Shibo Lu (001548075)

**Program Structures & Algorithms**

**Fall 2021**

**Assignment No. 5**

**Task:**

Your task is to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.

**Part 1:**

**Requirement:**

1. A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then you should use the system sort instead.

**Using different cutoff to test parSort in main() method:**

文本

描述已自动生成

**Part 1 Output:**

图片包含 表格

描述已自动生成表格

描述已自动生成

**Result:**

After using parSort to sort array with different ‘cutoff’, I found that the parSort much faster when the ‘cutoff’ between 9000-248000. When the ‘cutoff’ is 197000 parSort can sort 2000000 numbers in 46.7ms with 16 threads.

表格

描述已自动生成图形用户界面, 应用程序, 表格, Excel

描述已自动生成

**Part 2:**

**Requirement:**

1. Recursion depth or the number of available threads. Using this determination, you might decide on an ideal number (*t*) of separate threads (stick to powers of 2) and arrange for that number of partitions to be parallelized (by preventing recursion after the depth of *lg t* is reached).

**Using different thread number in Parsort() method:**

文本

描述已自动生成

**Output:**

图形用户界面, 文本, 应用程序

描述已自动生成

图表, 折线图

描述已自动生成

**Result:**

As we can see, the more threads we used the faster Parsort is. Because the CPU in my computer is Intel i7-11800H(8 Cores), 8 and 16 threads will be the most threads I can used.

**Git：**

<https://github.com/ShiboLu/INFO6205-Shibo-Lu/tree/main/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/par>

**Part 3:**

1. An appropriate combination of these.

**Relationship Conclusion:**

***When the ratio of cutoff and array sizes are between 0.004-0.124, the Parsort will have highest efficiency. With no more than 16 threads, the more thread we used the higher speed ParSort have. The prefer ratio is 0.08 (most stable value).***

**Evidence to support the conclusion:**

In part 2, I am sure that 16 threads will make Parsort fastest. Then I test Parsort in 16 threads with different cutoff and array sizes. First test is for specific cutoff of with different array sizes. It shows that the running time is relatively stable. In 1 million array size sample, the running time are lower when ratio of cutoff and array sizes are between 0.005-0.12. Second test is for specific proportion of cutoff and array size with different array sizes. In this test, I found that ParSort will more efficient when the ratio of cutoff and array sizes are between 0.004-0.124. It also showed in result in part 1. The ratio of ‘cutoff’ between 9000-248000 and array size 2000000 almost equals ratio 0.004-0.124.

**Analysis：**

In parallel sort, we can always get a result: more threads used higher efficiency sort have. But this efficiency will not more than the value when the saved time of data sort in parallel by each thread equals to the wasted time of data sent to each thread. The other side value appears when all the threads in the pool can be used. The ratio between these two values will always make the sort algorithm had high efficiency. This can be a great reason for the area I found above.

**Graphical Representation:**

图表, 折线图, 散点图

描述已自动生成

图表, 折线图

描述已自动生成

**Output:**

文本, 应用程序

描述已自动生成

**Git:**

<https://github.com/ShiboLu/INFO6205-Shibo-Lu/tree/main/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/par>