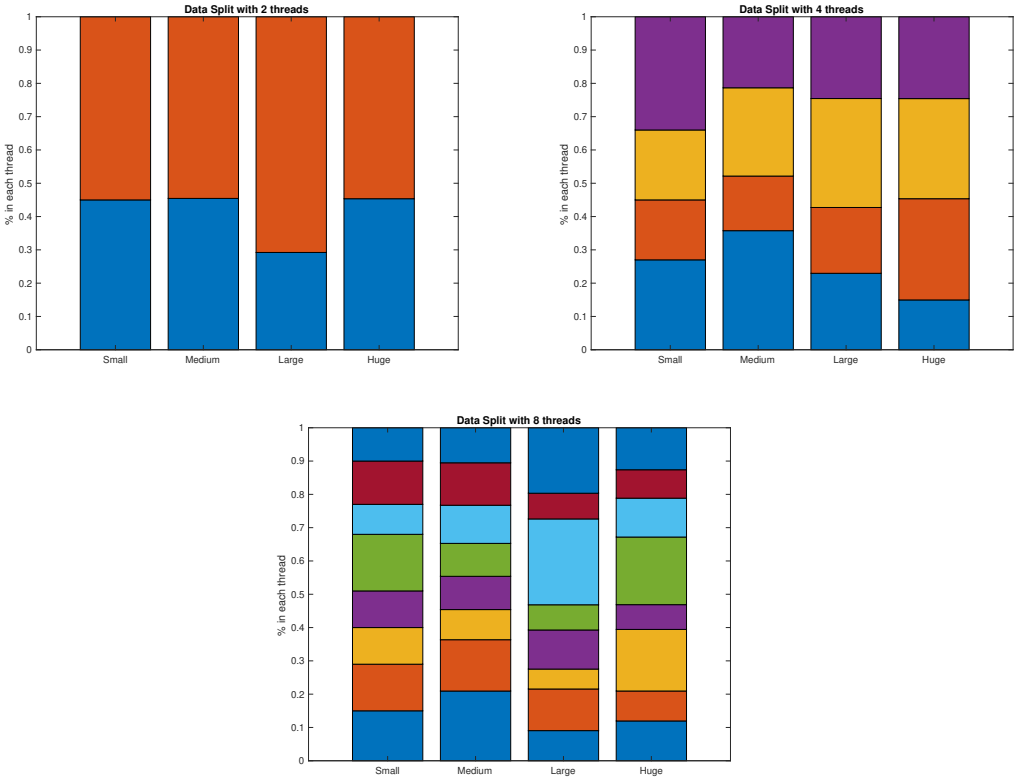

Figure 2: Visualized the data splits between processors for all datasets using (a) 2 threads, (b) 4 threads, and (c) 8 threads.