Xiyue Suo(001348347)

**Program Structures & Algorithms**

**Fall 2021**

**Assignment No. 5**

* **Task (List down the tasks performed in the Assignment)**

For this assignment, I carried out multiple experiments to test the efficiency of parallel sorting, with different array size(2,000,000, 3,500,000), thread number(2, 4, 6, 8), and cutoff values(10% - 80% array size).

The parallel sorting is basically from merge sorting with CompletableFuture, which can run sort tests at same time.

* **Overall Conclusions**

1. **The sort time decrease as the thread number increase. But it will have a most efficient number, in my computer, is 8(threads).**
2. **There is a rapid decrease in sort time when cutoff equals around 25% of array size (ex. 500,000 for array size = 2,000,000).**
3. **There is a rapid increase in sort time when cutoff equals around 50% of array size (ex. 1,000,000 for array size = 2,000,000).**

* **Observation and Conclusion:**

**Experiment A:**

**with array size = 2,000,000, thread = 2, 4, 8, cutoff for parallel sort from 510,000(25.5%) to 1,000,000(50%).**

1. The run time is large for first parallel sort run, but is relatively same for rest of sorts.
2. The sort time is about 170 when thread is 2, about 110 when thread is 4, about 80 when thread is 8.

Table

Description automatically generated

**Chart, scatter chart

Description automatically generated**

Array size: 2 million

Blue: thread = 2.

Yellow: threads = 4.

Gray: threads = 8.

**Conclusions:**

1. As the proportion of parallel sorted array increase, the sort time do not have obvious decrease.

2. The sort time decrease as the threads number increase.

**Experiment B:**

Sorting time decrease more significantly when the threads increased from 4 to 8, and time decrease less from 4 to 8 threads. This could happen because the optimal thread number is between 4 and 8. So, I will try thread is 6.

**Table

Description automatically generated**

**Observation:**

The sorting time is even a little less when using 6 threads (average is 75.652ms), comparing to threads = 8(average sorting time is 79.098).

**Conclusion 3:**

the optimal thread number in my computer is 6(8 is power of 2).

**Experiment C:**

Next step is to find the optimal cutoff. I will change the cutoff number(percentage), with same array size (2 million).

**Result:**

**Table

Description automatically generated**

**Chart, line chart

Description automatically generated**

Array size: 3.5 million

Blue: thread = 2.

Yellow: threads = 4.

Gray: threads = 6.

**Observation:**

From above table and chart, we can see the sort time decrease rapidly at 0.25, which is 25% cutoff.

**Conclusion 4:**

To be usable, the least cutoff is around 25% array size, or 500,000 when array size is 2,000,000.

**Experiment D:**

Next step is to confirm previous conclusions about optimal thread number and cutoff with larger array size, 3,500,000. This time, I will use same threads: 2, 4, 6, and 8. Also, I will use wider cutoff range to find the optimal cutoff range.

**Result:**

**Table

Description automatically generated**

**Chart, line chart

Description automatically generated**

Array size: 3.5 million

Blue: thread = 2.

Yellow: threads = 4.

Gray: threads = 6.

Orange: threads = 8.

**Observation:**

Sort time decrease rapidly if cutoff is 25% array size, and sort time goes back up at 50% for higher threads number(4, 6 and 8).

Sort time reach the min when thread is 6, not 8.

**Conclusion 5:**

Optimal cutoff is between 25-50% array size.

**Some snapshots of results:**

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated