Parallel And Distributed Computing

Fall 2014 Assignment 1 : Report

ABSTRACT

The report is case study of how different parallel computation perform by varying the size of problem, number of node and number of threads. The problem taken here is simple algorithm to compute all the prime numbers from 1 to the given range. The finally based on the result graph is plotted and compared.

By

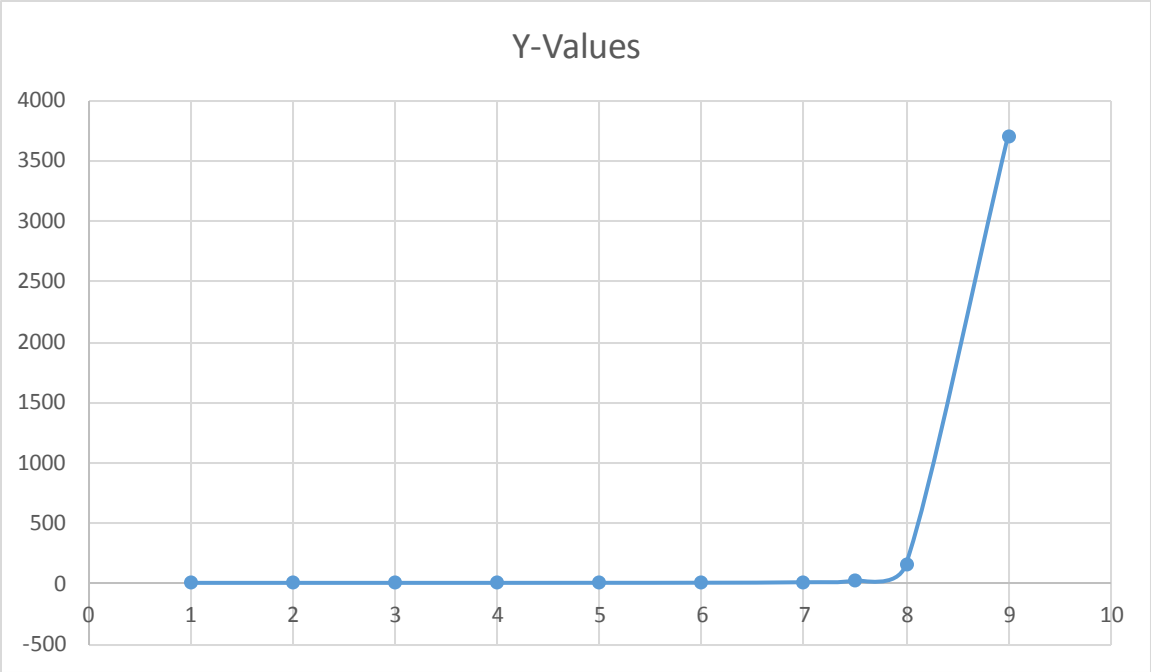
Vijay Manoharan
manohara@buffalo.edu
#50097716

Table of contents.

Sequential

Given algorithm for finding prime numbers between 1 and N, N is the positive integer. The input is the value of N and output is the number of primes and largest prime number in the range.

The graph between size of problem and time has plotted. The size of the problem is increased in the multiples of 10. As the scaling of the problem increases the execution time is increased expotential.



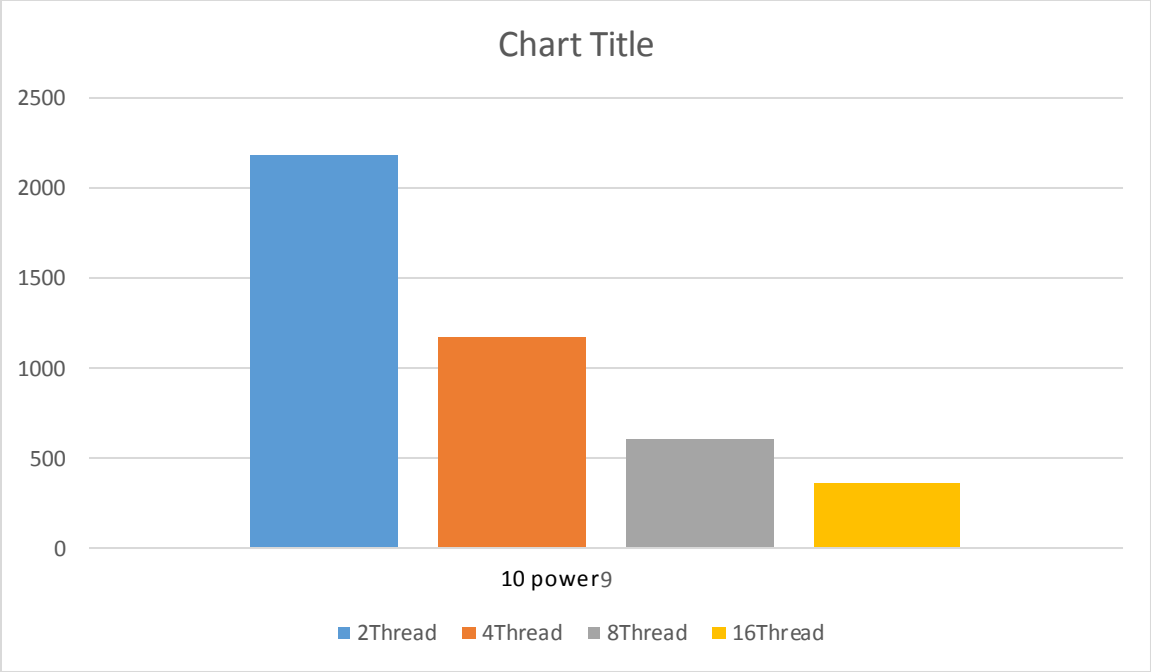
Graph and table of data collected during the program execution.

10 power x	Time in Secs
1	0.001339
2	0.001140
3	0.001225
4	0.001532
5	0.008619
6	0.188392
7	4.998390
8	137.516152
9	3651.332125

The data for varying the data size and time is recorded in the table shown.

Open MP

The algorithm is parallel executed by creating multiple threads in order to get computation benefits.The data have been recorded as below. And compared in the tabulation below. From the tabulation we can see that the amount of time taken to compute decreases as the number of threads are increased. The plot it made between the number of threads and time taken for a constant size 10^9.



10 power x	2 Thread	4 Thread	8 Thread	16 Thread
1	0.001412	0.002349	0.026998	0.026795
2	0.002815	0.001959	0.011207	0.036301
3	0.021634	0.00243	0.003257	0.026218
4	0.022672	0.001989	0.003108	0.058689
5	0.042452	0.003127	0.006021	0.05783
6	0.15869	0.053991	0.054097	0.060523
7	2.742629	2.33538	0.954322	0.792727
8	75.886118	40.91231	25.98343	15.92639
9	2190.003702	1176.282	607.078	365.2304

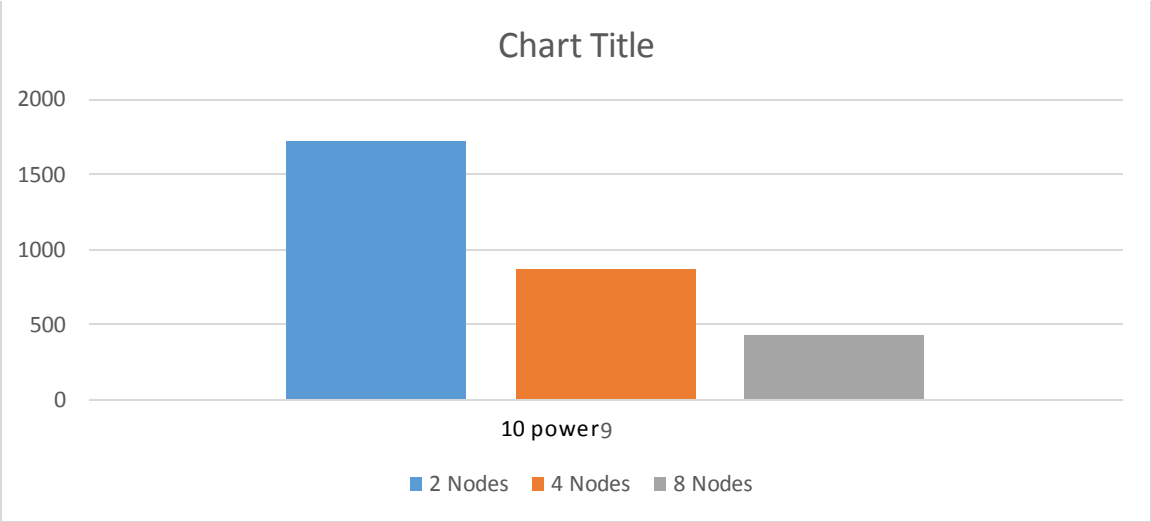
From the graph we can clearly see that the time taken is inversely proportional to the number of threads. This condition is true for larger size, however when noticed for smaller problem size the executiontimeis longer than for increased number of threads. This is due to the fact that the problem takes more computation time for dividing problem the problem. That is, latency for multithreading, creation of threads and memory access by the threads. There by the processor spends more time in accessing the memory than computation.

MPI

The algorithm is paralleled by executing on multiple processor in order to get computation benefits by using MPI libraries. The execution results have been recorded as below in table. And compared areplotted in the graph. From the tabulation we can see that the amount of time taken to compute decreases as the number of threads are increased. The plot it made between the number of processor and time taken for a constant size 10^9.

The processor are increased in order of 2, 4 and 8.

10powerx	2 Nodes	4 Nodes	8 Nodes
1	0.007709	0.025735	0.021723
2	0.00888	0.013192	0.02093
3	0.009936	0.012594	0.022634
4	0.010458	0.014139	0.021272
5	0.012386	0.014862	0.021706
6	0.096709	0.056798	0.04613
7	2.447468	1.235968	0.689578
8	61.03986	32.57695	16.08933
9	1720.288717	865.912	433.0501



We can see that as the scaling of the problem increases the execution time also increases, but here since the problem is executed in parallel the execution time is reduced in the multiples of scaling factor.Which is clearly seen from the above graph. However for the smaller size we can see that the parallel takesmore time, that is because of the latency that is created in dividing the problem is more than the execution of the problem.

SSE-AVX

In SSE-AVX the problem is made parallel by doing the arithmetic operation using Vector address, using __m128 data types. Since Vector addresses are faster to access and we can get 4 times the scalability. However if the problem size increase more than 2^24 the scalability can be only 2 times. As the most of the operation using vector array can be done only using 32-bit float values.

So the problem is scalable from 2 to 4 times depending on the size of the problem. And also since mode function is not available in SIMD SSE instructions. The following formula is used for computing the mode value to check a given number is prime or not.

X mod Y = X – Y* (floor (X/Y))

In terms of vector calculation I formulated it as

Vect_Rem=__mm_sub_ps(X, _mm_mul_ps(Y, _mm_floor_ps(_mm_div_ps(X,Y))))

The following scaling was obtained using SSE.

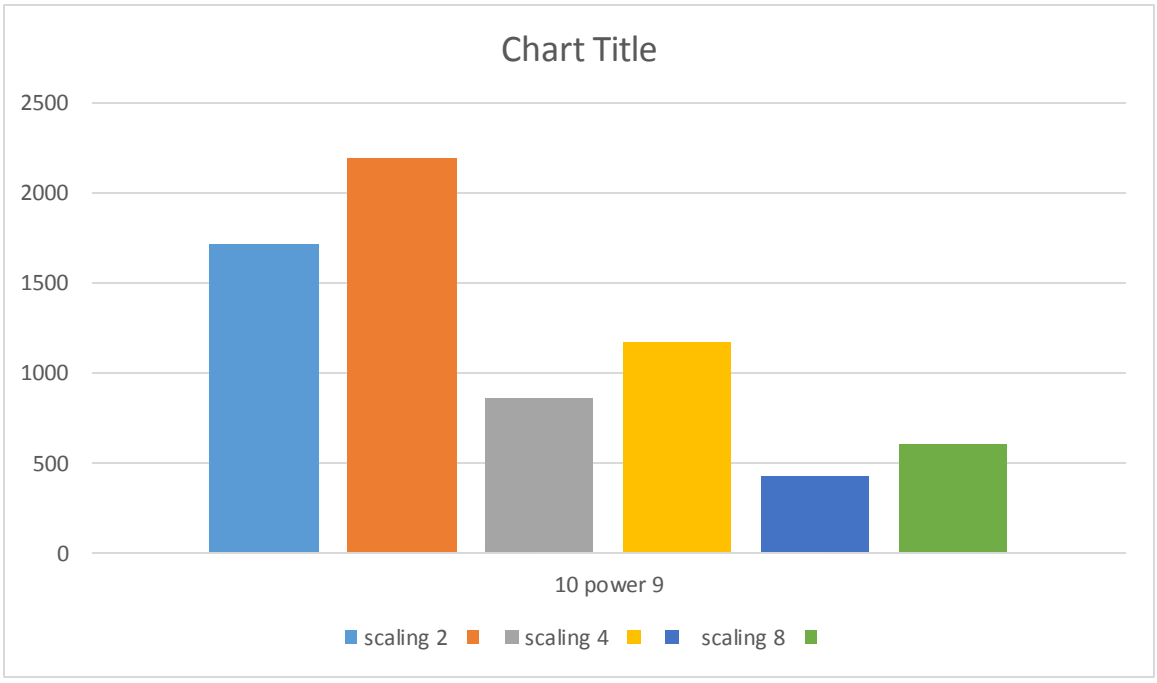
10powerx	Sequential	SSE-AVX
1	0.001339	0.000026
2	0.001140	0.001362
3	0.001225	0.001483
4	0.001532	0.001692
5	0.008619	0.007472
6	0.188392	0.156418
7	4.998390	4.126551
8	137.516152	105.708
9	3651.332125	2912.462

Comparison between openmp and MPI.

Let us compare two approached that gave better scaling in execution in terms of the numberofparallelism used.

10powerx	2 scaling Factor		4 scaling Factor		8 scaling Factor	
	MPI	OpenMP	MPI	OpenMP	MPI	OpenMP
1	0.007709	0.001412	0.025735	0.002349	0.021723	0.026998
2	0.00888	0.002815	0.013192	0.001959	0.02093	0.011207
3	0.009936	0.021634	0.012594	0.00243	0.022634	0.003257
4	0.010458	0.022672	0.014139	0.001989	0.021272	0.003108
5	0.012386	0.042452	0.014862	0.003127	0.021706	0.006021
6	0.096709	0.15869	0.056798	0.053991	0.04613	0.054097
7	2.447468	2.742629	1.235968	2.33538	0.689578	0.954322
8	61.03986	75.88612	32.57695	40.91231	16.08933	25.98343
9	1720.288717	2190.004	865.912	1176.282	433.0501	607.078

Comparison is show below for the larger problem size and different scaling factor. From the comparison, MPI out performs openmp.



OpenMPI(openmp + MPI)

The scaling factors of open and mpi are combined, now the algorithm is made to rum on multithreadsand multiprocessors to get much more scaling factor.

Here are the results of the computation, the node of nodes are kept constant and threads are varied to see the execution time. We can see as the number of threads are increased the execution time isreduced to a greater extend. Thus providing multiple scales of parallelism.

10 power x	number of threads		
	1	2	4
1	0.007709	0.086933	0.098087
2	0.00888	0.086381	0.103224
3	0.009936	0.087501	0.110245
4	0.010458	0.080957	0.132408
5	0.012386	0.09191	0.103714
6	0.096709	0.156069	0.269358
7	2.447468	0.638805	0.374777
8	61.03986	17.1768	9.577926
9	1720.288717	431.5131	245.8774

Similarly keeping the number of threads constant and increasing the number of nodes, the following results is obtained. Similar results have been obtained that is multilevel parallelism.

10 power x	number of nodes		
	1	4	8
1	0.002349	0.098087	0.137282
2	0.001959	0.103224	0.13176
3	0.00243	0.110245	0.114746
4	0.001989	0.132408	0.127061
5	0.003127	0.103714	0.133405
6	0.053991	0.269358	0.153668
7	2.33538	0.374777	0.488151
8	40.912312	9.577926	4.326781
9	1176.281911	245.8774	121.0424

Summary

From all the above results we have the following conclusion.

- As the size of the problem increases the execution time increases exponentially.
- As the number of thread increase with the constant problem size, the execution time decreases by the multiple factor of the thread used.
- As the number of cores/nodes/processor increases with the constant problem size, theexecution time decreases in the factor of the number of nodes.
- SSE does provide a better scaling than that of the sequential problem.
- By combining the scaling factors of Openmp and MPI, we get multiple scaling factor. Say if 4 core and 2 threads are used in the program execution, we get 8x parallelism. Thus reducing the execution time by 8 times the sequential execution for the given problem size.

Thus all the result obtained is similar to the theoretically and matches the scaling factor approximately.

Also the largest prime number and number of prime number for the given problem size has been found. The following results have been obtained.

10 power x	Number of primes	Largest prime
1	4	7
2	25	97
3	168	997
4	1229	9973
5	9592	99991
6	78498	999983
7	664579	9999991
8	5761455	99999989
9	50847534	999999937