Danlin Lu (001567176)

**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.

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.
2. 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).
3. An appropriate combination of these.

You must prepare a report that shows the results of your experiments and draws a conclusion (or more) about the efficacy of this method of parallelizing sort. Your experiments should involve sorting arrays of sufficient size for the parallel sort to make a difference. You should run with many different array sizes (they must be sufficiently large to make parallel sorting worthwhile, obviously) and different cutoff schemes.

* **Relationship Conclusion:**

1. When cutoff ratio(cutoff/size) chosen is quite small, around 0.1, we do not need to use sort in parallel as line programming performs well. However, when the ratio gets larger, parallel sorting can be a good choice.
2. No matter what size is an array and no matter how many numbers of available threads, the roughly good range of the cutoff is from 0.15 to 0.5. And 0.3 seems a general good choice for the cutoff.
3. The better number of available threads depends on the size of arrays. For the arrays with large size, like 8 million, 8 threads can be a better choice of parallel sorting; while for that with small size, like 1 million, 16 threads can be a better choice.

* **Evidence to support the conclusion:**

1. **Output**

Array sizes chosen are: 1 million, 2 million, 3 million, 4 million, 5 million, 6 million, 7 million, 8 million.

Cutoff schemes are: 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.1.

Number of threads chosen are: 1 (non-parallel), 2, 4, 8, 16, 32, 64

As the trend is the same, I do not divide running time by 10.

Here is one sample with 8 threads and 1 million size to show my code can run successfully:

**图形用户界面, 文本, 应用程序

描述已自动生成**

1. **Graphical Representation**

**Firstly, I fix the number of threads to find good value of the cutoff for arrays with different sizes. Here are the results:**

**表格

描述已自动生成**

**图表, 折线图

描述已自动生成**

Under 64 threads, you can see that the cutoffs roughly from 0.15 to 0.5 achieve the best performance, more accurately, 0.3.

表格

描述已自动生成

图表, 折线图

描述已自动生成

Under 32 threads, you can see that the cutoffs roughly from 0.15 to 0.5 achieve the best performance, more accurately, 0.25.

表格

描述已自动生成

图表, 折线图

描述已自动生成

Under 16 threads, you can see that the cutoffs roughly from 0.15 to 0.5 achieve the best performance, more accurately, 0.3.

表格

描述已自动生成

图表, 折线图

描述已自动生成

Under 8 threads, you can see that the cutoffs roughly from 0.2 to 0.5 achieve the best performance, more accurately, 0.4.

表格

描述已自动生成

图表, 折线图

描述已自动生成

Under 4 threads, you can see that the cutoffs roughly from 0.25 to 0.5 achieve the best performance, more accurately, 0.3.

表格

描述已自动生成

图表, 折线图

描述已自动生成

Under 2 threads, you can see that the cutoffs roughly from 0.25 to 0.5 achieve the best performance, more accurately, 0.3.

In conclusion, the roughly good range of the cutoff is from 0.15 to 0.5. And 0.3 seems a general good choice for the cutoff.

Secondly, to find better number of available threads, I fix the size of array. And to test whether the number of threads is related to the size, I choose two sizes: 1 million and 8 million.

图表, 折线图

描述已自动生成The result of 1 million array:

We can find that 16 threads performs better than others.

图表, 折线图

描述已自动生成The result of 8 million array:

We can find that 8 threads performs better than others.

And from those two images, we can find that when cutoff is small enough, around 0.1, we do not need to choose parallel sorting.