Vipul Sharma
B17069
25-10-2020

CS592
Reactive Design Patterns
Assignment A2

Aim-

a. To explore and get familiar with Aggregator Message Flow Pattern.
b. Write a program that computes all primes in the range 1..MAX, where MAX is a large positive integer. Divide the range into N subranges and use a fixed-sized pool of ephemeral threads to obtain the sum of prime numbers in each subrange.

Observations-

Here n = nThreads = No of threads in Fixed Thread Pool

For MAX = 100000

Time Taken Table

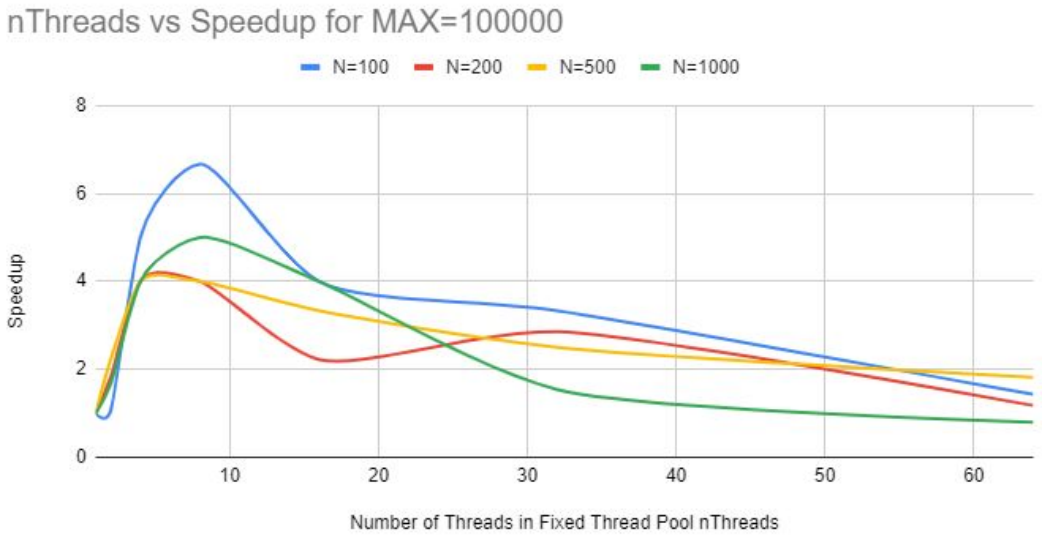
Time taken for Single Threaded Version = 20 ms

Number of Subranges N	**nThreads n=2**	**nThreads n=4**	**nThreads n=8**	**nThreads n=16**	**nThreads n=32**	**nThreads n=64**
100	19	4	3	12	6	14
200	11	5	5	9	7	17
500	9	5	5	6	8	11
1000	12	5	4	5	13	25

Speedup Table

Speedup = T(serial)/T(parallel) = 20/T(n)

Number of Subranges N	**nThreads n=2**	**nThreads n=4**	**nThreads n=8**	**nThreads n=16**	**nThreads n=32**	**nThreads n=64**
100	1.05	5	6.67	4	3.33	1.42
200	1.81	4	4	2.22	2.85	1.17
500	2.22	4	4	3.33	2.5	1.81
1000	1.66	4	5	4	1.53	0.8



For MAX = 1000000

Time Taken Table

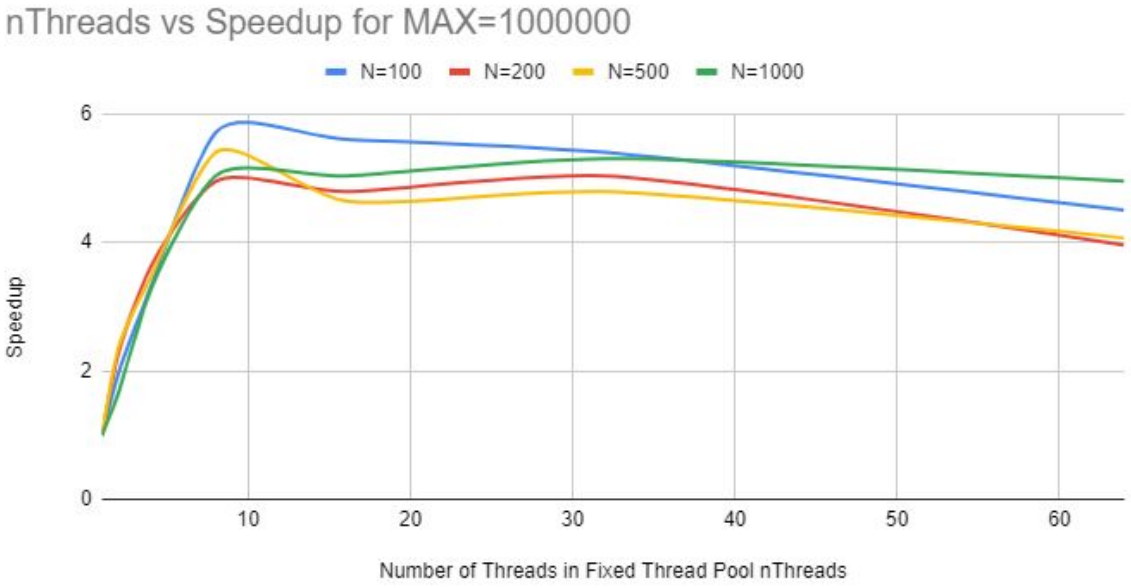
Time taken for Single Threaded Version = 297 ms

Number of Subranges N	nThreads n=2	nThreads n=4	nThreads n=8	nThreads n=16	nThreads n=32	nThreads n=64
100	152	90	52	53	55	66
200	131	82	60	62	59	75
500	126	86	55	64	62	73
1000	180	91	59	59	56	60

Speedup Table

Speedup = T(serial)/T(parallel) = 297/T(n)

Number of Subranges N	nThreads n=2	nThreads n=4	nThreads n=8	nThreads n=16	nThreads n=32	nThreads n=64
100	1.95	3.3	5.71	5.60	5.4	4.5
200	2.26	3.62	4.95	4.79	5.03	3.96
500	2.35	3.45	5.4	4.64	4.79	4.06
1000	1.65	3.26	5.03	5.03	5.30	4.95



Conclusions-

1. Fixed Thread Pool limits the maximum number of threads while the additional tasks wait in a queue.

2. Best speedup is achieved in Fixed Thread Pool with nThreads = 8 case and for some values of N, nThreads = 4 case. This is because I used a Quad core processor for the experiment . So only 4 threads actually execute concurrently. Also, there is a balance in work in the N=100 case. 100 tasks are submitted to the executor to be divided between the nThreads. Thus, work divided between the threads is optimal.

3. In our experiment, we are plotting a graph between speedup and number of threads in Fixed Thread Pool i.e. nThreads, and not the number of processors on which the tasks run. This is the reason the speedup decreases for large values of nThreads which we expect to eventually become constant in Amdahl's law, because the number of processors in our system is fixed, i.e. the Java compiler just virtualizes the running of multiple threads at the same time, but actually only 4 instructions can execute together. If we had multiple processors in our system the speedup would become constant after some value of nThreads. We can actually simulate Amdahl's Law by using Heterogeneous Computing with a Distributed Memory Architecture and a Message Passing Interface (MPI) where the tasks are divided between multiple computers on a network. Then, we would get a graph similar to the theoretical graph of Amdahl's Law.