Programming parallel computers: Exercise 1

Suhas Thejaswi Muniyappa (465959)

April 19, 2015

1 MF1

The efficiency of the program depends mainly in finding the median. The best performance was observed by using std::nth_element. Runtime of quick-select algorithm to find the median is almost same as nth_element but sorting (std::sort) resulted in worst performance of program. Table-1 is the statistics of runtime for different algorithms to find the median filter.

2 MF2

Table-2 is the statistics of runtime of the program with parallel execution using omp and a comparision of execution times with different number of threads. Figure-1 shows the plot of runtime of program for various image size and window size vs runtime. x-axis represents the combination of image and window size, y-axis represents execution time. The runtime of the program increases with increase in the image size(input size) and also increases with the increase in the window size. After parallelising using openmp, runtime decreases with increase in number of threads. With the increase in the number of threads by factor of 2, program runtime is almost reduced by a factor of 2 for large input and window size. For small Input and window size there is no significant change in execution time with increaase in number of threads. Figure-2, Figure-3, Figure-4, Figure-5, Figure-6 are the plots of window size vs runtime with various input size. From the plots it is observed that runtime of program increases exponentially with the increase in window size.

	ny	nx	window size	std::sort	quick select	std::nth_element
mf	100	100	1	0.002	0.002	0.003
mf	100	100	2	0.009	0.006	0.007
mf	100	100	5	0.040	0.018	0.020
mf	100	100	10	0.115	0.045	0.035
mf	200	200	1	0.004	0.009	0.004
mf	200	200	2	0.016	0.011	0.012
mf	200	200	5	0.111	0.047	0.045
mf	200	200	10	0.487	0.155	0.142
mf	500	500	1	0.025	0.025	0.026
mf	500	500	2	0.101	0.079	0.074
mf	500	500	5	0.701	0.297	0.277
mf	500	500	10	3.163	0.985	0.897
mf	1000	1000	1	0.098	0.100	0.107
mf	1000	1000	2	0.401	0.282	0.294
mf	1000	1000	5	2.827	1.169	1.111
mf	1000	1000	10	12.807	3.963	3.598
mf	2000	2000	1	0.379	0.362	0.428
mf	2000	2000	2	1.631	1.109	1.199
mf	2000	2000	5	11.369	4.668	4.463
mf	2000	2000	10	51.455	15.914	14.503

Table 1: MF1 runtime benchmark

	ny	nx	window size	1-thread	2-threads	4-threads	8-threads	default
mf	100	100	1	0.003	0.002	0.001	0.001	0.001
mf	100	100	2	0.007	0.004	0.003	0.009	0.002
mf	100	100	5	0.025	0.014	0.008	0.010	0.005
mf	100	100	10	0.039	0.025	0.020	0.016	0.007
mf	200	200	1	0.005	0.002	0.002	0.003	0.004
mf	200	200	2	0.013	0.013	0.005	0.010	0.003
mf	200	200	5	0.047	0.024	0.021	0.021	0.011
mf	200	200	10	0.156	0.084	0.051	0.031	0.029
mf	500	500	1	0.038	0.015	0.013	0.008	0.009
mf	500	500	2	0.080	0.041	0.027	0.017	0.018
mf	500	500	5	0.304	0.148	0.078	0.056	0.058
mf	500	500	10	0.937	0.471	0.266	0.170	0.174
mf	1000	1000	1	0.108	0.055	0.030	0.027	0.025
mf	1000	1000	2	0.303	0.154	0.094	0.062	0.071
mf	1000	1000	5	1.169	0.592	0.344	0.229	0.222
mf	1000	1000	10	3.752	1.890	1.029	0.671	0.670
mf	2000	2000	1	0.429	0.218	0.150	0.099	0.084
mf	2000	2000	2	1.209	0.627	0.358	0.224	0.226
mf	2000	2000	5	4.680	2.377	1.267	0.863	0.848
mf	2000	2000	10	15.011	7.670	3.998	2.713	2.759

Table 2: MF2 runtime benchmark

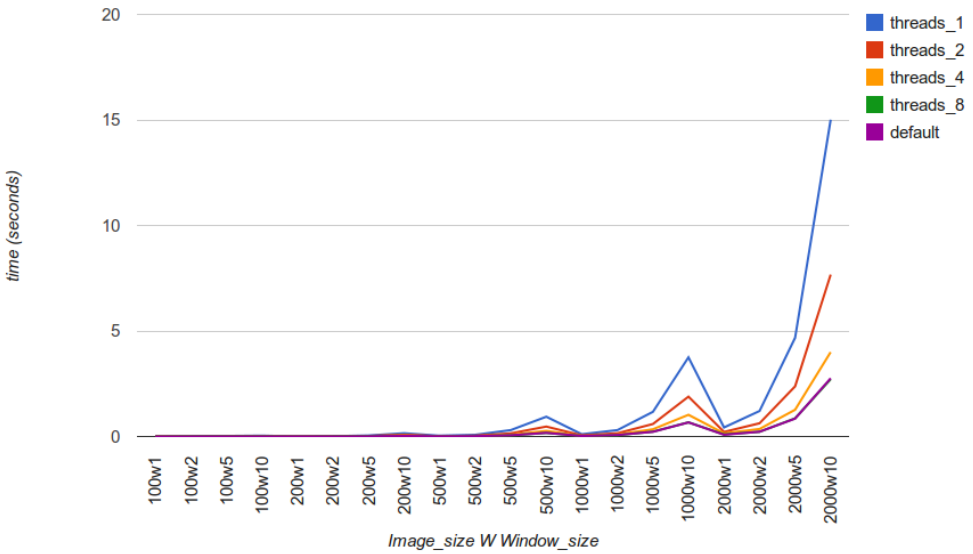

Figure 1: Plot of image vs time

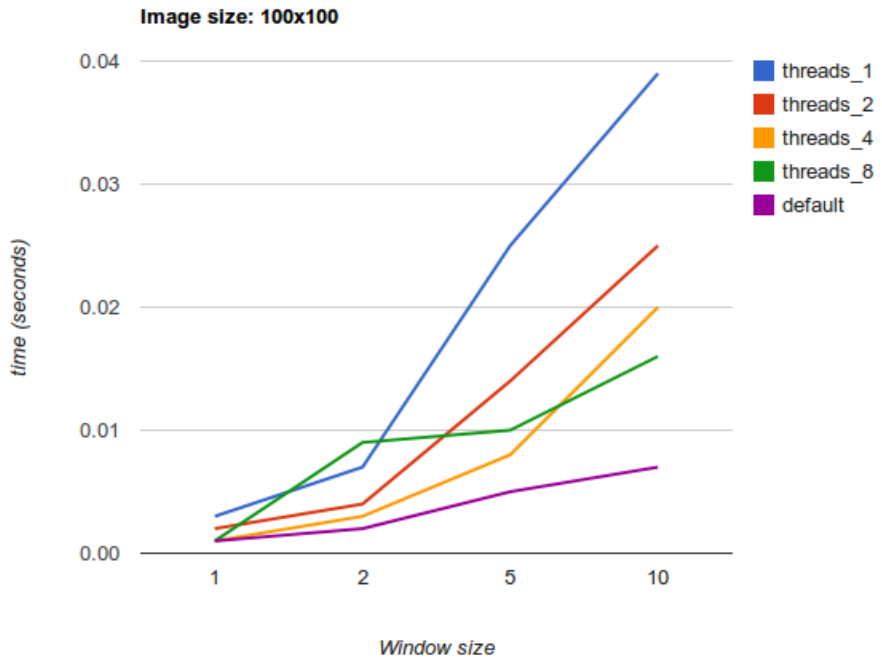

Figure 2: Plot of window size vs run-time for image size 100x100

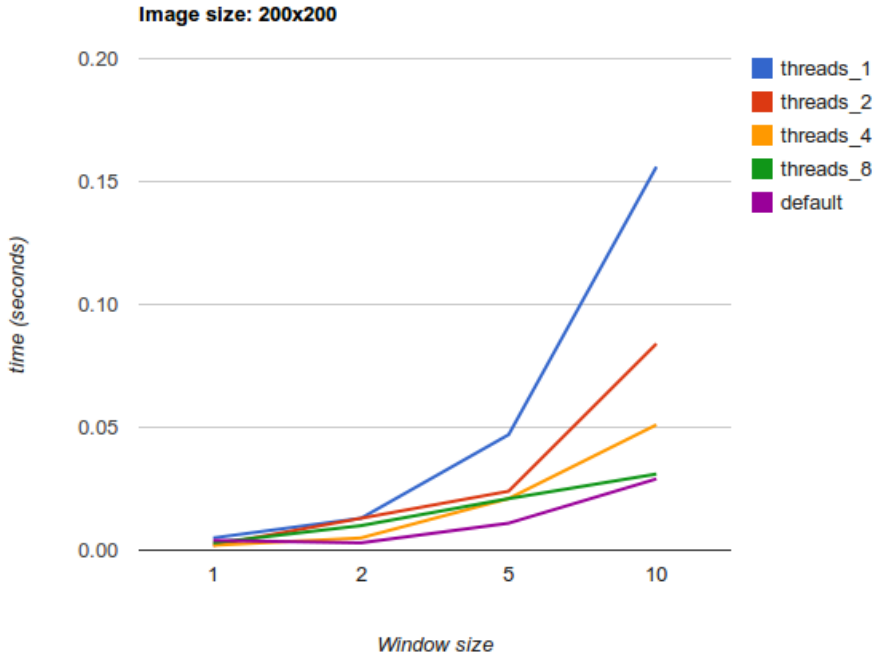

Figure 3: Plot of window size vs run-time for image size 200x200

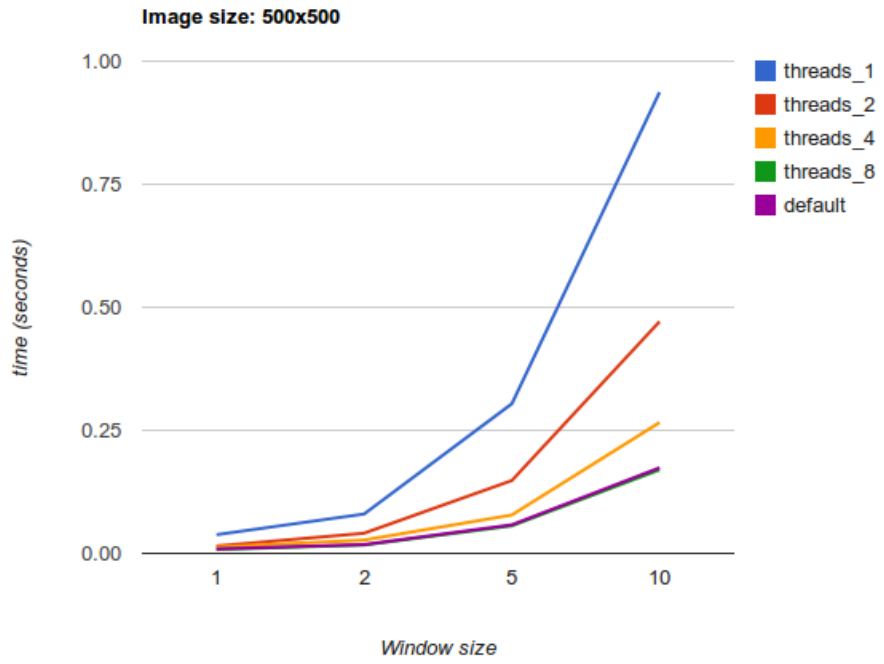

Figure 4: Plot of window size vs run-time for image size 500x500

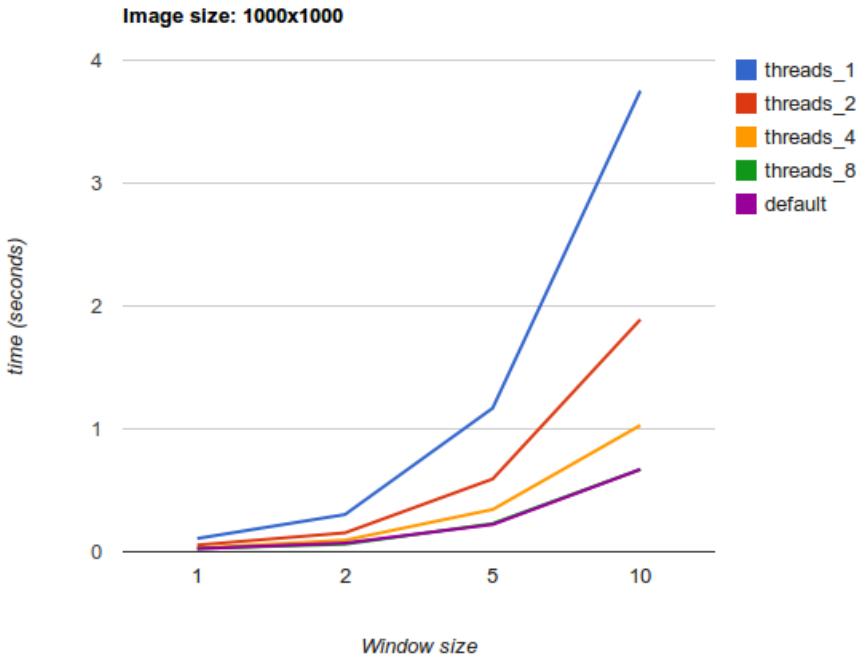

Figure 5: Plot of window size vs run-time for image size 1000x1000

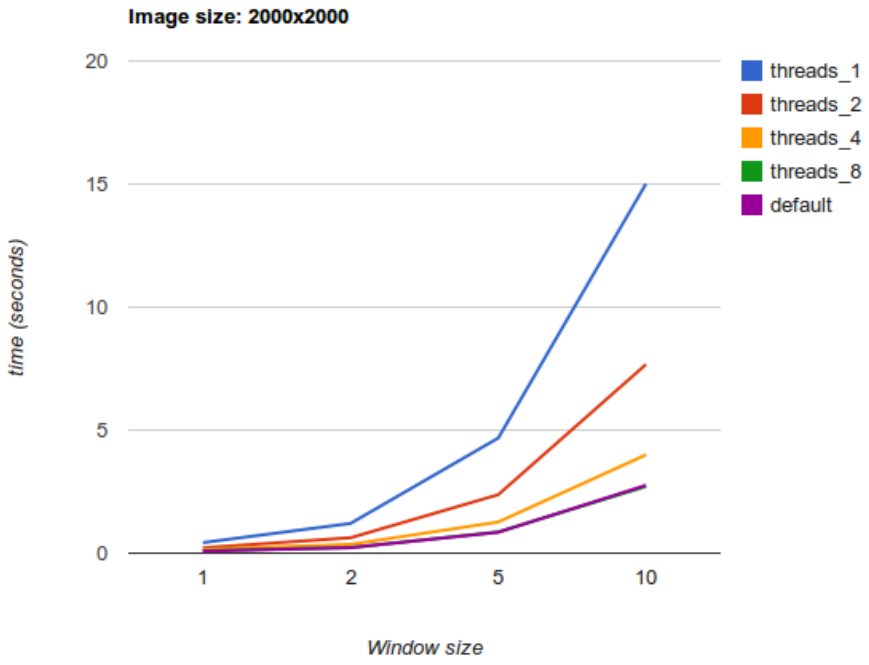

Figure 6: Plot of window size vs run-time for image size 2000x2000