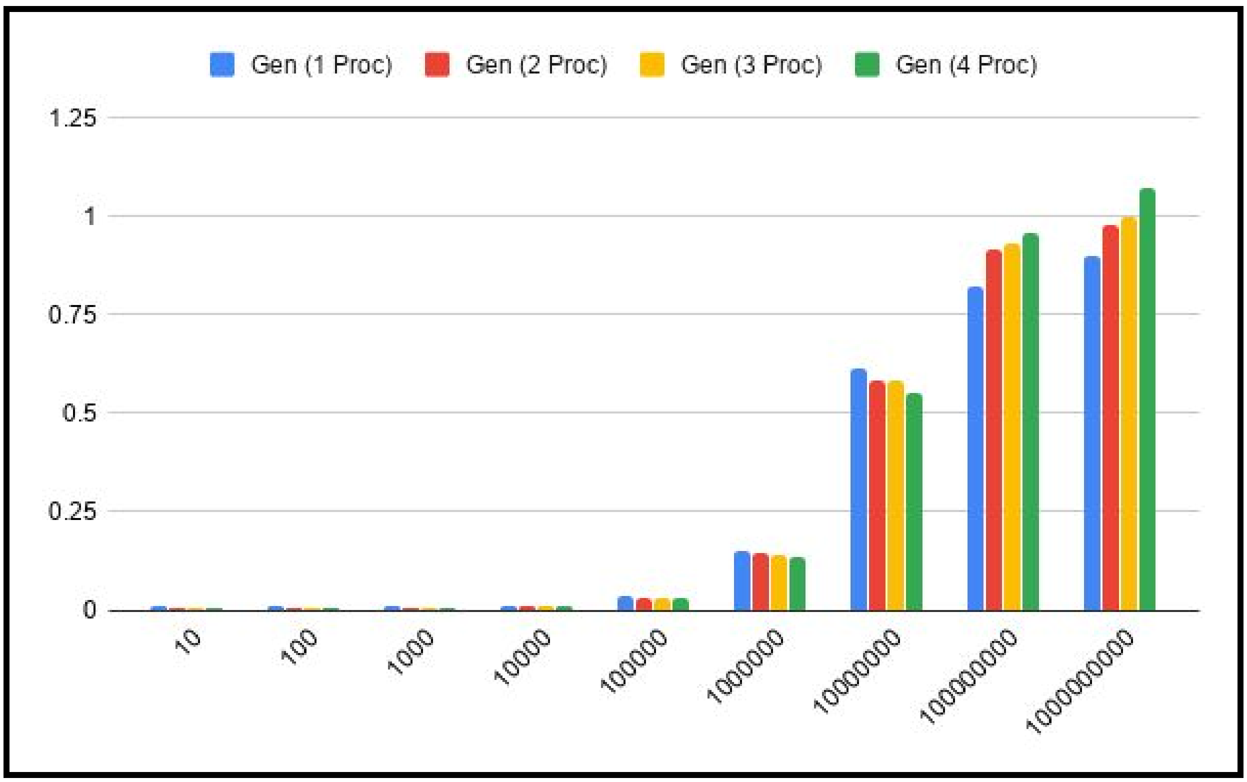
Berwin Gan (wqg203)
Lab 1: Parallel Computing

Speedup Table

N	Gen (1 Proc)	Gen (2 Proc)	Gen (3 Proc)	Gen (4 Proc)
10	0.008	0.0076	0.0069	0.0072
100	0.008	0.0077	0.007	0.0072
1000	0.0079	0.0077	0.0067	0.0071
10000	0.0106	0.0102	0.01	0.0093
100000	0.0339	0.0324	0.032	0.0307
1000000	0.1486	0.1444	0.1407	0.1352
10000000	0.6167	0.5851	0.581	0.5525
100000000	0.8218	0.9187	0.9324	0.9606
1000000000	0.9021	0.9784	0.9998	1.0727

Speedup Chart 1



Berwin Gan (wqg203)
Lab 1: Parallel Computing

Speedup Chart 2

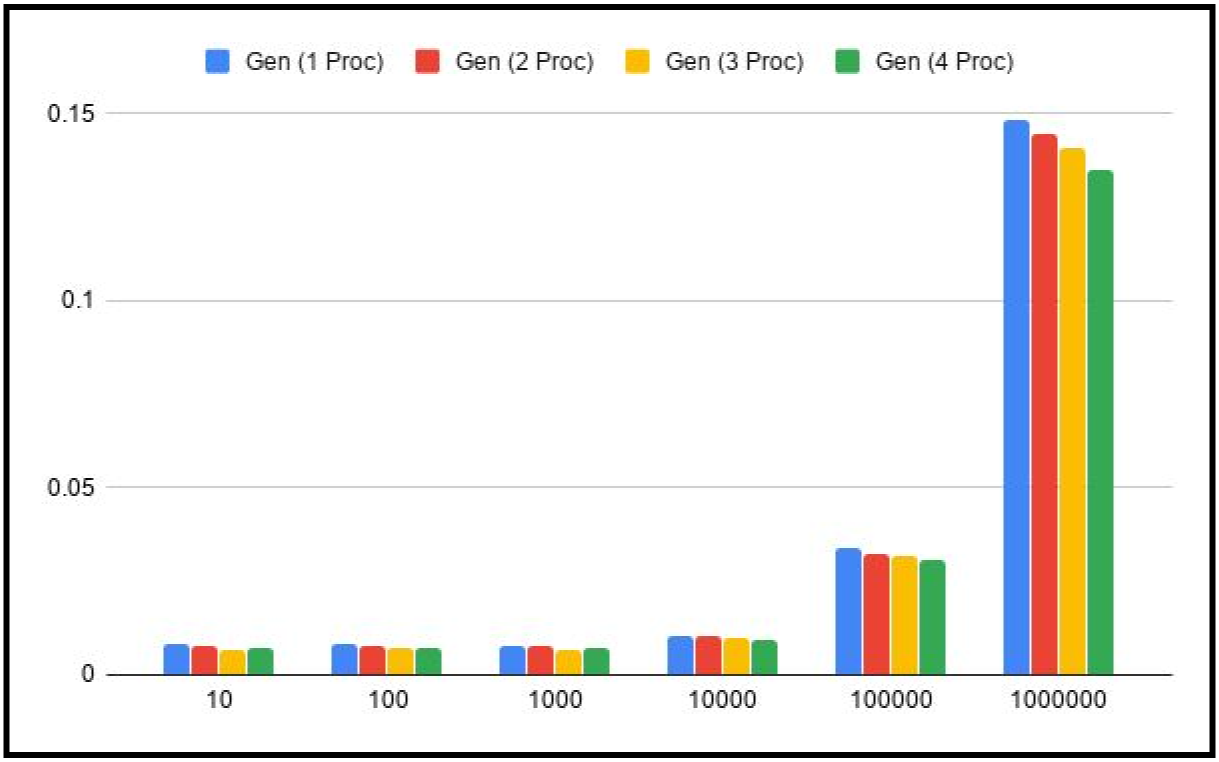


Table of Performance Times

N	Sequential	Gen (1 Proc)	Gen (2 Proc)	Gen (3 Proc)	Gen (4 Proc)
10	0.003	0.373	0.393	0.437	0.419
100	0.003	0.373	0.392	0.428	0.415
1000	0.003	0.381	0.392	0.445	0.424
10000	0.004	0.376	0.392	0.401	0.428
100000	0.013	0.384	0.401	0.406	0.423
1000000	0.066	0.444	0.457	0.469	0.488
10000000	0.674	1.093	1.152	1.16	1.22
100000000	6.928	8.43	7.541	7.43	7.212
1000000000	69.541	77.089	71.074	69.553	64.828

This and all below tables are in seconds with the exception of the ‘N’ column which shows the number which prime numbers were generated for. Gen refers to genprimes.c code and proc refers to how many processes was used.

Berwin Gan (wqg203)
Lab 1: Parallel Computing

The results show that for all but 1 of my result, my parallel code was slower than the serial code. Looking at Speedup Chart 2, it is clear that all the parallel code performed on 0.15 or worse of the speed of the serial code for N up till 1000000. However, looking at Speedup Chart 1, all parallel code, even using 1 processes managed to increase their speed up as the size of the problem increased. The portion of the overhead from calling MPI is becoming smaller and smaller portion of the overall program when N increases. That said, except for 1 case which is when p = 4 and N = 1000000000, all the parallel codes speed up were under 1 which meant that they were still slower.

After running several tests with <time.h>, I learned that the bottleneck was located at the MPI_Reduce which I used to coalesce all the arrays from the difference process. When using 4 processes, I learnt that slightly more than half the time was spent on MPI_Reduce. The overhead of calling and using MPI was simply too great that any speedup from the parallelism was unseen until roughly 4 processes was used with N=10000000000 where for the first time, the parallel code ran faster than the sequential code. I believe that as N increases, any speedup was cancelled by the communication cost required of MPI_Reduce to internally transfer and compare the ever increasing array size. As Prof Zahran said, communication is pretty expensive in comparison to computation. That said, perhaps there is a smarter way to go about parallelizing this lab.

Berwin Gan (wqg203)
Lab 1: Parallel Computing

N [SeqGenPrimes]	1	2	3	4	5	Median
10	0.004	0.003	0.003	0.003	0.003	0.003
100	0.004	0.004	0.004	0.004	0.004	0.003
1000	0.005	0.003	0.003	0.003	0.003	0.003
10000	0.007	0.004	0.004	0.004	0.004	0.004
100000	0.013	0.245	0.014	0.01	0.009	0.013
1000000	0.066	0.067	0.064	0.063	0.067	0.066
10000000	0.67	0.678	0.674	0.678	0.673	0.674
100000000	6.885	6.928	6.821	6.978	7.239	6.928
1000000000	67.223	68.761	69.541	70.001	69.982	69.541

N [GenPrimes (1 Process)]	1	2	3	4	5	Median
10	0.383	0.373	0.372	0.372	0.377	0.373
100	0.38	0.373	0.373	0.373	0.372	0.373
1000	0.381	0.384	0.376	0.373	0.382	0.381
10000	0.374	0.376	0.376	0.375	0.385	0.376
100000	0.384	0.384	0.384	0.399	0.384	0.384
1000000	0.486	0.442	0.443	0.444	0.46	0.444
10000000	1.104	1.101	1.093	1.129	1.114	1.093
100000000	8.746	8.317	8.474	8.43	7.88	8.43
1000000000	82.412	84.55	75.231	77.011	77.089	77.089

N [GenPrimes (2 Process)]	1	2	3	4	5	Median
10	0.414	0.392	0.393	0.39	0.394	0.393
100	0.392	0.39	0.392	0.394	0.404	0.392
1000	0.39	0.391	0.394	0.392	0.403	0.392
10000	0.393	0.392	0.39	0.391	0.406	0.392
100000	0.398	0.402	0.401	0.398	0.409	0.401
1000000	0.454	0.457	0.459	0.456	0.468	0.457
10000000	1.135	1.169	1.126	1.152	1.182	1.152
100000000	7.604	7.541	7.381	7.504	7.784	7.541
1000000000	71.074	73.554	73.125	70.115	71.002	71.074

Berwin Gan (wqg203)
Lab 1: Parallel Computing

N [GenPrimes (3 Process)]	1	2	3	4	5	Median
10	0.407	0.86	0.437	0.586	0.436	0.437
100	0.428	0.427	0.429	0.427	0.45	0.428
1000	0.403	0.436	0.445	0.456	0.456	0.445
10000	0.401	0.402	0.401	0.401	0.417	0.401
100000	0.407	0.516	0.405	0.406	0.405	0.406
1000000	0.486	0.469	0.464	0.465	0.475	0.469
10000000	1.16	1.091	1.161	1.194	1.159	1.16
100000000	7.424	7.749	7.767	7.43	7.416	7.43
1000000000	72.441	69.553	69.501	70.143	69.091	69.553

N [GenPrimes (4 Process)]	1	2	3	4	5	Median
10	0.423	0.412	0.42	0.419	0.416	0.419
100	0.453	0.414	0.428	0.417	0.425	0.425
1000	0.423	0.42	0.424	0.431	0.44	0.424
10000	0.415	0.428	0.426	0.428	0.432	0.428
100000	0.423	0.422	0.426	0.42	0.427	0.423
1000000	0.525	0.483	0.485	0.488	0.498	0.488
10000000	1.101	1.237	1.241	1.188	1.22	1.22
100000000	7.212	8.234	6.806	7.043	7.889	7.212
1000000000	67.123	66.321	64.452	64.828	62.443	64.828