Blake Cruz

Code Report for Project 3

For this project I was required to write two different parallel programs based on the serial solution which involves the multiplication of 2 matrices. This sequential solution is very inefficient and can be improved by implementing parallel programming. The first solution uses a very simple approach where we implement the line “pragma omp parallel for” to handle the multithreading. The second approach separates the matrices into blocks and calculates these blocks separately in different threads.

**Code Explanation:**

The first solution (parallel1.c) needs to utilize the imported library of <omp.h> to do a lot of the heavy lifting for parallelization. This was also hinted at as a solution given that it was also imported in the serial solution. I used the OpenMP Directive “pragma omp parallel for” inserted before the outermost loop to help parallelize this loop across the multiple available threads. I also chose to include “pragma omp simd” to help vectorize the loop as well to try and help the program run even faster. It is important to note that the algorithm for solving the matrix multiplication actually remains the same, I am just splitting the work to other threads.

The second solution (parallel2.c) improves upon the previous solution by dividing the matrix into blocks denoted by the block size defined at the start of the program. I have chosen to use the block size of 32. While at first glance it may seem that this program would run way slower due to the amount of nested loops, it in fact ends up being the fastest parallel program I could create. We again start by using OpenMP by adding “pragma omp parallel collapse(2)” to the outermost loop to split this process among separate threads. Within each of the blocks we will perform the matrix multiplication for the specific bound set for that block. We can better utilize the cache by using blocks because each of the block’s fits into the cache and will overall decrease the amount of misses from the cache. We then use these blocks to form a result in the resultValue.

**Analysis:**

It should be noted that I chose to measure the time in nanoseconds similar to how I measured it in project 2. I did this because I was more comfortable programing the clock this way and it gave me less trouble than the provided time measurement system in the serial solution. Below is the data I collected and then displayed in a line chart.

A screenshot of a computer

Description automatically generated

This data shows us that the performance for parallel2 (the block based one) is by far the best scaling algorithm. The sequential solution was not much slower at the smaller sizes of matrices, but the sequential solution exponentially took longer to complete compared to the other choices. When dealing with a matrix size of 2048 we saw that parallel 1 performed the task 2.62 times faster than the sequential solution and the parallel 2 solution performed 16.99 times faster than the sequential solution. We can also observe the trend line for parallel 2 and see that there is no clear uptick in performance time in the near future. Having the matrix divided into blocks seems to have the greatest scalability for future computation. Having this block implementation allows the cache to have an improved hit rate because a part of the data for the matrix for each block is put into the cache to help with the performance.

Comparing the data for my parallel 1 solution to my parallel 2 solution we can find that parallel 2 runs 6.49 times faster than parallel 1 when running with a matrix size of 2048. Per forming the parallelization without blocks still lead to an unnecessary performance hit and the graph above shows that the time roughly doubles after each increment of 512 matrix size. Compare this to the parallel 2 solution where it is much more scalable, with only a block size of 32. This block approach allows for better cache utilization because each of the threads is dealing with a certain block size instead of just the parallelized version of the serial loop to calculate the multiplication of two matrices.

**OUTPUT:**

**KEY FOR FILE NAMES:**

parallel1.c = task 1 using parallelization

parallel2.c = task 2 using block-optimized approach

simple.c =sequential solution

With a high value of N(matrix size) (2048):

A screen shot of a computer

Description automatically generated

With a medium value of N(matrix size) (1536):

A screen shot of a computer

Description automatically generated

With another medium value of N(matrix size) (1024):

A screen shot of a computer program

Description automatically generated

With a low value of N(matrix size) (512):

A screen shot of a computer

Description automatically generated