Wei Chen(001562214)

Program Structures & Algorithms

Fall 2021

Assignment No.5

1. Task

Please see the presentation on Assignment on Parallel Sorting under the Exams. etc. module.

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.

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 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.

1. Relationship Conclusion:

In my laptop(cpu: 1.4 GHz four cores Intel Core i5), when the length of array equals 2,000,000.

when degree of parallelism(number of threads) equals 8, the time decreases obviously when the cutoff value > array.length / threadCount which is 250,000. As test more cutoff values I find that when the recursion depth equals two (uses two threads through first recursion and four threads through second recursion) which means the cutoff values between(500,000, 1,000,000], the time is the least.

When the degree of parallelism equals 16 the time decreases obviously when the cutoff value > array.length / threadCount. As test more cutoff values I find that when the recursion depth equals three (uses two threads through first recursion and four threads through second recursion and use eight threads through third recursion) whicn means the cutoff values between(250,000, 500,000], the time is the least.

When the degree of parallelism increases bigger and bigger the time don’t decrease obviously when the cutoff value > array.length / threadCount. As test more cutoff values I find that when the recursion depth equals bigger than two which means the cutoff value betweens(array.length / threadCount, 1,000,000],the time will decrease faster. I think that’s because the time to create more threads might be offset the efficiency to parallelism.

Use ForkJoinPool to new a myPool and set its threadCount then pass the myPool as parameter to sort() and parsort() so that the CompletatbleFuture.supplyAsync() can use the myPool. Every time the parsort() been called it create a new thread through myPool instead of system common pool and the number of thread that can be parallel used is no more than the threadCount set. For example, when the length of array is 2,000,000,when we set the threadCount to 2, and cutoff value firstly equals the array size and then become half of itself. First time we execute main function we call the sort() and since the array size equals cutoff it becomes two 1,000,000 size array and call the parsort() separately and needs to new two threads, but then two parsort() call the sort() separately and the each array size is less than cutoff, so it just come to use system sort rather than call the parsort() again. Thus means use one level of recursion and then use system sort.

1. Evidence to support the conclusion:

Output:

图形用户界面, 文本

描述已自动生成

电脑萤幕的截图

描述已自动生成

电脑萤幕的截图

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

图形用户界面, 文本

描述已自动生成

电脑的截图

描述已自动生成

图形用户界面, 文本

描述已自动生成

电脑的截图

描述已自动生成

图形用户界面, 文本

描述已自动生成

Graphical Representation: