Assignment- 5 Parallel Sorting

10.Nov.2021

Thejas Ganjigunte Ramesh
NU ID : 001586113

Problem Statement

Implementing a parallel sorting algorithm such that each partition of the array is sorted in parallel. Considering two different schemes for deciding whether to sort in parallel.

1. A cutoff (defaults to, say, 1000) which will update according to the first argument in the command line when running. To experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then system sort should be used 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.

Method of experimentation

1. To find the right number of threads or depth of recursion for maximum efficiency with least sorting time.
2. To find the best cut off limit for any array size.

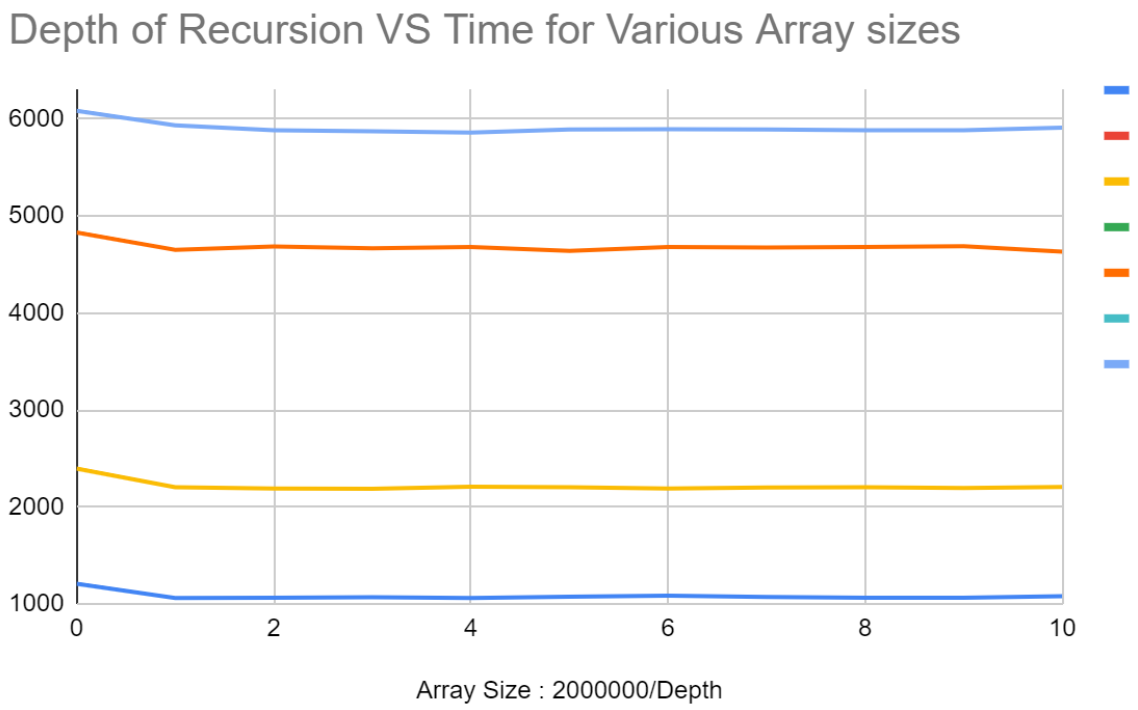
All the experiments were done on a windows machine with a x86 CPU of 8 cores at base clock of 3.2 GHZ and 16 GB of memory.

Finding the most optimal number of threads/depth of recursion for parallel sorting

By making changes to the thread pool count and the number of times recursive parallel sorting calls, I limited the number of threads created while keeping the cutoff constant at the value 1 so it doesn't affect our results. Below are the observations for various array sizes.

Array Size : 2000000		Array Size : 4000000		Array Size : 8000000		Array Size : 10000000	
Depth	Time	Depth	Time	Depth	Time	Depth	Time
0	1207	0	2396	0	4831	0	6086
1	1061	1	2203	1	4652	1	5938
2	1063	2	2190	2	4688	2	5885

3	1068	3	2186	3	4667	3	5876
4	1060	4	2209	4	4681	4	5861
5	1075	5	2202	5	4641	5	5893
6	1084	6	2190	6	4682	6	5896
7	1070	7	2200	7	4676	7	5895
8	1064	8	2203	8	4681	8	5887
9	1064	9	2195	9	4690	9	5887
10	1080	10	2206	10	4634	10	5913



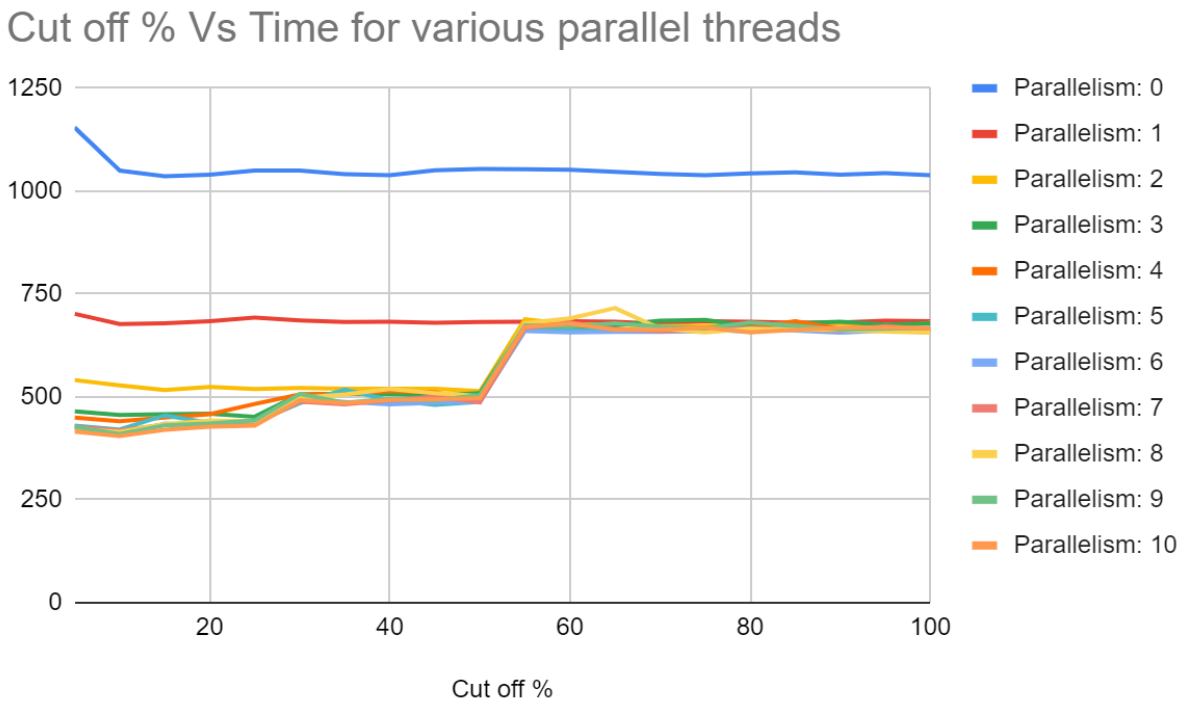
We can observe from the obtained data and the subsequent graph that the least time taken for sorting is around **4 recursive depth or 16 threads.** Any more threads doesn't seem to have any significant impact and might even consume more time in creation and processing of the threads itself.

Finding the most optimal cut off percentage for any array size for parallel sorting

By making changes to the cut off value and for each level of recursion and also for each size of the array, I obtained the following values and subsequent graphs for different array sizes.

Array Size : 2,000,000

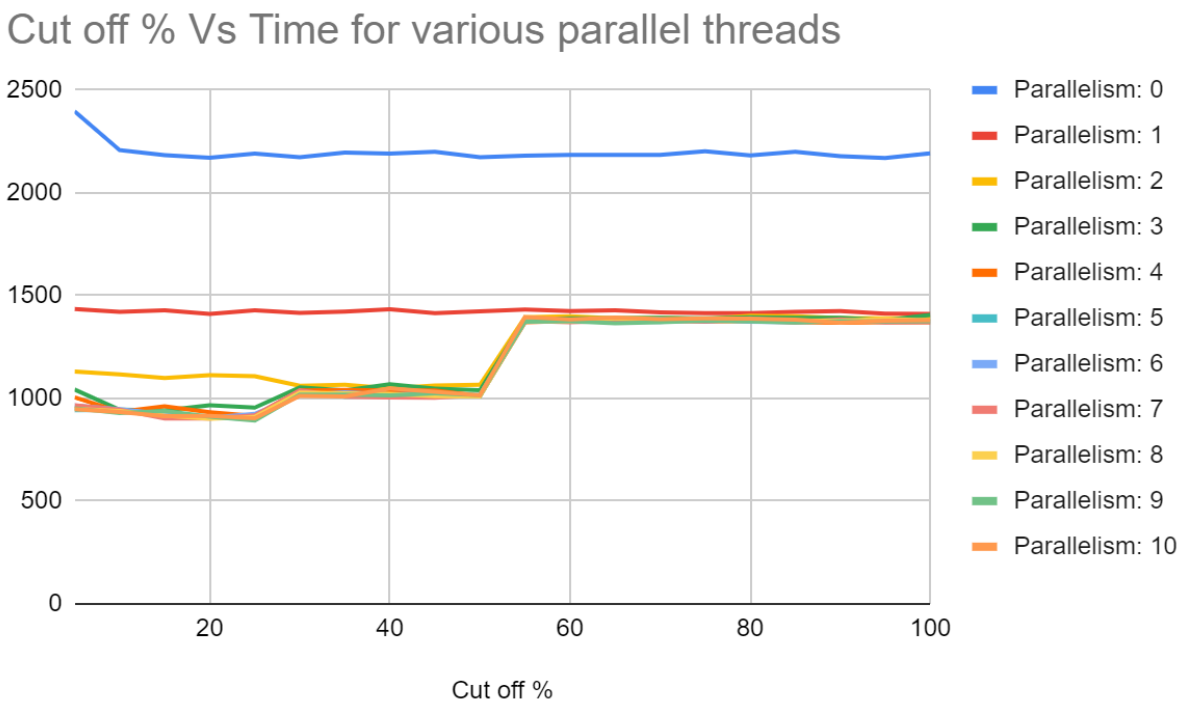
Cut off %	Parallelism: 0	Parallelism: 1	Parallelism: 2	Parallelism: 3	Parallelism: 4	Parallelism: 5	Parallelism: 6	Parallelism: 7	Parallelism: 8	Parallelism: 9	Parallelism: 10
5	1155	702	541	465	450	430	428	428	422	427	416
10	1050	677	528	456	441	421	407	418	415	410	405
15	1036	679	517	458	450	455	435	433	434	431	420
20	1040	684	524	459	458	437	431	433	442	435	428
25	1050	693	519	452	483	438	440	436	441	443	430
30	1050	686	522	506	506	485	496	488	498	507	492
35	1041	682	520	508	507	518	488	482	506	486	485
40	1039	683	520	507	514	492	482	492	519	496	493
45	1051	680	520	503	511	481	486	501	509	492	494
50	1054	682	514	509	493	488	489	487	502	504	497
55	1053	683	689	677	664	663	660	667	679	674	670
60	1052	684	677	669	663	662	657	681	691	672	678
65	1047	683	676	674	662	662	658	665	716	679	663
70	1042	678	672	685	661	661	658	660	666	671	663
75	1039	684	673	687	664	664	660	661	657	667	669
80	1043	683	677	672	669	659	667	662	666	681	657
85	1046	680	670	679	684	669	661	665	663	673	663
90	1040	681	679	683	666	664	656	661	663	664	668
95	1044	685	667	676	664	665	661	672	659	666	669
100	1039	684	680	679	660	663	657	665	656	668	667



Array Size: 4,000,000

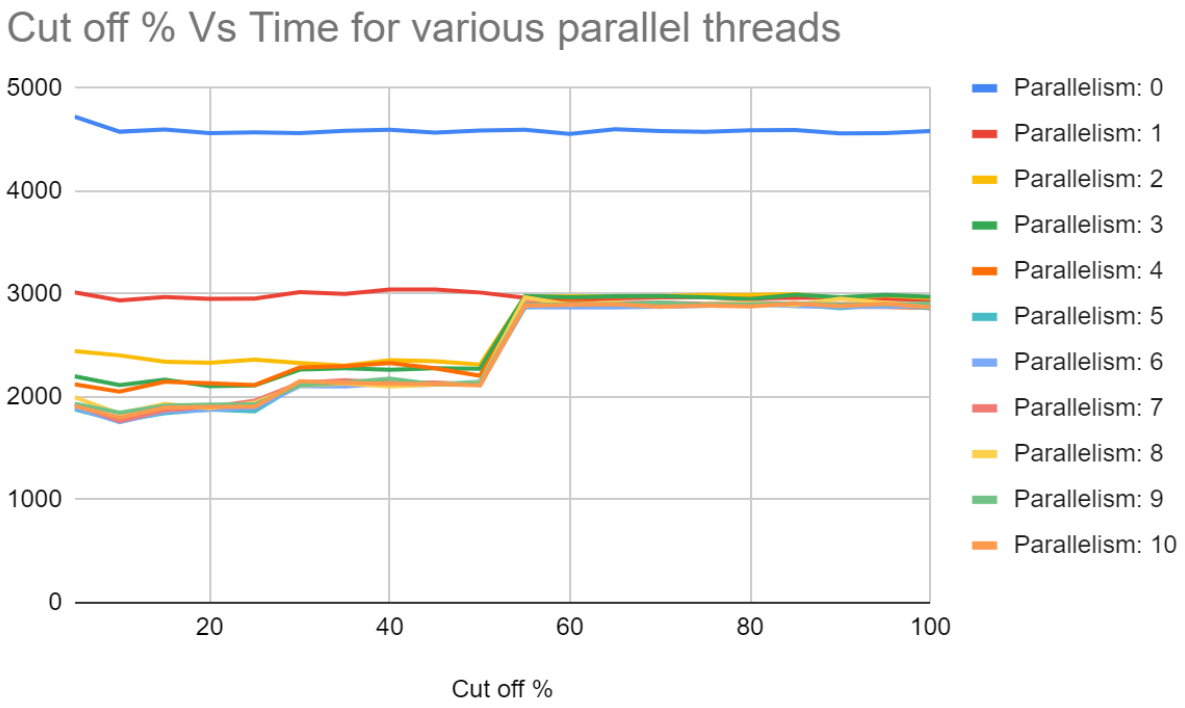
Cut off %	Parallelism: 0	Parallelism: 1	Parallelism: 2	Parallelism: 3	Parallelism: 4	Parallelism: 5	Parallelism: 6	Parallelism: 7	Parallelism: 8	Parallelism: 9	Parallelism: 10
5	2397	1435	1131	1043	1005	942	965	966	949	952	947
10	2208	1421	1117	945	935	944	948	945	939	929	934
15	2183	1429	1099	941	961	924	924	902	916	939	914
20	2171	1411	1113	967	933	908	906	902	901	911	915
25	2191	1428	1108	955	915	898	925	917	910	892	903
30	2174	1416	1061	1053	1038	1029	1016	1039	1028	1018	1012
35	2196	1422	1066	1039	1041	1008	1030	1008	1024	1018	1009
40	2191	1434	1049	1069	1039	1012	1019	1005	1022	1014	1049
45	2200	1415	1063	1047	1038	1011	1025	1003	1011	1025	1032
50	2174	1423	1066	1040	1011	1011	1016	1016	1010	1016	1015
55	2181	1432	1395	1384	1371	1382	1374	1377	1375	1373	1396
60	2185	1425	1399	1390	1377	1376	1373	1370	1376	1375	1385

65	2185	1428	1390	1389	1370	1385	1376	1377	1390	1366	1393
70	2185	1418	1392	1392	1379	1388	1372	1376	1379	1370	1386
75	2203	1415	1386	1389	1376	1373	1384	1374	1380	1378	1389
80	2182	1415	1402	1394	1376	1389	1373	1381	1380	1374	1388
85	2200	1421	1399	1392	1370	1373	1370	1370	1379	1369	1383
90	2179	1425	1391	1392	1368	1377	1376	1385	1378	1376	1368
95	2170	1412	1390	1381	1370	1376	1372	1369	1387	1374	1378
100	2192	1411	1391	1406	1371	1374	1378	1371	1379	1376	1381



Array Size:8,000,000

Cut off %	Parallelism: 0	Parallelism: 1	Parallelism: 2	Parallelism: 3	Parallelism: 4	Parallelism: 5	Parallelism: 6	Parallelism: 7	Parallelism: 8	Parallelism: 9	Parallelism: 10
5	4725	3017	2445	2200	2122	1879	1893	1930	1998	1934	1912
10	4578	2938	2404	2115	2051	1766	1754	1764	1837	1847	1800
15	4600	2970	2343	2167	2148	1842	1858	1871	1932	1916	1895
20	4565	2953	2333	2104	2132	1877	1876	1904	1891	1925	1902
25	4571	2955	2363	2113	2116	1861	1892	1963	1944	1928	1905
30	4565	3019	2329	2266	2287	2126	2106	2136	2113	2116	2153
35	4588	3001	2304	2280	2298	2134	2101	2164	2129	2140	2137
40	4597	3045	2356	2264	2329	2111	2129	2138	2102	2179	2127
45	4569	3043	2348	2278	2278	2120	2130	2139	2121	2122	2125
50	4589	3015	2314	2274	2206	2147	2120	2119	2137	2132	2113
55	4597	2963	2976	2980	2910	2872	2872	2915	2968	2895	2887
60	4556	2953	2979	2969	2925	2889	2869	2904	2894	2900	2894
65	4602	2959	2977	2979	2909	2900	2869	2911	2919	2903	2901
70	4583	2968	2982	2982	2908	2874	2878	2918	2881	2917	2878
75	4577	2971	2991	2968	2892	2894	2884	2906	2901	2895	2887
80	4592	2959	2991	2950	2895	2891	2907	2919	2910	2899	2881
85	4594	2962	2998	2994	2909	2898	2883	2905	2893	2910	2907
90	4561	2965	2957	2968	2878	2863	2875	2902	2955	2896	2883
95	4564	2954	2985	2991	2874	2898	2877	2906	2915	2898	2904
100	4583	2944	2960	2974	2864	2862	2876	2898	2886	2907	2872



From the above output values and the graphs we can observe that the least time taken for various arrays and parallelism(depth of recursion) is **25% of the total array size.**

Conclusion

From the above observations we can conclude that:

1. The depth of recursion is only effective till **4 recursive depth or 16 threads**.
2. The optimal cut off point for various array sizes and threads is around **25% of the total array size.**