Assignment 4 Parellel Sorting

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.

There is a *Main* class and the *ParSort* class in the *sort.par* package of the INFO6205 repository. The *Main* class can be used as is but the *ParSort* class needs to be implemented where you see "TODO..." [it turns out that these TODOs are already implemented].

Unless you have a good reason not to, you should just go along with the Java8-style future implementations provided for you in the class repository.

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.

1. Cutoff choice

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 50000 | 100000 | 150000 | 200000 | 250000 | 300000 | 350000 | 400000 | 450000 | 500000 |
| 100000 | 156 | 69 | 51 | 47 | 75 | 51 | 47 | 47 | 47 | 46 |
| 200000 | 255 | 56 | 70 | 69 | 138 | 131 | 114 | 116 | 115 | 116 |
| 400000 | 249 | 98 | 150 | 101 | 176 | 148 | 147 | 146 | 147 | 242 |
| 600000 | 337 | 139 | 132 | 162 | 203 | 160 | 230 | 228 | 232 | 227 |
| 800000 | 401 | 196 | 200 | 176 | 213 | 218 | 215 | 212 | 308 | 304 |
| 1000000 | 506 | 200 | 212 | 265 | 222 | 264 | 269 | 270 | 268 | 267 |

The row represents the size of the array and the column represents the number of the cutoff. The thread count is set to be 32

We can tell that the time consumed least around 10000. But this number can vary from different thread counts. So we can conclude that a certain thread count should have a ideal cutoff period.

1. Thread Count

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| 100000 | 132 | 70 | 65 | 26 | 27 | 26 | 25 | 26 | 26 | 23 |
| 200000 | 171 | 119 | 72 | 48 | 40 | 39 | 42 | 41 | 39 | 45 |
| 400000 | 326 | 140 | 177 | 121 | 130 | 84 | 85 | 87 | 80 | 84 |
| 600000 | 436 | 261 | 213 | 179 | 119 | 125 | 126 | 126 | 124 | 124 |
| 800000 | 368 | 230 | 297 | 260 | 231 | 161 | 166 | 174 | 186 | 197 |
| 1000000 | 469 | 367 | 304 | 323 | 243 | 207 | 210 | 216 | 232 | 231 |

The row represents the size of the array and the column represents the count of the threads. The cutoff is set to be 50000

We can tell that in all circumstances the time consumed will drop when we use more threads. But even though the thread counts increase exponentially. The time consumed decrease very slowly. We can conclude that there is a marginal effect.

1. Combination choice

In conclusion I will choose the combination of 8 threads and 100000 to be the cutoff. Here is the result

电脑萤幕的截图

描述已自动生成

It’s a pretty good result.

Other screenshot:

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

描述已自动生成电脑的截图

描述已自动生成电脑的截图

描述已自动生成电脑的截图

描述已自动生成电脑的截图

描述已自动生成电脑的截图

描述已自动生成电脑的截图

描述已自动生成