Parallel matrix norms using Pthreads

Lee Kai Yang (23205838)
kai.y.lee@ucdconnect.ie

November 12, 2023

1 Dependence of program execution time on matrix size

The benchmark was ran on a machine with 8 x Intel(R) Xeon(R) E5-2620 v3 @ 2.40GHz processors.

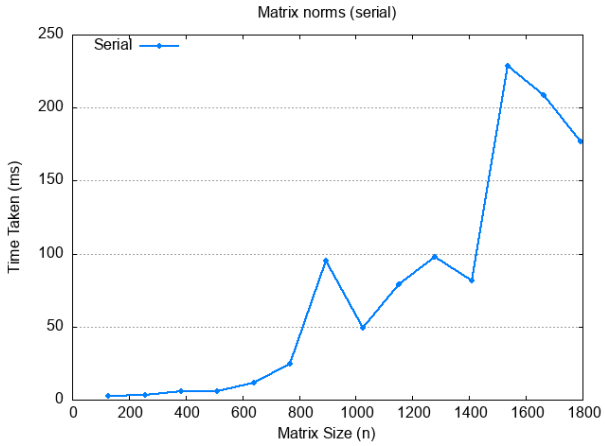

Figure 1: Matrix norm using serial algorithm

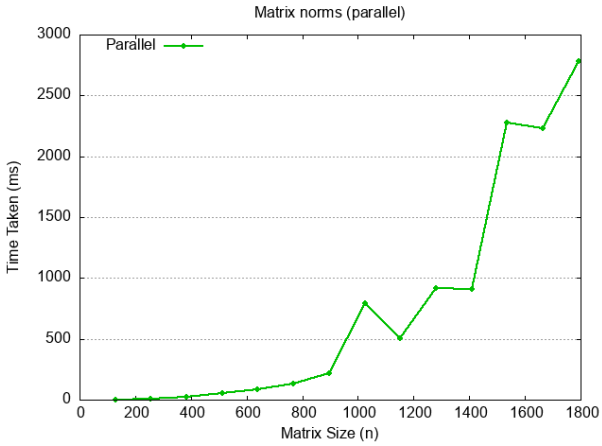

Figure 2: Matrix norm using pthreads

2 Speedup of blocked algorithms over non-blocked

In this section, I will be experimenting with different matrix sizes n and compare the speedup between the blocked and non-blocked matrix multiplication algorithms. I will be using **ATLAS** for the blocked *ijk* algorithm and **BLAS** for the non-blocking dgemm routine. For each algorithm, I ran 11 iterations with increasing matrix size n from 1024 to 4096 with a step of 384. Table 1, Figure 1 and Figure 2 below shows the result between the blocked and non-blocked algorithm.

Matrix size (n)	Time taken for non-blocked (ms)	Time taken for blocked (ms)
1024	239.12	251.55
1331	545.80	280.57
1638	948.60	242.28
1945	1591.63	250.95
2252	2403.19	1825.92
2560	3511.97	1831.08
2867	4900.08	1828.35
3174	6570.59	6056.96
3481	8719.13	6078.35
3788	11159.21	6078.67
4096	14086.89	14282.82

Table 1: Time taken for blocked and non-blocked algorithm

From the results shown above, the average time taken for the non-blocked algorithm is 4970.56ms and 3546.13ms for the blocked algorithm. The speedup is 40% calculated using the formula:

$$\frac{(avg\ time_{blocked} - avg\ time_{non-blocked})}{avg\ time_{non-blocked}} \cdot 100\%$$

.

3 Influence of block size on cache utilization and execution time

The benchmark was ran on a machine with 8 x Intel(R) Xeon(R) E5-2620 v3 @ 2.40GHz processors but the number of cores is not important in this case, since the computation is single threaded. From the output of **lscpu**, the L1d cache is 32KB, L1i cache is 32KB, L2 cache is 256KB and L3 cache is 15360KB. Since only the L1d cache is storing data, the size of a single block should not exceed 32KB. The size of a single *double* is 8 bytes, let b = block size, the optimal block size should be:

$$\begin{aligned} b * b * sizeof(double) &= sizeof(cache) \\ b^2 * sizeof(double) &= sizeof(cache) \\ b &= \sqrt{\frac{sizeof(cache)}{sizeof(double)}} \\ b &= \sqrt{\frac{32 * 1024}{8}} \\ b &= 64 \end{aligned}$$

Since L2 cache is 256KB, the optimal block size should be:

$$b = \sqrt{\frac{256 * 1024}{8}}$$
$$b = 181$$

Since L3 cache is 15360KB, the optimal block size should be:

$$b = \sqrt{\frac{15360 * 1024}{8}}$$
$$b = 1402$$

I have ran the blocked ijk ATLAS routine with 10 different block sizes including the optimal block sizes calculated above. The results are shown in Figure 3 below.

From the results above, we can deduce that as block size increases, execution time decreases. However, there are a few block sizes that shown a significant decrease in execution time compared to others which are 64, 181 and 1402. The reason being that these block sizes are the optimal block sizes that fits the L1d, L2 and L3 cache respectively.

The fastest of them all is the block size 1402 but the execution time for the first few matrix sizes are zero because the block size is larger than the matrix size and therefore the algorithm is not executed. Hence, choosing a suitable block size based on the matrix size is equally important as choosing the optimal block size. One of the ways to mitigate this issue is to add a check to see if the block size is larger than the matrix size and if it is, set the block size to the matrix size.