Work-stealing scheduler Benchmark results

Mirko Bez, Alessio Vitella

The Assignment

The assignment is to build a work-stealing scheduler, and a particular application (quick-sort), in Java. This technique is useful when there are a certain number of executors (Servers) and each one of them has a queue of Tasklets to do. When a server finishes its Tasklets, it can steal Tasklets from other servers' queue. Thus, simultaneously saving time and granting a better exploitation of the computer's hardware resources.

The Scheduler class has the following methods:

- `WorkStealingScheduler()`: Given a number N of servers it creates and starts N server threads that will execute the Tasklets;
- `spawn()`: Add a tasklet to the queue of the current server;
- `shutdown()`: Stops all the servers gracefully;
- `printStats()`: Print statistics about how many Tasklets have been spawned and how many Tasklets have been stolen;
- `waitForAll()`: Wait for all Tasklets to be executed.

Each Server thread owns a double ended queue of Tasklets. When a Tasklet is spawned it is added as the first element of the queue. If the queue is not empty the server executes the first Tasklet of the queue, otherwise it tries to steal a Tasklet taking the last element of other servers' queue.

The sorting algorithm is a Quicksort with a sequential cutoff, that is the number of elements under which the array is ordered sequentially through the standard `Arrays.sort()` method, which use a Dual-pivot quicksort algorithm.

Every run of the main routine of the program creates an array of a given size and fills it with random integer numbers. First of all the array is copied and sorted sequentially (using the standard Quicksort algorithm). Afterwards the scheduler is created and the array is sorted in parallel. The wall-clock time required for the two types of sortings is measured (using the java `System.nanoTime()` method) and compared.

The Experiment

We created a script that executes the program several times with different parameters (array length, number of servers, cutoff) in order to understand when our scheduler is more efficient than the sequential algorithm and to find the best configurations.
The parameters used in the tests are:

- **Array Length**: 10^4, 10^5, 10^6, 10^7, 10^8
- **Number of servers**: 1, 2, 4, 8
- **Cutoff:** the cutoff are obtained by dividing the array lengths by power of 2 until the cutoff becomes less than a fixed threshold (32).
- **Seed:** we used 5 different seeds (1241, 4238, 5249, 8282, 9636) for the random fill of the array (to try to make the running time independent from the random choice of the seed).

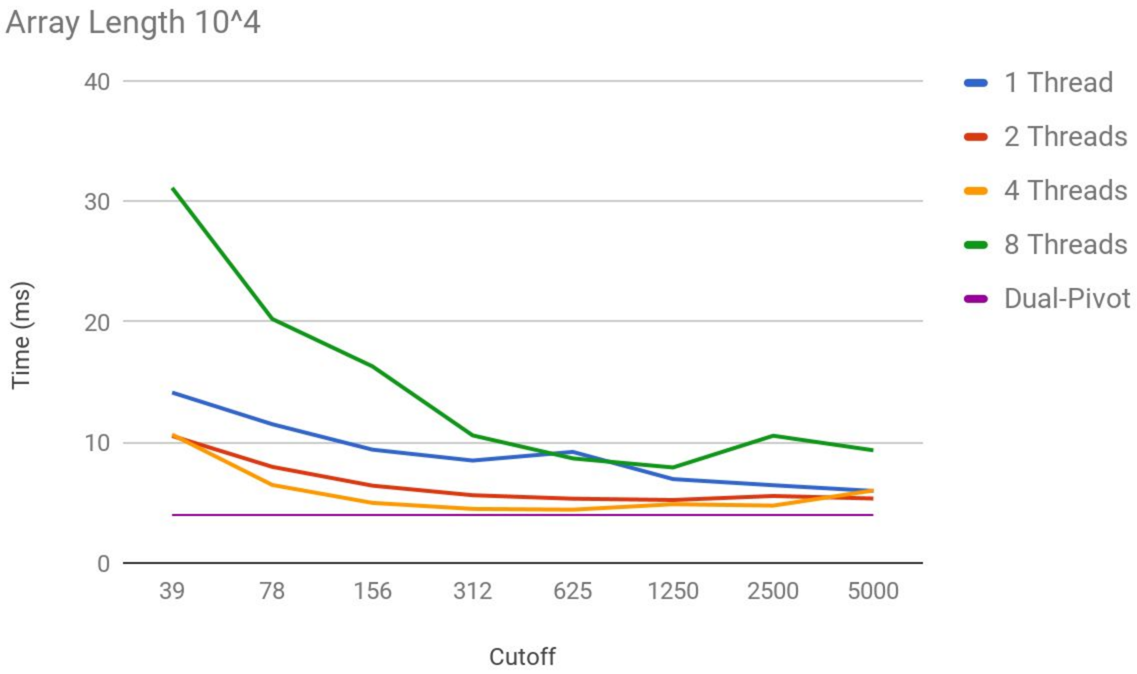
We compile the program using 3 different algorithms:

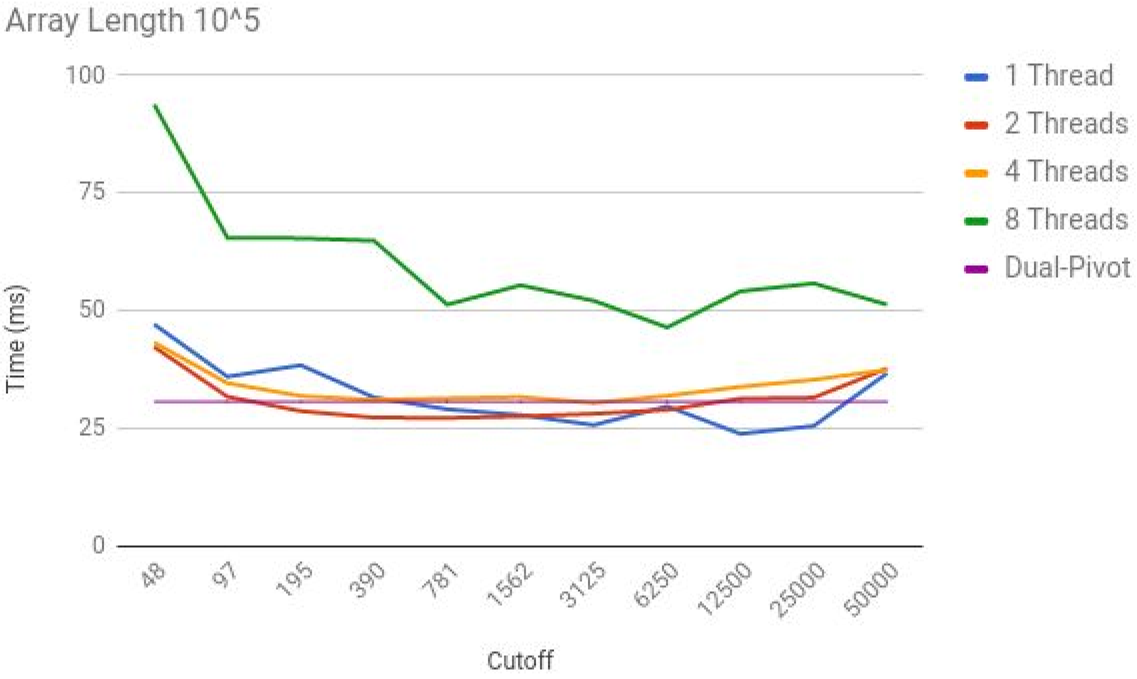
1. Sequential `Arrays.sort()` method (Dual-Pivot Quicksort Algorithm)
2. The work-stealing scheduler
3. The scheduler without stealing

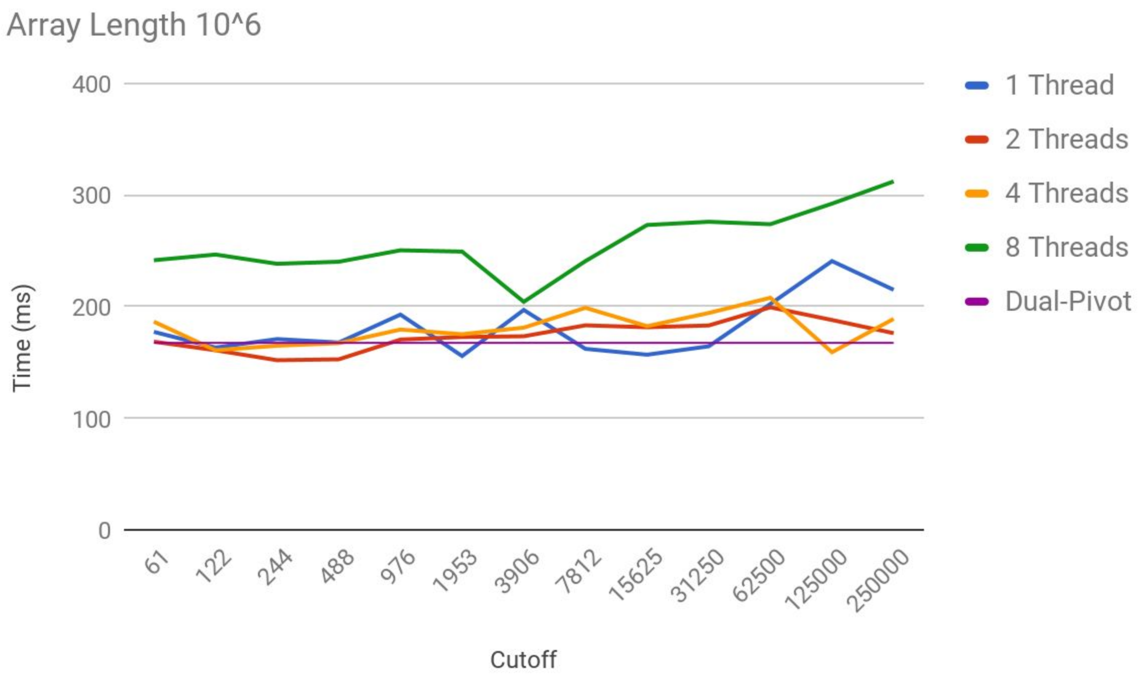
For every configuration we did 5 runs, thus obtaining 25 executions for each configuration.
The computer used for the test has an Intel CPU i7-3610QM 2.30 GHz with 4 cores (and 8 virtual threads), 8GB of RAM.

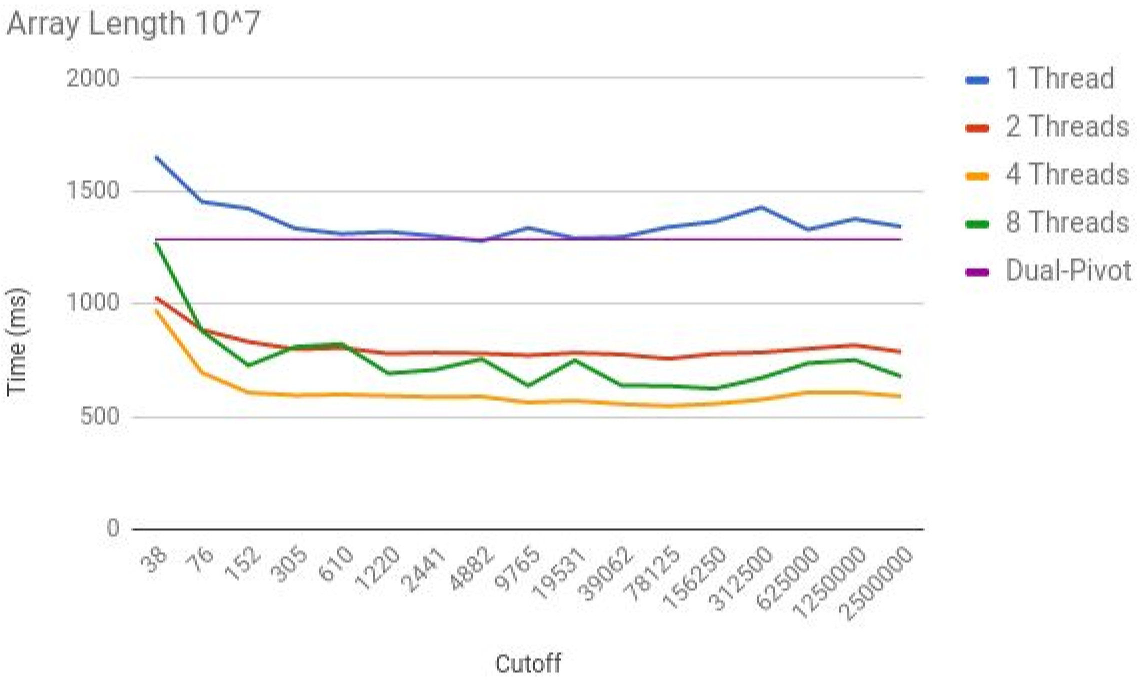
The results show that the Work-Stealing scheduler takes less time when the array length is bigger. It happens because, as we expected, the parallel algorithm has an overhead of operation due to thread management and coordination, so it usually takes more time unless the size of the array would be big enough. Anyhow the parallel scheduler without work-stealing is almost always slower than the one with work-stealing, indeed the work-stealing causes a considerable speed-up.
These results are summarized in the tables of the appendix, in which we marked with a light green color the fastest configuration for every array length and number of threads and with a light red color the fastest configuration of all.

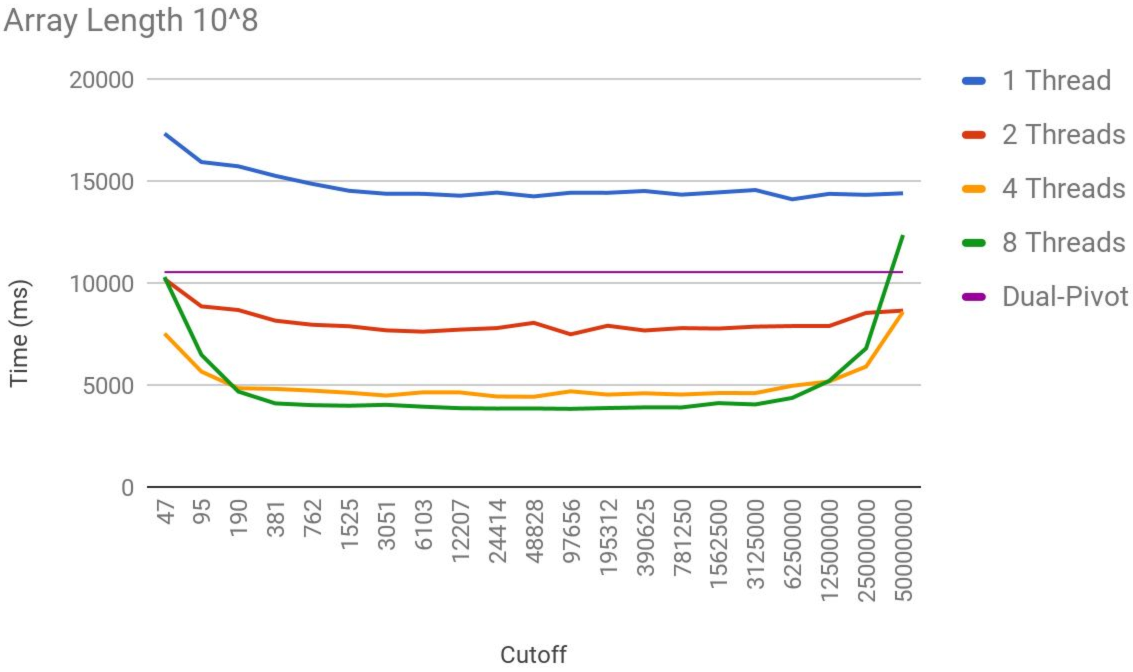
Finally, here we summarize the main results. For each array size and for each algorithm, we show the time in function of the cutoff:

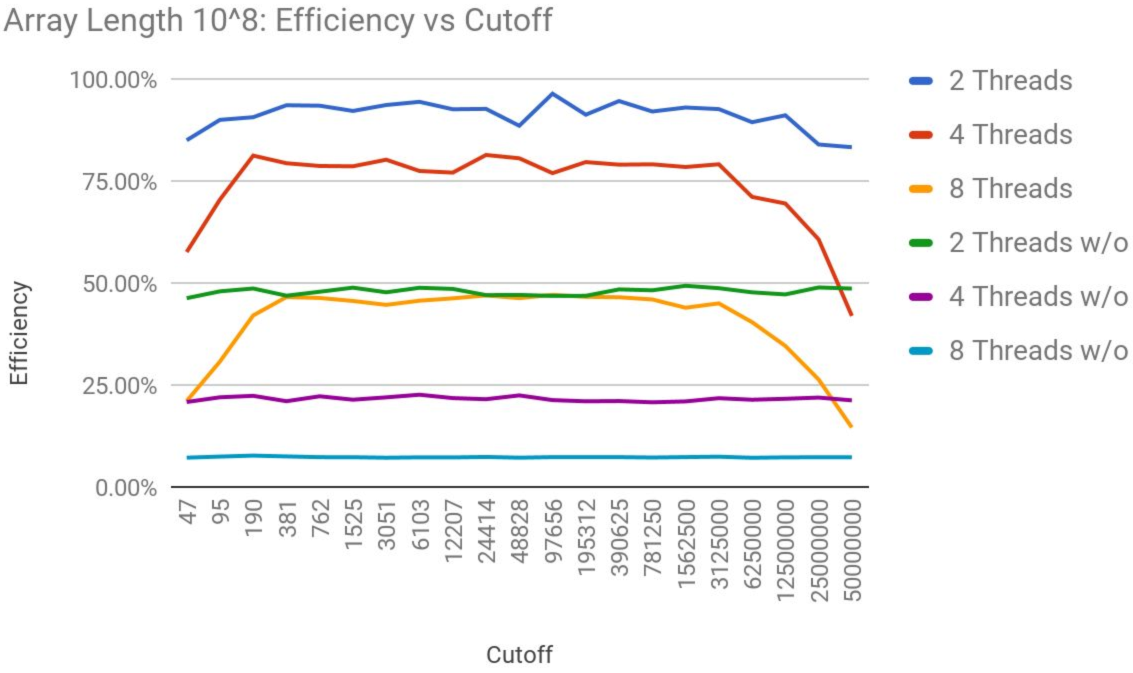

Array Length 10^4
40
30
20
10
0
Time (ms)
39
78
156
312
625
1250
2500
5000
Cutoff
1 Thread
2 Threads
4 Threads
8 Threads
Dual-Pivot


Array Length 10^5
100
75
50
25
0
Time (ms)
48
97
195
390
781
1562
3125
6250
12500
25000
50000
Cutoff
1 Thread
2 Threads
4 Threads
8 Threads
Dual-Pivot


Array Length 10^6
400
300
200
100
0
Time (ms)
61
122
244
488
976
1953
3906
7812
15625
31250
62500
125000
250000
Cutoff
1 Thread
2 Threads
4 Threads
8 Threads
Dual-Pivot


Array Length 10^7
2000
1500
1000
500
0
Time (ms)
38
76
152
305
610
1220
2441
4882
9765
19531
39062
78125
156250
312500
625000
1250000
2500000
Cutoff
1 Thread
2 Threads
4 Threads
8 Threads
Dual-Pivot


Array Length 10^8
Time (ms)
20000
15000
10000
5000
0
47
95
190
381
762
1525
3051
6103
12207
24414
48828
97656
195312
390625
781250
1562500
3125000
6250000
12500000
25000000
50000000
Cutoff
1 Thread
2 Threads
4 Threads
8 Threads
Dual-Pivot


Array Length 10^8: Efficiency vs Cutoff
Efficiency
100.00%
75.00%
50.00%
25.00%
0.00%
47
95
190
381
762
1525
3051
6103
12207
24414
48828
97656
195312
390625
781250
1562500
3125000
6250000
12500000
25000000
50000000
Cutoff
2 Threads
4 Threads
8 Threads
2 Threads w/o
4 Threads w/o
8 Threads w/o

Conclusions

The best configuration changes according to the number of items to be ordered:

- Array length 10^4: The best one is the Dual-Pivot Algorithm. This happen because there are too few elements and the overhead due to thread penalizes our algorithms.
- Array length 10^5: The best one is our quicksort implementation with one thread and cutoffs 12500 (23.86 ms) and 25000 (25.6). Similar results are obtained also for 2 threads, 4 threads and sequential Dual-Pivot quicksort algorithm. The version with 8 threads requires significantly more time.
- Array length 10^6: All the algorithms have similar performances, except the one with 8 thread. The sequential quicksort is a little faster than the parallel versions. The 2 Threads and 4 Threads versions are substantially equivalent and seem to be independent from the sequential cutoff. The version with 8 threads has no intersection with the 1 Thread/2 Thread and 4 thread lines.
- Array Length 10^7: The sequential algorithms (1 Thread and Dual-Pivot) require more time than the parallel ones. In particular the version with 4 Threads results faster than the one with 2 and 8 Threads for each cutoff. The line of the version with 8 threads is sandwiched between the line of 2 and 4 threads.
- Array length 10^8: Here the fastest version is the one with 8 threads and presents a parabolic trend, which means that cutoff too high or too low are not suitable. The minimum value of 3.8 seconds is for cutoff 97656 (i.e. for array-length/1024). The version with 4 threads has a similar behaviour but a minimum value of 4.4 seconds for a cutoff of 48828 (i.e. for array-length/2048).

From these particular considerations we can conclude that under a certain number of array elements (less than 1 Milion) it is always better to use the standard library Dual-Pivot Quicksort algorithm because we obtain good performances using only a single thread and it is already implemented.

The fact that the sequential versions performs better is probably due to the overhead of the thread management.

When we are considering arrays of 10 Millions and 100 Millions size we can note that the parallel versions are considerably faster.

In addition we compared the speedup (T_1 / T_N, where T_1 is the time required with one thread, and T_N the time required with N threads) and efficiency (T_1/T_N/N) of the parallel algorithms with work-stealing and without (w/o) it. The collected data confirms the fact that work-stealing leads to better hardware exploitation, especially when the array-size increases, as demonstrated by the reported graph. For instance the algorithm with work-stealing and 2 threads has a measured efficiency very close to 100% whereas the version with 2 threads without work-stealing has an efficiency of circa 50%.

Appendix: Work-stealing results

Array length: 10^4

Time (ms)

Cutoff	Dual-Pivot	1 Thread	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
39	3.980398	14.1408062	10.5388376	10.657622	31.1248332	14.4695894	18.5830594	25.028056
78	3.980398	11.5112188	7.9749418	6.4703408	20.2445164	11.3527806	13.8310288	25.185414
156	3.980398	9.4019834	6.4142794	4.9844316	16.310518	9.3591884	11.5938914	25.9624968
312	3.980398	8.4974256	5.619544	4.484722	10.5844322	8.0011642	10.2160844	19.9604768
625	3.980398	9.2148402	5.331387	4.4225866	8.675204	7.3319438	9.979176	12.7955584
1250	3.980398	6.9585068	5.2229216	4.8739322	7.922828	6.4461738	8.2704694	13.6739416
2500	3.980398	6.4493906	5.557268	4.7596044	10.5559264	6.3400758	8.3307658	15.9386318
5000	3.980398	5.9742238	5.3477932	6.0137074	9.3472774	5.9232374	7.984088	9.2664296

Table 1: Array length 10000, execution time

Speedup wrt 1 Thread

Cutoff	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
39	134.18%	132.68%	33.86%	97.73%	76.10%	56.50%
78	144.34%	177.91%	39.39%	101.40%	83.23%	45.71%
156	146.58%	188.63%	39.33%	100.46%	81.09%	36.21%
312	151.21%	189.47%	53.09%	106.20%	83.18%	42.57%
625	172.84%	208.36%	61.46%	125.68%	92.34%	72.02%
1250	133.23%	142.77%	65.92%	107.95%	84.14%	50.89%
2500	116.05%	135.50%	52.65%	101.72%	77.42%	40.46%
5000	111.71%	99.34%	57.21%	100.86%	74.83%	64.47%

Table 2: Array length 10000, Speedup wrt 1 Thread

Efficiency = Speedup / # Servers

Cutoff	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
39	67.09%	33.17%	4.23%	48.86%	19.02%	7.06%
78	72.17%	44.48%	4.92%	50.70%	20.81%	5.71%
156	73.29%	47.16%	4.92%	50.23%	20.27%	4.53%
312	75.61%	47.37%	6.64%	53.10%	20.79%	5.32%
625	86.42%	52.09%	7.68%	62.84%	23.09%	9.00%
1250	66.62%	35.69%	8.24%	53.97%	21.03%	6.36%
2500	58.03%	33.88%	6.58%	50.86%	19.35%	5.06%
5000	55.86%	24.84%	7.15%	50.43%	18.71%	8.06%

Table 3: Array length 10000, Efficiency

Stolen Tasklets (%)

Cutoff	Total Tasklets	1 Thread	2 Threads	4 Threads	8 Threads
39	1139.8	0	0.95%	3.89%	10.63%
78	581	0	1.71%	7.41%	17.71%
156	295	0	3.12%	11.31%	29.78%
312	146.2	0	5.47%	19.53%	37.92%
625	72.2	0	9.25%	30.25%	49.31%
1250	32.6	0	17.55%	46.50%	66.26%
2500	16.6	0	26.75%	60.96%	82.65%
5000	7.8	0	43.59%	88.21%	96.92%

Table 4: Array length 10000, Stolen Tasklets

Array length: 10^5

Time (ms)

Cutoff	Dual-Pivot	1 Thread	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
48	30.700728	47.1259206	42.3157598	43.2456812	93.8040318	41.1809652	53.2157714	133.4798746
97	30.700728	36.003556	31.7547806	34.6370094	65.5037226	33.5071186	43.2115058	76.4420542
195	30.700728	38.4394146	28.7023782	31.9080628	65.3950738	30.1337354	36.451444	69.8077038
390	30.700728	31.604734	27.3188146	31.085333	64.8888418	30.0742714	35.7500002	62.0781232
781	30.700728	29.0775846	27.2468296	31.4171026	51.3142692	30.1496128	33.6943874	62.5793328
1562	30.700728	27.860856	27.5699016	31.6866258	55.4266486	27.362921	30.845213	53.8233338
3125	30.700728	25.7346238	28.1409782	30.4253168	52.1259684	25.6108938	29.5942894	50.8168524
6250	30.700728	29.7108758	28.9610386	31.9516038	46.467785	24.1756846	28.1298106	51.1214218
12500	30.700728	23.862862	31.351783	33.853966	54.1386446	23.4501148	26.6157368	43.4135588
25000	30.700728	25.5411332	31.5522892	35.3556988	55.8138552	24.1849844	26.323782	44.136982
50000	30.700728	36.7176036	37.7624086	37.4579274	51.3056658	36.9977892	38.0910874	60.5375534

Table 5: Array length 100000, execution time

Speedup wrt 1 Thread

Cutoff	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
48	111.37%	108.97%	50.24%	114.44%	88.56%	35.31%
97	113.38%	103.95%	54.96%	107.45%	83.32%	47.10%
195	133.92%	120.47%	58.78%	127.56%	105.45%	55.06%
390	115.69%	101.67%	48.71%	105.09%	88.40%	50.91%
781	106.72%	92.55%	56.67%	96.44%	86.30%	46.47%
1562	101.06%	87.93%	50.27%	101.82%	90.32%	51.76%
3125	91.45%	84.58%	49.37%	100.48%	86.96%	50.64%
6250	102.59%	92.99%	63.94%	122.90%	105.62%	58.12%
12500	76.11%	70.49%	44.08%	101.76%	89.66%	54.97%
25000	80.95%	72.24%	45.76%	105.61%	97.03%	57.87%
50000	97.23%	98.02%	71.57%	99.24%	96.39%	60.65%

Table 6: Array length 100000, Speedup wrt 1 Thread

Efficiency = Speedup / # Servers

Cutoff	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
48	55.68%	27.24%	6.28%	57.22%	22.14%	4.41%
97	56.69%	25.99%	6.87%	53.73%	20.83%	5.89%
195	66.96%	30.12%	7.35%	63.78%	26.36%	6.88%
390	57.84%	25.42%	6.09%	52.54%	22.10%	6.36%
781	53.36%	23.14%	7.08%	48.22%	21.57%	5.81%
1562	50.53%	21.98%	6.28%	50.91%	22.58%	6.47%
3125	45.72%	21.15%	6.17%	50.24%	21.74%	6.33%
6250	51.29%	23.25%	7.99%	61.45%	26.41%	7.26%
12500	38.06%	17.62%	5.51%	50.88%	22.41%	6.87%
25000	40.47%	18.06%	5.72%	52.80%	24.26%	7.23%
50000	48.62%	24.51%	8.95%	49.62%	24.10%	7.58%

Table 7: Array length 100000, Efficiency

Stolen Tasklets (%)

Cutoff	Total Tasklets	1 Thread	2 Threads	4 Threads	8 Threads
48	9303.4	0.00%	0.114%	0.70%	2.17%
97	4599	0.00%	0.232%	1.28%	4.38%
195	2332.2	0.00%	0.405%	2.20%	7.82%
390	1188.2	0.00%	0.757%	4.41%	13.91%
781	590.6	0.00%	1.443%	6.92%	23.75%
1562	289.4	0.00%	2.488%	12.48%	35.22%
3125	141.8	0.00%	4.570%	21.64%	45.33%
6250	74.6	0.00%	8.043%	28.79%	52.65%
12500	33.8	0.00%	15.266%	46.15%	65.44%
25000	16.6	0.00%	24.096%	62.41%	83.37%
50000	7.4	0.00%	43.784%	87.03%	95.14%

Table 8: Array length 100000, Stolen Tasklets

Array length: 10^6

Time (ms)

Cutoff	Dual-Pivot	1 Thread	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
61	167.6305676	214.5885426	181.2520478	208.3414308	251.3428702	181.7815552	223.0540388	487.070484
122	167.6305676	177.5395876	168.7448876	186.6597996	241.680968	174.5199396	216.0422656	442.7910074
244	167.6305676	163.4419398	161.050726	161.0434154	246.7625866	183.2118904	210.7849498	424.7453248
488	167.6305676	171.066122	152.1352002	165.1198446	238.4835306	219.5508118	204.5093446	411.1357328
976	167.6305676	167.986448	152.9417574	167.3604928	240.3303626	159.1884776	185.49156	384.849397
1953	167.6305676	192.9978916	170.6242492	179.5988884	250.5845556	163.7378422	198.3042952	372.7956838
3906	167.6305676	155.8769052	172.9234222	175.3962896	249.4038918	162.8618446	183.66903	385.2799844
7812	167.6305676	197.1580542	173.58582	181.2981026	204.4244876	161.7578934	181.1280914	308.2992154
15625	167.6305676	162.3177252	183.3786608	199.0600152	240.9064298	161.2921056	180.0438848	351.8311592
31250	167.6305676	157.0774984	181.6523896	182.631587	273.2433042	171.9129874	181.3319848	344.904345
62500	167.6305676	164.578349	183.2515784	194.4237274	276.1140344	170.0212416	189.7056298	342.7407462
125000	167.6305676	202.5033448	199.4885358	208.071759	273.9407124	189.0809196	209.5073914	363.8167184
250000	167.6305676	240.898021	188.1423248	159.2796884	292.3508578	221.3184294	239.3225464	424.3358754
500000	167.6305676	215.1909612	176.3458542	189.146675	312.2646128	204.6679676	258.821246	447.026401

Table 9: Array length 1000000, execution time

Speedup wrt 1 Thread

Cutoff	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
61	118.39%	103.00%	85.38%	118.05%	96.20%	44.06%
122	105.21%	95.11%	73.46%	101.73%	82.18%	40.10%
244	101.48%	101.49%	66.23%	89.21%	77.54%	38.48%
488	112.44%	103.60%	71.73%	77.92%	83.65%	41.61%
976	109.84%	100.37%	69.90%	105.53%	90.56%	43.65%
1953	113.11%	107.46%	77.02%	117.87%	97.32%	51.77%
3906	90.14%	88.87%	62.50%	95.71%	84.87%	40.46%
7812	113.58%	108.75%	96.45%	121.88%	108.85%	63.95%
15625	88.52%	81.54%	67.38%	100.64%	90.15%	46.14%
31250	86.47%	86.01%	57.49%	91.37%	86.62%	45.54%
62500	89.81%	84.65%	59.61%	96.80%	86.75%	48.02%
125000	101.51%	97.32%	73.92%	107.10%	96.66%	55.66%
250000	128.04%	151.24%	82.40%	108.85%	100.66%	56.77%
500000	122.03%	113.77%	68.91%	105.14%	83.14%	48.14%

Table 10: Array length 1000000, Speedup wrt 1 Thread

Efficiency = Speedup / # Servers

Cutoff	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
61	59.20%	25.75%	10.67%	59.02%	24.05%	5.51%
122	52.61%	23.78%	9.18%	50.87%	20.54%	5.01%
244	50.74%	25.37%	8.28%	44.60%	19.38%	4.81%
488	56.22%	25.90%	8.97%	38.96%	20.91%	5.20%
976	54.92%	25.09%	8.74%	52.76%	22.64%	5.46%
1953	56.56%	26.87%	9.63%	58.94%	24.33%	6.47%
3906	45.07%	22.22%	7.81%	47.86%	21.22%	5.06%
7812	56.79%	27.19%	12.06%	60.94%	27.21%	7.99%
15625	44.26%	20.39%	8.42%	50.32%	22.54%	5.77%
31250	43.24%	21.50%	7.19%	45.69%	21.66%	5.69%
62500	44.91%	21.16%	7.45%	48.40%	21.69%	6.00%
125000	50.76%	24.33%	9.24%	53.55%	24.16%	6.96%
250000	64.02%	37.81%	10.30%	54.42%	25.16%	7.10%
500000	61.01%	28.44%		52.57%	20.79%	6.02%

Table 11: Array length 1000000, Efficiency

Stolen Tasklets (%)

Cutoff	Total Tasklets	1 Thread	2 Threads	4 Threads	8 Threads
61	73358.2	0	0.02%	0.11%	0.44%
122	37059	0	0.04%	0.20%	0.85%
244	18677	0	0.07%	0.36%	1.33%
488	9339	0	0.12%	0.60%	2.25%
976	4613	0	0.25%	1.15%	3.73%
1953	2309	0	0.49%	2.01%	6.33%
3906	1163.8	0	0.86%	3.69%	10.11%
7812	583.4	0	1.60%	6.17%	16.93%
15625	291.8	0	2.78%	9.42%	23.50%
31250	144.6	0	5.34%	16.49%	34.94%
62500	77	0	8.99%	24.88%	45.56%
125000	35	0	17.49%	38.63%	61.49%
250000	16.2	0	30.86%	55.06%	82.22%
500000	7	0	45.14%	83.43%	93.71%

Table 12: Array length 1000000, Stolen Tasklets

Array length: 10^7

Time (ms)

Cutoff	Dual-Pivot	1 Thread	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
38	1287.296558	1653.578568	1031.221432	976.2003248	1275.580318	1732.338412	1985.769649	2999.020474
76	1287.296558	1453.515399	887.1063712	696.9910634	881.4497978	1480.802291	1575.726168	3004.188167
152	1287.296558	1423.091911	833.5751844	608.7112382	728.9272064	1409.673625	1601.559374	2606.97764
305	1287.296558	1335.569331	800.8414066	596.7028794	811.5389522	1365.981822	1548.780944	2429.143859
610	1287.296558	1311.544587	806.8837398	600.6980276	823.6325672	1348.533185	1676.784164	2564.560306
1220	1287.296558	1320.983329	781.4904964	594.557301	693.5271544	1379.013677	1478.643534	2477.70607
2441	1287.296558	1301.455236	785.6135632	589.1772922	710.3353736	1370.753008	1555.022269	2359.388056
4882	1287.296558	1279.896854	782.4205852	591.191473	757.5926132	1317.801245	1482.504548	2402.720996
9765	1287.296558	1337.734612	773.497303	565.190388	639.6063332	1352.460753	1531.911669	2345.916822
19531	1287.296558	1291.812075	784.747165	572.847838	751.4583846	1344.513044	1441.089128	2383.822971
39062	1287.296558	1297.185879	777.5857092	558.1874536	640.5272664	1357.091316	1477.337439	2383.714736
78125	1287.296558	1341.283441	758.8028168	549.6010068	638.4626186	1404.778997	1519.541491	2331.736615
156250	1287.296558	1365.713643	780.2762414	559.3057354	626.2815254	1371.51733	1574.471488	2340.729488
312500	1287.296558	1428.834284	786.6345546	578.0985776	673.9862552	1384.981876	1567.474449	2373.129261
625000	1287.296558	1330.348758	802.8122946	610.4559956	739.0069112	1402.132939	1568.277751	2396.572459
1250000	1287.296558	1377.376411	818.5893214	609.7572888	753.0542308	1414.385601	1626.776936	2396.000754
2500000	1287.296558	1343.313912	788.9052484	592.2859042	680.4154798	1379.970241	1573.279369	2364.756664
5000000	1287.296558	1332.816885	818.2184888	761.1828952	1171.713912	1359.235417	1551.236293	2317.372398

Table 13: Array length 10000000, execution time

Speedup wrt 1 Thread

Cutoff	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
38	160.35%	169.39%	129.63%	95.45%	83.27%	55.14%
76	163.85%	208.54%	164.90%	98.16%	92.24%	48.38%
152	170.72%	233.79%	195.23%	100.95%	88.86%	54.59%
305	166.77%	223.82%	164.57%	97.77%	86.23%	54.98%
610	162.54%	218.34%	159.24%	97.26%	78.22%	51.14%
1220	169.03%	222.18%	190.47%	95.79%	89.34%	53.31%
2441	165.66%	220.89%	183.22%	94.94%	83.69%	55.16%
4882	163.58%	216.49%	168.94%	97.12%	86.33%	53.27%
9765	172.95%	236.69%	209.15%	98.91%	87.32%	57.02%
19531	164.62%	225.51%	171.91%	96.08%	89.64%	54.19%
39062	166.82%	232.39%	202.52%	95.59%	87.81%	54.42%
78125	176.76%	244.05%	210.08%	95.48%	88.27%	57.52%
156250	175.03%	244.18%	218.07%	99.58%	86.74%	58.35%
312500	181.64%	247.16%	212.00%	103.17%	91.16%	60.21%
625000	165.71%	217.93%	180.02%	94.88%	84.83%	55.51%
1250000	168.26%	225.89%	182.91%	97.38%	84.67%	57.49%
2500000	170.28%	226.80%	197.43%	97.34%	85.38%	56.81%
5000000	162.89%	175.10%	113.75%	98.06%	85.92%	57.51%

Table 14: Array length 10000000, Speedup wrt 1 Thread

Efficiency = Speedup / # Servers

Cutoff	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
38	80.18%	42.35%	16.20%	47.73%	20.82%	6.89%
76	81.92%	52.14%	20.61%	49.08%	23.06%	6.05%
152	85.36%	58.45%	24.40%	50.48%	22.21%	6.82%
305	83.39%	55.96%	20.57%	48.89%	21.56%	6.87%
610	81.27%	54.58%	19.90%	48.63%	19.55%	6.39%
1220	84.52%	55.54%	23.81%	47.90%	22.33%	6.66%
2441	82.83%	55.22%	22.90%	47.47%	20.92%	6.90%
4882	81.79%	54.12%	21.12%	48.56%	21.58%	6.66%
9765	86.47%	59.17%	26.14%	49.46%	21.83%	7.13%
19531	82.31%	56.38%	21.49%	48.04%	22.41%	6.77%
39062	83.41%	58.10%	25.31%	47.79%	21.95%	6.80%
78125	88.38%	61.01%	26.26%	47.74%	22.07%	7.19%
156250	87.51%	61.05%	27.26%	49.79%	21.69%	7.29%
312500	90.82%	61.79%	26.50%	51.58%	22.79%	7.53%
625000	82.86%	54.48%	22.50%	47.44%	21.21%	6.94%
1250000	84.13%	56.47%	22.86%	48.69%	21.17%	7.19%
2500000	85.14%	56.70%	24.68%	48.67%	21.35%	7.10%
5000000	81.45%	43.77%	14.22%	49.03%	21.48%	7.19%

Table 15: Array length 10000000, Efficiency

Stolen Tasklets (%)

Cutoff			2 Threads	4 Threads	8 Threads
38	1162277	0.00%	0.001%	0.01%	0.05%
76	591542.2	0.00%	0.003%	0.02%	0.07%
152	298552.6	0.00%	0.005%	0.03%	0.17%
305	149528.2	0.00%	0.010%	0.07%	0.29%
610	75092.6	0.00%	0.018%	0.12%	0.62%
1220	37475	0.00%	0.034%	0.25%	1.03%
2441	18697.4	0.00%	0.066%	0.43%	1.77%
4882	9402.2	0.00%	0.125%	0.75%	2.82%
9765	4672.6	0.00%	0.222%	1.30%	4.45%
19531	2364.6	0.00%	0.391%	2.30%	6.89%
39062	1154.2	0.00%	0.710%	3.88%	12.17%
78125	583.4	0.00%	1.248%	6.48%	18.85%
156250	279	0.00%	2.280%	11.17%	27.14%
312500	143.8	0.00%	4.117%	17.02%	37.55%
625000	69.4	0.00%	6.052%	26.40%	49.51%
1250000	33	0.00%	10.909%	38.06%	66.55%
2500000	13.4	0.00%	22.388%	60.30%	94.03%
5000000	5.8	0.00%	41.379%	77.93%	98.62%

Table 16: Array length 10000000, Stolen Tasklets

Array length: 10^8

Time (ms)

Cutoff	Dual-Pivot	1 Thread	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
47	10530.51122	17318.54869	10192.33814	7520.300798	10286.73017	18717.5907	20794.53552	30163.50744
95	10530.51122	15924.6676	8850.879585	5653.673749	6476.673178	16610.19478	18117.35474	26733.99974
190	10530.51122	15715.24373	8671.840101	4838.616787	4674.42594	16155.91217	17593.09608	25566.71369
381	10530.51122	15248.51179	8150.602097	4805.395586	4095.261769	16266.63912	18140.6275	25457.86321
762	10530.51122	14855.06638	7949.658361	4721.395518	4009.042709	15526.21487	16712.69604	25444.04869
1525	10530.51122	14515.78444	7876.696803	4618.062111	3980.935094	14859.27879	16971.69556	24892.88695
3051	10530.51122	14370.4719	7676.569479	4479.947736	4025.560105	15061.19092	16358.89699	25064.96279
6103	10530.51122	14367.50568	7611.590331	4638.580334	3934.727712	14717.58166	15890.54089	24741.65789
12207	10530.51122	14274.87075	7711.305905	4632.786215	3860.480577	14705.99076	16387.36545	24654.74019
24414	10530.51122	14427.2794	7785.840536	4433.349568	3842.277188	15342.8376	16772.91391	24514.12152
48828	10530.51122	14241.72987	8044.890378	4420.66691	3847.057521	15132.45636	15866.5807	24818.16476
97656	10530.51122	14420.89535	7481.374146	4687.754572	3828.610497	15399.68957	16938.24632	24646.87007
195312	10530.51122	14416.35773	7901.352538	4526.870585	3867.649239	15387.1862	17168.21605	24639.23081
390625	10530.51122	14505.34482	7668.934	4591.322529	3899.894311	14980.45182	17222.86298	24796.20112
781250	10530.51122	14326.2423	7785.064617	4529.745943	3897.412115	14862.88282	17263.90745	24811.764
1562500	10530.51122	14438.38755	7763.629999	4603.632161	4108.859079	14643.94676	17217.08266	24664.89595
3125000	10530.51122	14553.2061	7858.167678	4600.981799	4043.919455	14938.16598	16725.63178	24534.03116
6250000	10530.51122	14098.57204	7886.161474	4958.299004	4364.082078	14779.57961	16490.22625	24646.3558
12500000	10530.51122	14366.17694	7886.976765	5170.324198	5195.6254	15224.20968	16621.53155	24748.67242
25000000	10530.51122	14318.46228	8530.522341	5899.517524	6794.180621	14643.53528	16344.14712	24587.91445
50000000	10530.51122	14392.55452	8643.304677	8584.932639	12350.98619	14802.79182	16954.03714	24709.49307

Table 17: Array length 100000000, execution time

Speedup wrt 1 Thread

Cutoff	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
47	169.92%	230.29%	168.36%	92.53%	83.28%	57.42%
95	179.92%	281.67%	245.88%	95.87%	87.90%	59.57%
190	181.22%	324.79%	336.20%	97.27%	89.33%	61.47%
381	187.08%	317.32%	372.35%	93.74%	84.06%	59.90%
762	186.86%	314.63%	370.54%	95.68%	88.88%	58.38%
1525	184.29%	314.33%	364.63%	97.69%	85.53%	58.31%
3051	187.20%	320.77%	356.98%	95.41%	87.84%	57.33%
6103	188.76%	309.74%	365.15%	97.62%	90.42%	58.07%
12207	185.12%	308.13%	369.77%	97.07%	87.11%	57.90%
24414	185.30%	325.43%	375.49%	94.03%	86.02%	58.85%
48828	177.03%	322.16%	370.20%	94.11%	89.76%	57.38%
97656	192.76%	307.63%	376.66%	93.64%	85.14%	58.51%
195312	182.45%	318.46%	372.74%	93.69%	83.97%	58.51%
390625	189.14%	315.93%	371.94%	96.83%	84.22%	58.50%
781250	184.02%	316.27%	367.58%	96.39%	82.98%	57.74%
1562500	185.97%	313.63%	351.40%	98.60%	83.86%	58.54%
3125000	185.20%	316.31%	359.88%	97.42%	87.01%	59.32%
6250000	178.78%	284.34%	323.06%	95.39%	85.50%	57.20%
12500000	182.15%	277.86%	276.51%	94.36%	86.43%	58.05%
25000000	167.85%	242.71%	210.75%	97.78%	87.61%	58.23%
50000000	166.52%	167.65%	116.53%	97.23%	84.89%	58.25%

Table 18: Array length 100000000, Speedup wrt 1 Thread

Efficiency = Speedup / # Servers

Cutoff	2 Threads	4 Threads	8 Threads	2 Threads w/o W-S	4 Threads w/o W-S	8 Threads w/o W-S
47	84.96%	57.57%	21.04%	46.26%	20.82%	7.18%
95	89.96%	70.42%	30.73%	47.94%	21.97%	7.45%
190	90.61%	81.20%	42.02%	48.64%	22.33%	7.68%
381	93.54%	79.33%	46.54%	46.87%	21.01%	7.49%
762	93.43%	78.66%	46.32%	47.84%	22.22%	7.30%
1525	92.14%	78.58%	45.58%	48.84%	21.38%	7.29%
3051	93.60%	80.19%	44.62%	47.71%	21.96%	7.17%
6103	94.38%	77.43%	45.64%	48.81%	22.60%	7.26%
12207	92.56%	77.03%	46.22%	48.53%	21.78%	7.24%
24414	92.65%	81.36%	46.94%	47.02%	21.50%	7.36%
48828	88.51%	80.54%	46.27%	47.06%	22.44%	7.17%
97656	96.38%	76.91%	47.08%	46.82%	21.28%	7.31%
195312	91.23%	79.62%	46.59%	46.85%	20.99%	7.31%
390625	94.57%	78.98%	46.49%	48.41%	21.06%	7.31%
781250	92.01%	79.07%	45.95%	48.19%	20.75%	7.22%
1562500	92.99%	78.41%	43.92%	49.30%	20.97%	7.32%
3125000	92.60%	79.08%	44.98%	48.71%	21.75%	7.41%
6250000	89.39%	71.09%	40.38%	47.70%	21.37%	7.15%
12500000	91.08%	69.46%	34.56%	47.18%	21.61%	7.26%
25000000	83.92%	60.68%	26.34%	48.89%	21.90%	7.28%
50000000	83.26%	41.91%	14.57%	48.61%	21.22%	7.28%

Table 19: Array length 100000000, Efficiency

Stolen Tasklets (%)

Cutoff	Total Tasklets	1 Thread	2 Threads	4 Threads	8 Threads
47	9467293.4	0.00%	0.00%	0.00%	0.01%
95	4751687.8	0.00%	0.00%	0.00%	0.03%
190	2392325	0.00%	0.00%	0.01%	0.06%
381	1197559	0.00%	0.00%	0.01%	0.10%
762	600229.4	0.00%	0.00%	0.02%	0.21%
1525	299860.2	0.00%	0.01%	0.04%	0.38%
3051	149830.2	0.00%	0.01%	0.08%	0.67%
6103	74965	0.00%	0.02%	0.15%	1.14%
12207	37457.4	0.00%	0.03%	0.23%	1.84%
24414	18751	0.00%	0.06%	0.45%	3.14%
48828	9348.6	0.00%	0.11%	0.81%	5.10%
97656	4683.4	0.00%	0.20%	1.55%	7.73%
195312	2314.2	0.00%	0.39%	2.71%	11.04%
390625	1168.6	0.00%	0.73%	4.56%	16.78%
781250	575.4	0.00%	1.28%	7.97%	24.87%
1562500	283.8	0.00%	2.23%	13.09%	36.66%
3125000	143	0.00%	4.03%	21.03%	44.31%
6250000	68.2	0.00%	7.80%	30.21%	58.65%
12500000	38.2	0.00%	12.57%	39.16%	69.95%
25000000	13	0.00%	29.54%	55.69%	96.92%
50000000	6.6	0.00%	51.52%	80.61%	96.36%

Table 20: Array length 100000000, Stolen Tasklets