Parallel matrix norms using OpenMP

Marcus Lee
marcustutorial@hotmail.com

November 25, 2023

1 Introduction

The aim of this experiemnt is to discover the difference between serial and parallel algorithm for computing matrix norms. The parallel algorithm used in this experiemnt was imlemented using **OpenMP** and the benchmark was ran on a machine with 8 x Intel(R) Xeon(R) E5-2620 v3 @ 2.40GHz processors. The program uses the number of processors as the number of threads hence the results for parallel algorithm shown in following sections uses 8 threads.

All the matrix multiplication algorithms used in this experiemnt were manually written without using any third-party dependencies both for the serial and the parallel algorithm.

2 Dependence of program execution time on matrix size

Since the matrix size n must be divisible by the number of threads, the benchmark was ran with n ranging from 128 to 1792 with a step of 128. Figures below show the execution time for the serial and parallel algorithm with increasing matrix size n.

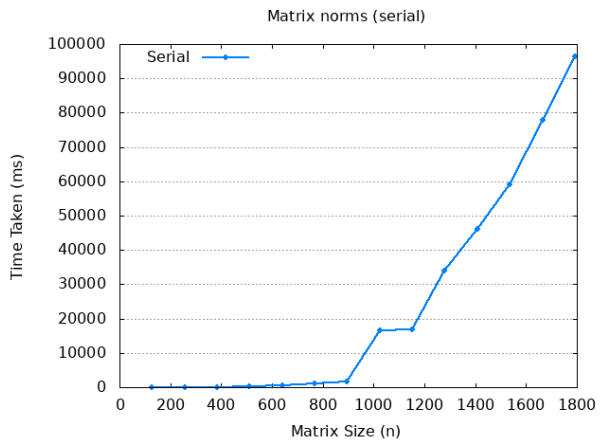

Figure 1: Matrix norm with serial algorithm

images/benchmark_parallel_openmp.png

Figure 2: Matrix norm with parallel algorithm

From the results above, we can deduce that as matrix size n increases, execution time increases. This is true for both cases because the number of computation needed increases.

3 Speedup of parallel algorithm over serial algorithm

To further investigate on the speedup of the parallel algorithm over the serial algorithm, Figure 3 below shows the plot of the execution time for the serial algorithm and the parallel algorithm on the same graph. Moreover, Table 1 below includes the execution time for both algorithms for different matrix size n.

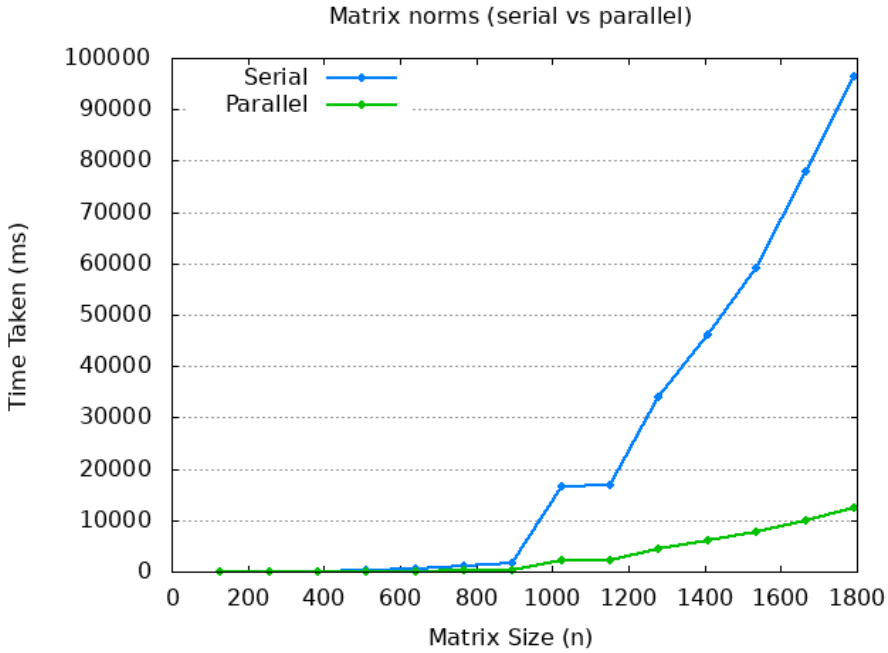

Figure 3: Matrix norm benchmark

Matrix size (n)	Time taken for serial (ms)	Time taken for parallel (ms)
128	2.52	8.88
256	37.96	26.85
384	117.79	55.16
512	296.34	75.57
640	565.67	122.97
768	987.15	201.13
896	1589.10	278.08
1024	16625.47	2221.06
1152	16897.73	2229.01
1280	33874.76	4455.29
1408	46162.84	6041.34
1536	59080.25	7733.39
1664	77936.19	10077.36
1792	96502.45	12328.03

Table 1: Time taken for serial and parallel algorithm

From the results shown above, the average time taken for the serial algorithm is $25048.32ms$ and $3275.29ms$ for the parallel algorithm. The speedup is **86.9%** calculated using the formula:

$$\frac{(avg\ time_{parallel} - avg\ time_{serial})}{avg\ time_{serial}} \cdot 100\%$$

.

That is a very significant speedup overall however the speedup depends heavily on the matrix size n. For example, when $n = 128$ the parallel algorithm is slower than the serial algorithm by **3.5** times. This is due to the overhead of creating and joining threads which is more expensive than the computation itself. However, when $n = 1792$ the parallel algorithm is faster than the serial algorithm by **7.8** times. This is because as the input size increases, the overhead of creating and joining threads decreases comparatively.