Jiao Gong(001561450)

Program Structures &Algorithms

Fall2021

Assignment No.5

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.

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.
For varying the number of threads available, you might want to consult the following resources:

- https://www.callicoder.com/java-8-completablefuture-tutorial/#a-note-about-executor-and-thread-pool (Links to an external site.)

- https://stackoverflow.com/questions/36569775/how-to-set-forkjoinpool-with-the-desired-number-of-worker-threads-in-completable (Links to an external site.)

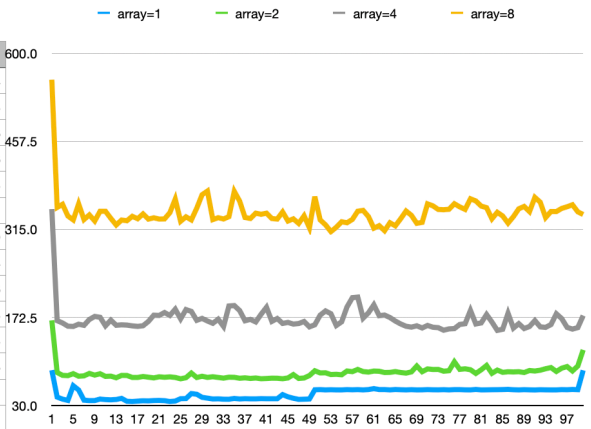
Good luck and enjoy.

Experiments

1. A cutoff

Different Cutoff

Cutoff	array=1	array=2	array=4	array=8
1	87.0	167.7	347.8	557.4
2	43.4	83.2	167.2	350.5
3	40.1	78.6	163.4	355.6
4	38.0	78.1	158.2	336.6
5	61.8	81.3	157.8	330.3
6	54.9	77.2	161.4	356.2
7	38.4	77.8	159.6	330.9
8	37.7	81.7	169.1	338.4
9	37.8	79.0	174.0	328.6
10	40.2	81.2	172.6	344.3
11	39.3	76.9	158.8	344.3
12	39.0	77.3	168.0	332.8
13	39.5	74.7	159.7	322.1
14	41.3	78.5	160.4	330.1
15	36.6	78.4	160.0	329.0
16	36.2	75.2	158.7	336.1
17	36.9	75.2	157.9	332.2
18	37.4	76.7	158.9	340.0
19	37.2	75.6	165.9	331.6
20	37.8	74.5	176.0	333.7



(Figure 1.different cutoff value with different array length)

Figure 1 is the experimental data of cutoff. The base of cutoff is 10,000. From the statistical data chart, we can see that as the cuffoff increases to about 40,000, the time spent to sort arrays of different lengths decreases significantly, but then with the cutoff. Increasing, the time consumed for sorting random arrays of the same length has not changed significantly, and it is basically a horizontal line. Therefore, when the number of threads is fixed, the recommended cutoff value is 40,000.

2. available threads

Table 1

Threadnum	array=1	array=2	array=4	array=8
1	32	36	98	236
2	57	39	54	104
4	52	78	54	116
8	72	89	61	112
16	75	106	56	111
32	98	36	33	107
64	79	23	38	84
128	74	34	41	85
256	83	23	50	92
512	91	21	46	87

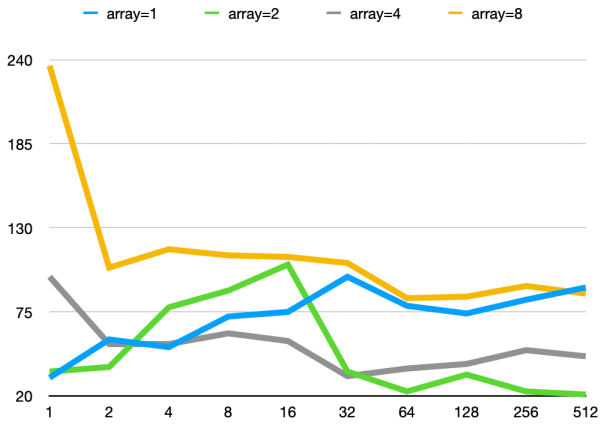
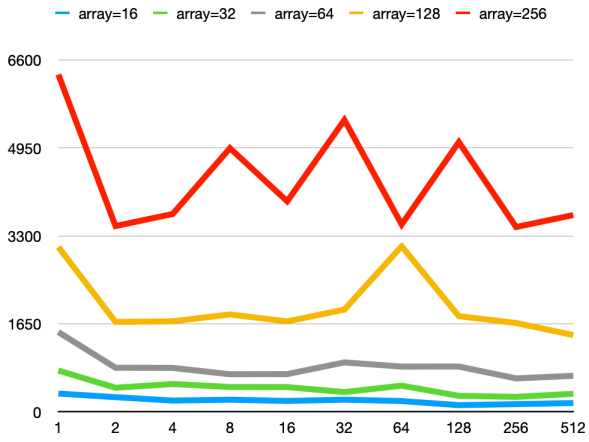


Table 1-1

Threadnum	array=16	array=32	array=64	array=128	array=256
1	344	776	1498	3094	6329
2	276	453	828	1688	3486
4	212	523	825	1699	3713
8	229	467	706	1828	4947
16	205	465	709	1697	3950
32	229	372	928	1920	5476
64	203	492	850	3106	3516
128	126	301	851	1796	5060
256	145	282	628	1667	3468
512	166	339	677	1441	3692



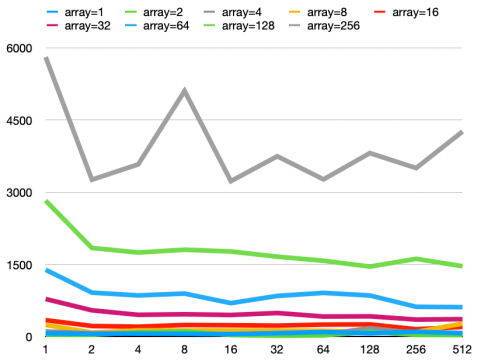
(Figure 2.different thread numbers value with different array length)

Figure 2 shows the fixed cutoff value, cutoff=40000, and the base of the array length is 250,000. The table data is the sorting time required for different numbers of threads to sort arrays of the same length. It can be seen that when the length of the array is large, as the number of threads increases, the required time decreases sharply, but when the number of threads exceeds a certain value, Performance declines, and the time required for sorting may increase. I guess the reason is that in addition to multi-threading will increase the creation overhead, there is also the overhead of switching context. The optimal value of the number of threads is not easy to determine. When the array length is small (length<250,000), the advantage of multithreading is not obvious. When the array length is large (length>=2,000,000), the number of threads is 4, 8, 16 may be the best choice.

3.An appropriate combination

cutoff=10,000

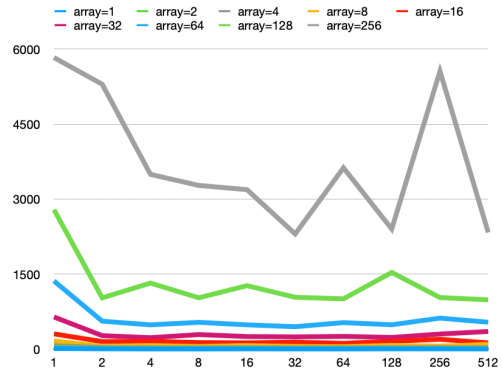
Threadnum	array=1	array=2	array=4	array=8	array=16	array=32	array=64	array=128	array=256
1	70	39	108	243	343	782	1390	2827	5810
2	59	37	68	85	221	547	915	1843	3262
4	64	89	73	130	207	453	856	1748	3581
8	58	108	68	129	242	465	895	1806	5106
16	54	33	42	128	237	450	696	1770	3232
32	72	20	56	125	229	491	846	1662	3749
64	96	26	46	122	250	418	909	1577	3269
128	69	98	176	93	255	422	855	1456	3815
256	102	35	71	96	154	354	620	1617	3504
512	72	33	57	271	201	364	613	1463	4263



(Figure 3.cutoff=10,000 and different thread numbers value with different array length)

cutoff=20,000

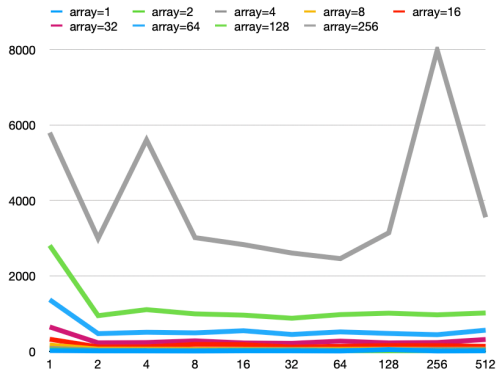
Threadnum	array=1	array=2	array=4	array=8	array=16	array=32	array=64	array=128	array=256
1	15	36	70	165	307	645	1365	2790	5828
2	9	15	32	59	153	272	561	1027	5294
4	7	13	27	73	158	231	487	1322	3494
8	6	13	26	54	134	293	538	1028	3273
16	8	14	30	73	127	254	487	1270	3189
32	7	13	25	55	141	247	451	1038	2304
64	6	13	27	50	118	257	532	1007	3625
128	9	14	30	76	159	236	489	1534	2403
256	8	15	29	62	199	304	622	1033	5554
512	7	13	28	96	127	355	540	987	2333



(Figure 4.cutoff=20,000 and different thread numbers value with different array length)

cutoff=40,000

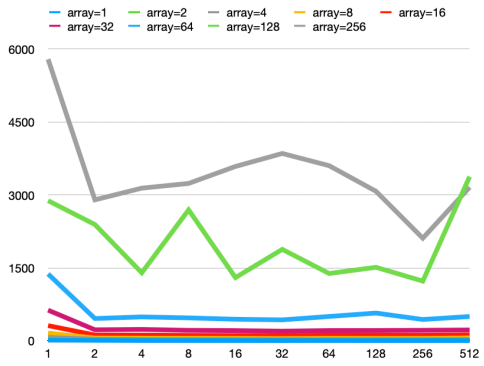
Threadnum	array=1	array=2	array=4	array=8	array=16	array=32	array=64	array=128	array=256
1	16	38	72	164	318	644	1366	2805	5797
2	9	15	28	55	114	222	466	943	2991
4	8	14	26	53	117	230	503	1103	5604
8	5	12	25	50	180	275	486	990	3013
16	11	22	33	49	174	219	543	957	2828
32	10	20	39	47	111	209	444	875	2604
64	6	13	25	50	129	268	513	971	2456
128	43	13	26	53	152	220	471	1009	3141
256	6	13	43	50	149	230	438	965	7986
512	11	20	41	52	130	311	557	1015	3553



(Figure 5.cutoff=40,000 and different thread numbers value with different array length)

cutoff=80,000

Threadnum	array=1	array=2	array=4	array=8	array=16	array=32	array=64	array=128	array=256
1	15	36	68	160	313	631	1375	2882	5790
2	11	18	33	56	119	227	459	2391	2899
4	7	15	27	54	117	235	492	1399	3139
8	7	12	25	53	112	216	472	2692	3236
16	7	11	24	51	110	211	444	1300	3585
32	6	10	23	46	112	199	431	1881	3852
64	7	12	24	52	115	212	501	1384	3601
128	6	12	24	49	115	213	571	1512	3075
256	6	12	24	50	114	217	439	1228	2110
512	6	13	28	59	122	224	500	3376	3158



(Figure 6.cutoff=80,000 and different thread numbers value with different array length)

From Figure 3, Figure 4, Figure 5, Figure 6, we observe the experimental data of different cutoffs and thread numbers, we can find that when the array length is small (length<2,000,000), cutoff=4000, ThreadNumber=8, which is the optimal combination , When the length of the array is large (length>=2,000,000), cutoff=4000, ThreadNumber=16, it is the optimal combination.