ECEC 413
Assignment 1 – Ways to SAXPY
Performance analysis and report
Aleksandar Aleksandrov

1. Introduction

For this assignment two ways of parallelizing a SAXPY loop are implemented – chunking and striding. The two methods' performance is then compared with the synchronous code. The chunking method splits the two arrays into equally sized chunks and one thread operates on one chunk. For the striding method every thread operates on equally spaced indexes of the arrays. The spacing is determined by the number of threads that are used. For example, if we have 4 threads, thread 0 will operate on items 0,4,8 etc.

2. Results

Threads	Elements	Time Synchronous (s)	Time Chunking (s)	Time Striding (s)
4	10^4	0.000010	0.000753	0.000198
4	10^6	0.000820	0.001289	0.001756s
4	10^8	0.110732	0.036838	0.108770
8	10^4	0.000010	0.002217	0.000405
8	10^6	0.000890	0.001380	0.002091
8	10^8	0.123012	0.062605	0.164438
16	10^4	0.000010	0.002334	0.002022
16	10^6	0.000876	0.001501	0.002833
16	10^8	0.078600	0.053498	0.196057

The table above shows the achieved results from the parallelization on the Drexel Xunil Machine. From the table we can infer that multiple threads do not always provide performance improvement and also the method to parallelize the program could also give worse performance than if the program runs synchronously.

For example, for arrays with size 10^4 the synchronous code gave better times when compared to chunking and striding no matter the used cores. This could be from the fact that when the arrays have a small number of elements, when the CPU requests data from the main memory, the predictor utilizes spatial locality and brings most of the next elements in the array into the Cache. Therefore, for a small number of elements there is a smaller amount of data going from the main memory to the cache and the memory speed bottleneck has a lesser impact on performance. In this case, utilizing multiple threads does not provide any value since the arithmetic operation is simple.

For arrays of size 10^6, the multithreaded programs also do not provide any performance improvement, however, now we can see that the time it took to perform those operations is significantly closer between the synchronous implementation and chunking and striding. Therefore, for larger array size we can expect the multithreaded programs to give some performance improvement.

For array of size 10^8, we can finally see that the chunking method performed the operation for a shorter amount of time than the synchronous method. For the larger array, the synchronous program gets slowed down by the large amount of data that has to be brought from the main memory. In this case, having more threads alleviates this bottleneck. However, we can see that the striding method, however, performs worse than both the synchronous program and the chunking method. This could be due to the fact that this method does not utilize spatial locality. Every thread requests array elements from main memory which are spaced out by the stride. In this case, when the thread 0 requests element 0, the processor brings in a whole block from the cache which would include elements 0,1,2,3. However elements 1,2,3 are not utilized by that core at all, and for the next operation it will have to pull element 4 from the memory.

In addition, we can also notice that generally, increasing the number of cores does provide a performance improvement, however, this improvement is not linear. Therefore, for some cases going to 16 cores might not be the most optimal decision.