Program Structures and Algorithms

Spring 2023(SEC – 8)

NAME: Natarajan Lekshmi Narayana Pillai

NUID: 002766033

Assignment 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.

Relationship Conclusion:

Cut-Off Ratio = (Cut-Off value/Size of the array)

The cut-off ratio refers to the point at which the algorithm switches from parallel to sequential processing. A smaller cut-off ratio means that the algorithm will switch to sequential processing for smaller problems, while a larger cut-off ratio means that the algorithm will continue to use parallel processing for larger problems.

Part 1.

According to the data observed in the trials, we can see that for a fixed array size, the time required typically gets shorter as the cut-off ratio gets higher. This is due to the algorithm performing more work in parallel and switching to sequential processing less frequently as the cut-off ratio rises.

However, the array size also affects the relationship between the cut-off ratio and the processing time. Although the time required often reduces as the cut-off ratio rises for lower array sizes, the relationship is more complicated for higher array sizes. As seen in the data, as the cut-off ratio rises, the processing time may initially decline, but for higher cut-off ratios, it may rise once again.

Part 2.

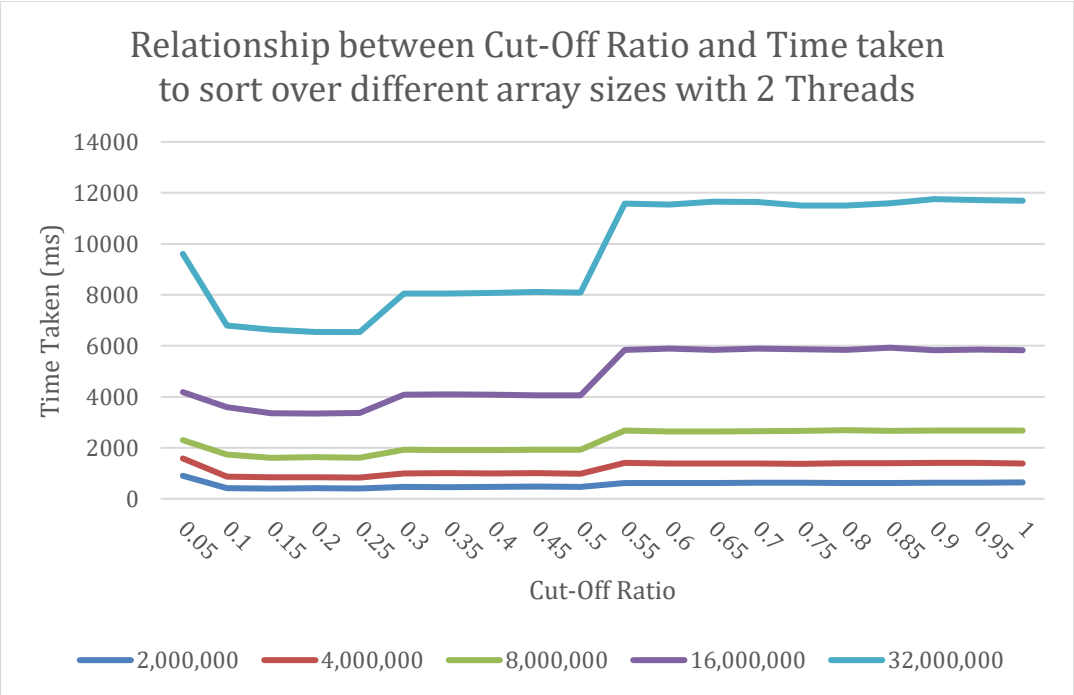
We can see from the data that the link between Cut-Off Ratio and execution time is not straightforward, and that it also relies on the number of threads being used. For instance, at a Cut-Off Ratio of 0.25, going from 4 to 8 threads results in a decrease in execution time, but adding more threads results in a performance decrease.

Evidence to support that conclusion:

Part 1

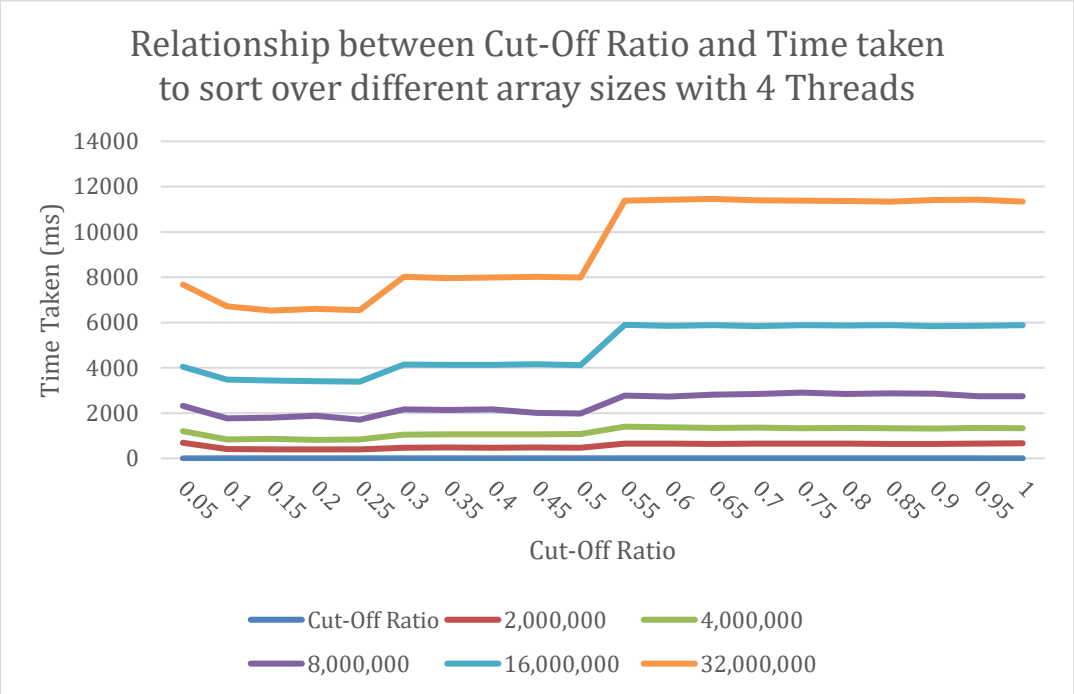
Threads = 2

Cut-Off Ratio	2,000,000	4,000,000	8,000,000	16,000,000	32,000,000
0.05	904	1584	2299	4180	9603
0.1	412	865	1734	3589	6802
0.15	400	839	1607	3359	6631
0.2	415	842	1631	3344	6541
0.25	404	833	1615	3373	6539
0.3	466	1000	1921	4084	8051
0.35	453	1007	1909	4099	8049
0.4	464	994	1918	4078	8076
0.45	487	1005	1922	4060	8118
0.5	468	980	1929	4061	8084
0.55	622	1404	2674	5847	11576
0.6	621	1386	2640	5892	11544
0.65	622	1386	2646	5842	11651
0.7	637	1381	2647	5886	11638
0.75	636	1375	2667	5864	11505
0.8	622	1396	2693	5845	11503
0.85	621	1400	2663	5928	11590
0.9	629	1410	2672	5830	11756
0.95	631	1407	2680	5853	11719
1	638	1388	2680	5832	11691



Threads = 4

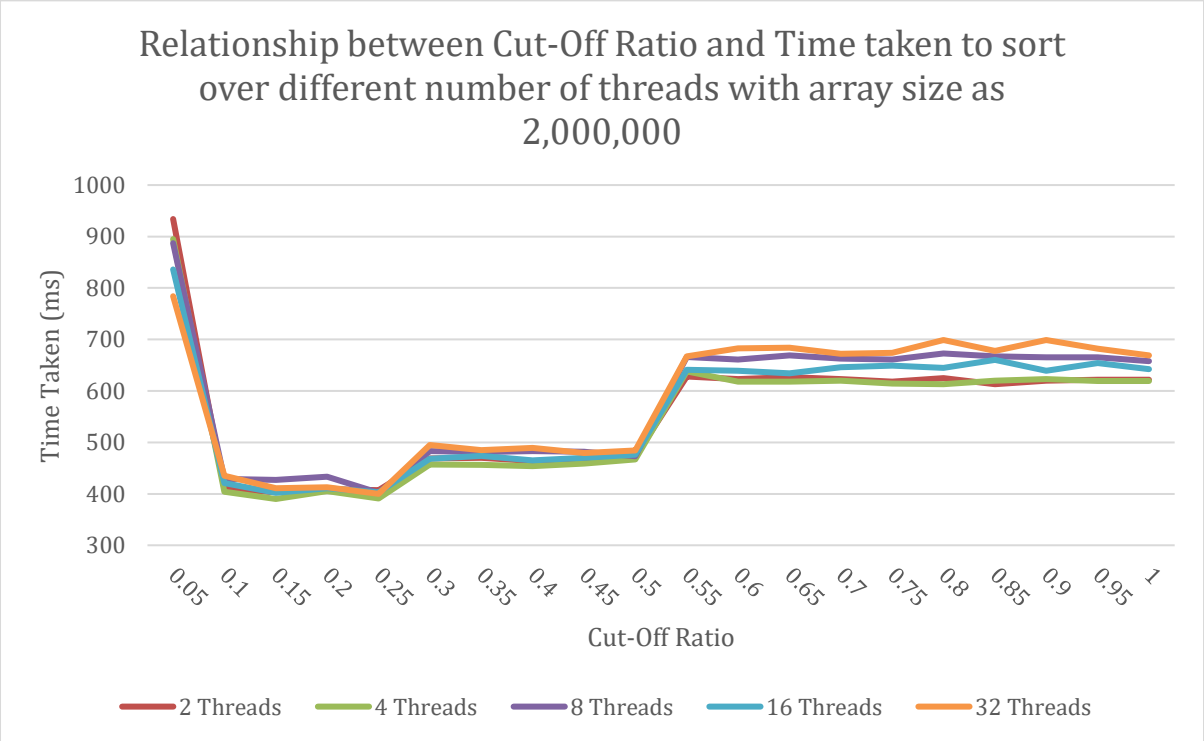
Cut-Off Ratio	2,000,000	4,000,000	8,000,000	16,000,000	32,000,000
0.05	691	1204	2317	4042	7677
0.1	409	842	1770	3475	6723
0.15	402	861	1795	3438	6527
0.2	397	817	1890	3410	6601
0.25	399	832	1711	3390	6544
0.3	472	1051	2162	4147	8023
0.35	478	1057	2139	4137	7955
0.4	471	1064	2161	4130	7996
0.45	483	1067	2016	4155	8022
0.5	475	1077	1988	4114	7995
0.55	649	1402	2772	5897	11383
0.6	652	1380	2731	5861	11418
0.65	645	1349	2821	5877	11459
0.7	648	1357	2842	5845	11401
0.75	647	1339	2906	5878	11374
0.8	650	1340	2840	5872	11366
0.85	646	1334	2875	5878	11345
0.9	646	1316	2853	5834	11407
0.95	650	1342	2740	5860	11424
1	663	1334	2740	5880	11334



Part 2

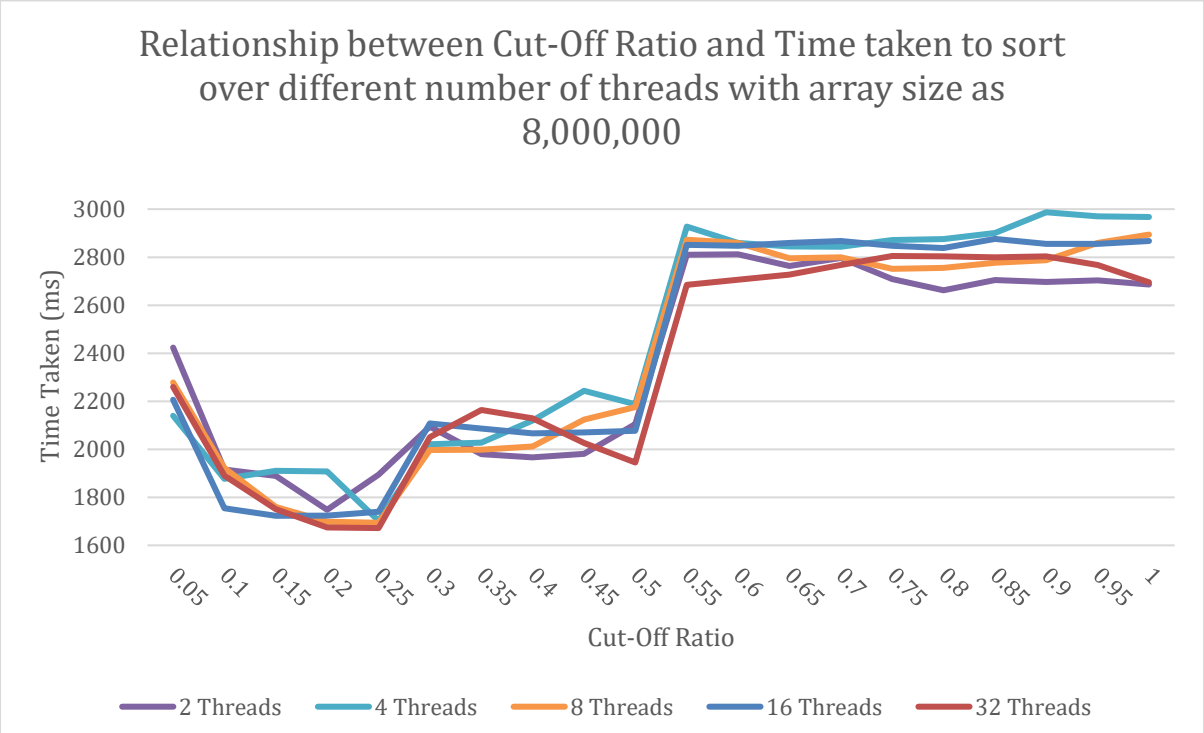
Array Size = 2,000,000

Cut-Off Ratio	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads
0.05	934	895	887	836	784
0.1	410	404	429	421	435
0.15	403	390	427	402	411
0.2	409	405	433	411	413
0.25	407	391	402	403	400
0.3	469	457	483	468	495
0.35	470	456	481	474	485
0.4	464	454	483	465	489
0.45	465	459	482	470	479
0.5	473	467	473	476	484
0.55	628	638	666	641	667
0.6	623	618	661	639	683
0.65	627	618	669	634	684
0.7	623	620	663	646	672
0.75	618	614	661	649	674
0.8	625	613	673	645	699
0.85	613	620	667	660	678
0.9	620	623	665	639	699
0.95	622	619	665	654	682
1	622	619	658	642	669



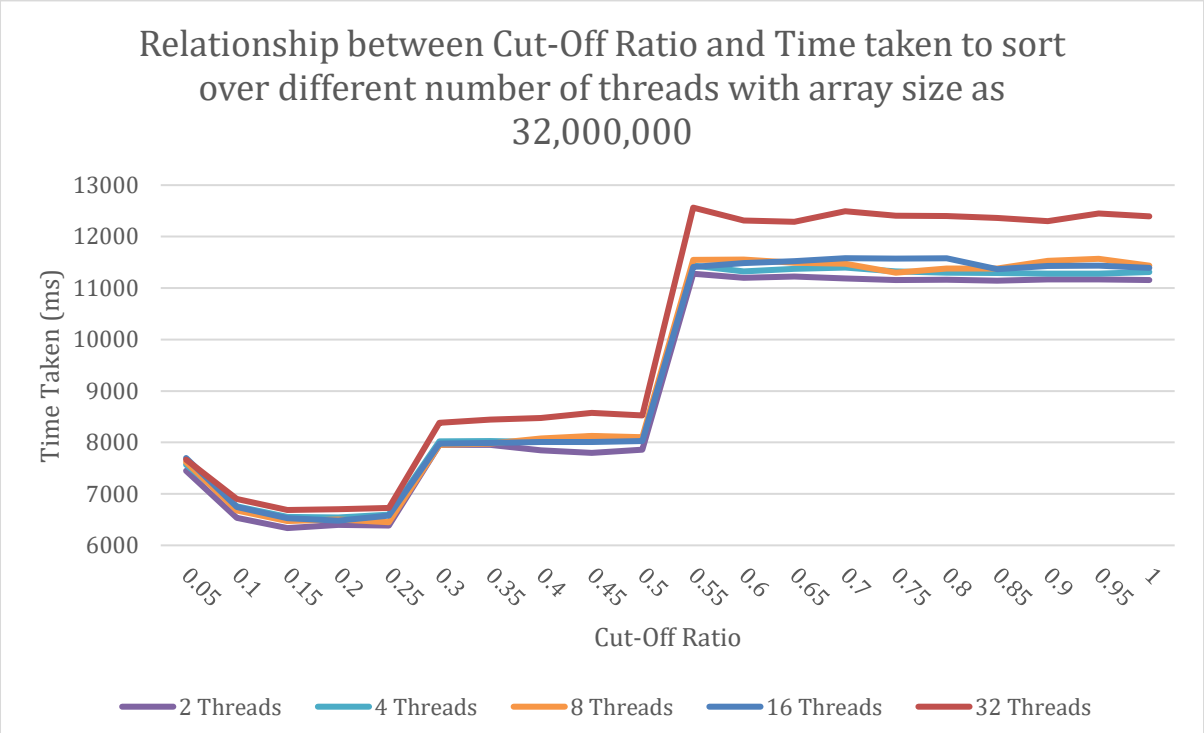
Array Size = 8,000,000

Cut-Off Ratio	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads
0.05	2424	2140	2278	2206	2260
0.1	1916	1877	1923	1754	1891
0.15	1889	1911	1760	1723	1751
0.2	1748	1908	1698	1724	1674
0.25	1895	1702	1694	1740	1672
0.3	2094	2021	1997	2108	2050
0.35	1980	2028	1999	2087	2164
0.4	1966	2120	2012	2066	2129
0.45	1981	2243	2123	2070	2026
0.5	2105	2187	2176	2077	1945
0.55	2810	2927	2873	2852	2685
0.6	2812	2860	2860	2847	2706
0.65	2764	2845	2795	2860	2727
0.7	2797	2843	2800	2867	2767
0.75	2709	2872	2751	2847	2805
0.8	2662	2876	2755	2838	2803
0.85	2705	2901	2777	2876	2799
0.9	2697	2987	2787	2856	2803
0.95	2703	2970	2860	2856	2767
1	2686	2967	2894	2868	2696



Array Size = 32,000,000

Cut-Off Ratio	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads
0.05	7451	7564	7618	7697	7665
0.1	6532	6756	6680	6747	6903
0.15	6336	6555	6473	6527	6687
0.2	6399	6542	6501	6480	6703
0.25	6386	6597	6449	6576	6726
0.3	7952	8020	7953	7978	8382
0.35	7955	8027	7980	7987	8442
0.4	7846	8006	8075	8010	8475
0.45	7794	8120	8126	8009	8572
0.5	7858	8061	8104	8027	8522
0.55	11278	11437	11545	11413	12563
0.6	11197	11324	11552	11486	12315
0.65	11221	11371	11489	11520	12288
0.7	11186	11398	11475	11580	12492
0.75	11156	11322	11297	11573	12408
0.8	11159	11301	11381	11577	12397
0.85	11145	11295	11381	11370	12364
0.9	11170	11282	11528	11430	12301
0.95	11167	11280	11568	11436	12450
1	11158	11313	11434	11390	12391



Output Screenshot:

All screenshots of the various outputs are provided in the Output folder in the repository under Assignment 5.