MYE023: OpenMP Assignment

Due on Monday, Apr 27, 2020

Instructor: Vassilios V. Dimakopoulos

Alexandros Alexiou AM. 2929

Apr 19, 2020

Contents

1 Matrix multiplication parallelizing for loops 3
1.1 Parallelizing outer for loop . . . 3
1.2 Parallelizing middle for loop . . . 3
1.3 Parallelizing inner for loop . . . 3
1.4 Results . . . 4
1.5 Comments about results . . . 4

2 Parallel calculation of prime numbers up to a specified limit 5
2.1 Parallelizing for loop . . . 5
2.2 Results . . . 5
2.3 Comments about results . . . 5

3 Parallel matrix multiplication with Checkerboard partitioning. 6
3.1 Checkerboard partitioning . . . 6
3.2 Using OpenMP tasks. . . . 6
3.3 Results using various S values and 1024 x 1024 matrices . . . 6
3.4 Comments about results . . . 7
3.5 Results accuracy . . . 7

1 Matrix multiplication parallelizing for loops

1.1 Parallelizing outer for loop

We first try to parallelize the outer loop of the serial program. We declare the pragma directive with private variables(i, j, k, sum) and we will run the calculation with static and dynamic scheduling. Each thread will have its own set of the declared private variables and the only shared ones are the arrays A, B and C such that we have the minimum access to shared variables in order to make the calculation as fast as possible.

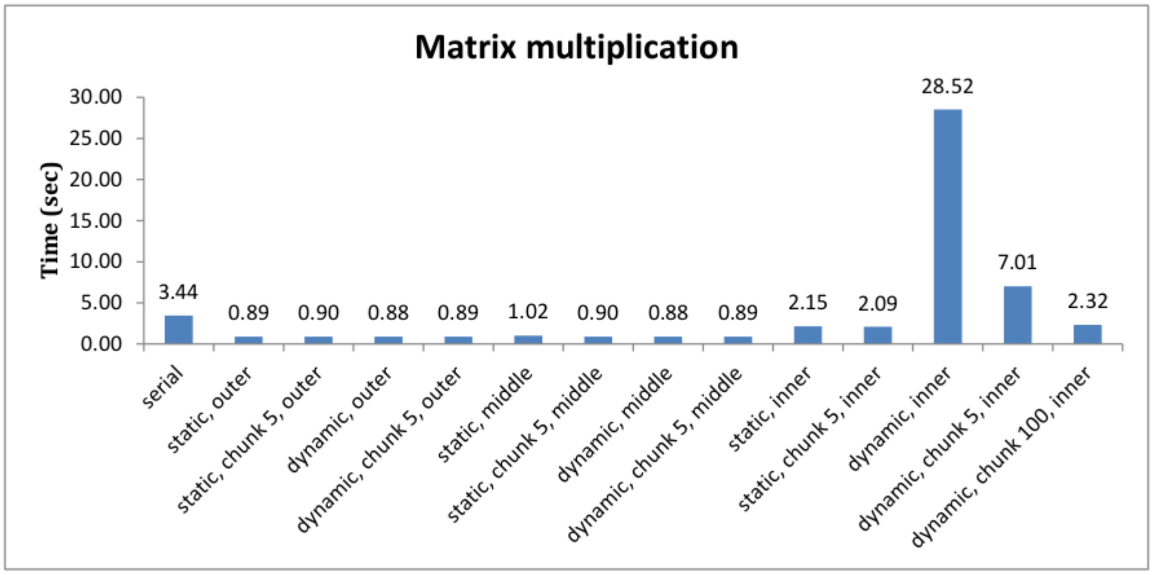
1.2 Parallelizing middle for loop

Then we try to parallelize the middle for loop of the serial program. Now things change a little bit. Each thread will have the following private variables(j, k, sum) but now they need access to the i parameter of the outer for loop. We expect that this approach will be worse than the previous one because we have more shared variables, but not by much.

1.3 Parallelizing inner for loop

Finally, we try to parallelize the inner for loop of the serial program. Now things change a lot because the inner for does the inner product. Each thread adds it to the sum variable and then assigns the value sum to the appropriate cell of the C array. This means that the sum variable cannot be private anymore and cannot be shared either unless we add mutual exclusion which will make the calculation a lot slower. We will take a different approach, using the reduction clause with the plus operator. The reduction clause lets us specify one or more thread-private variables that are subject to a reduction operation at the end of the parallel region. Here we will declare the sum variable to be in the reduction clause. In this way we save a lot of time. We still have the other shared variables(i, j, A, B, C) but we cannot do something about it. We expect the inner for calculation time to be the worst one of all 3 attempts.

1.4 Results



The diagram above shows the detailed results using different scheduling types utilizing 4 threads.
All measurements were done using the laboratory machine with id: opti3060ws08

1.5 Comments about results

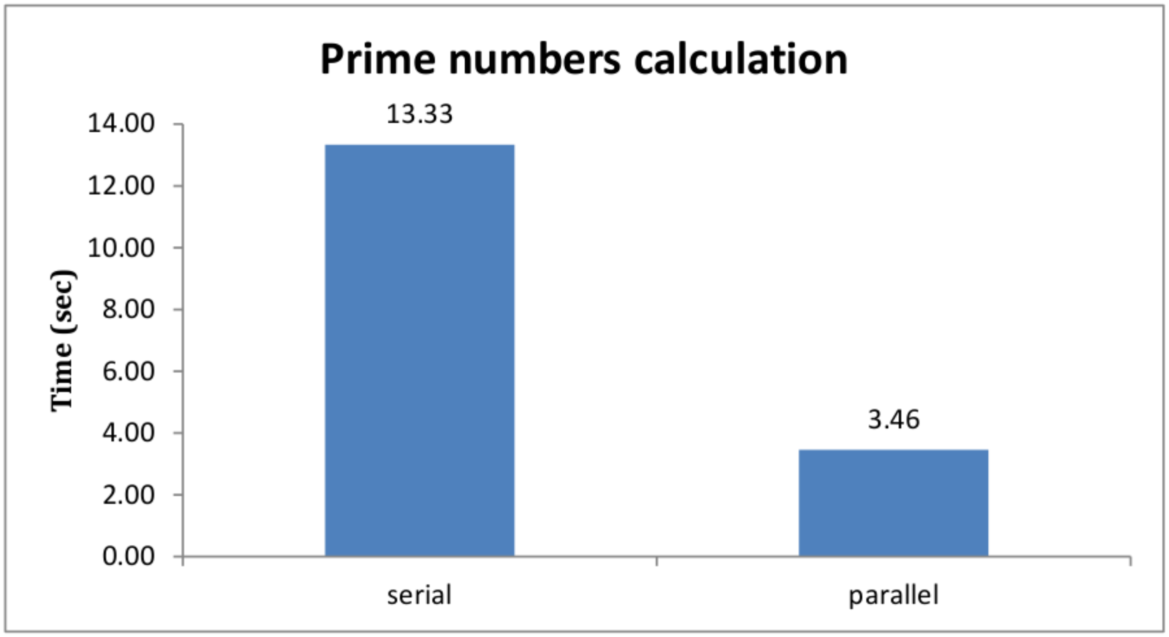
Parallelizing the inner for with the default dynamic schedule with no Chunk size(it is set to 1 by default) we can see that the calculation is 14 times slower than the static schedule. That is because the threads are ”racing” a lot to get the calculation done and as said before we have access in many shared variables which has a big overhead. The whole point of dynamic scheduling is to improve the distribution of work in the case where each loop iteration does not contain the same amount of work.
Using chunk size = 5 in the inner loop we can see that it is 3,5 times slower. There is an improvement but it is still about 2 times slower than the serial calculation. Making the chunk size bigger, the threads are ”racing” less so we get better results but the best way to go here is with the parallelization of the outer for using static scheduling distributing the load evenly among the threads.

2 Parallel calculation of prime numbers up to a specified limit

2.1 Parallelizing for loop

Here we have a serial program that calculates prime numbers up to a specified limit and prints the total ammount of prime numbers found as well as the last prime number of the limit.
We can see that there is a for loop so we will start by parallelizing it. The following variables will be private (i, num,divisor ,quotient ,remainder). The count and the lastprime values need attention. The count variable is a counter that counts the number of prime numbers found. So we will need mutual exclusion unless we use the reduction clause with plus operator. We will go with the reduction clause. The lastprime variable holds the last prime number calculated. Mutual exclusion here is useless because we need the last value of the variable (serial program). We will use the reduction clause with the max operator so we will get the max number of the lastprime variable calculated by each thread and it is exactly what we need.

2.2 Results



The diagram above shows the detailed results utilizing 4 threads.
All measurements were done using the laboratory machine with id: opti3060ws08

2.3 Comments about results

We can see that the parallel calculation times is 3,8 times faster that the serial one.

3 Parallel matrix multiplication with Checkerboard partitioning.

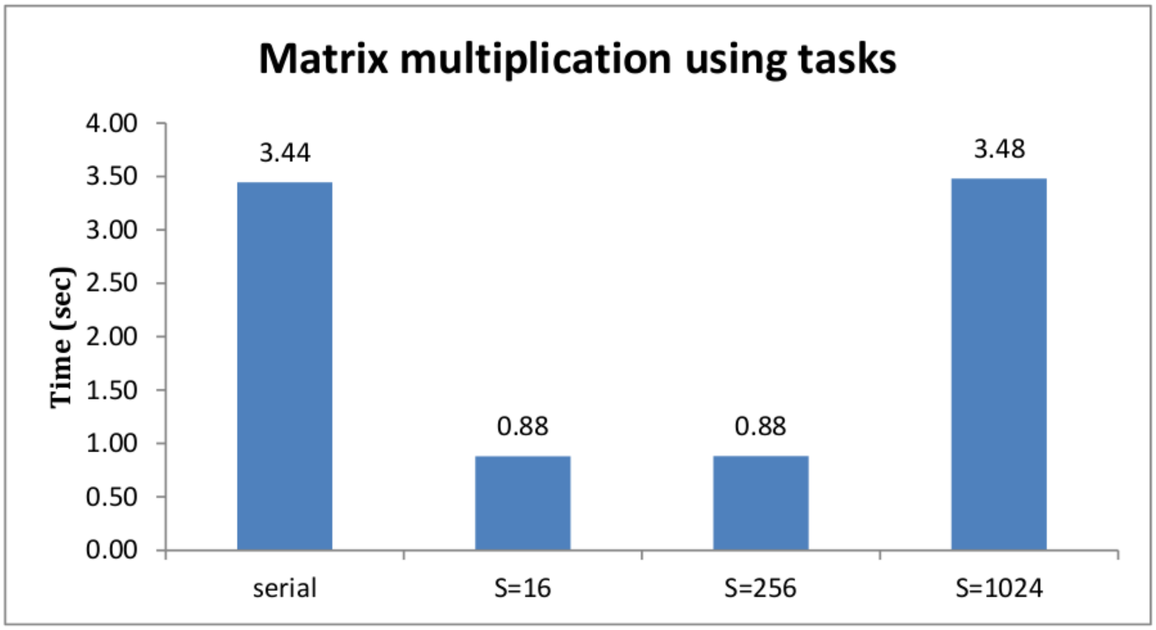
3.1 Checkerboard partitioning

Checkerboard partitioning is a technique in which we divide the matrices(n x n) we need to multiply into submatrices and then we define OpenMP tasks to handle each submatrix multiplication. In our case we need to pass a value S in our program which corresponds to each submatrix dimension(S x S). For example if the matrices we want to multiply are 1024x1024 and we want 128 x 128 (S = 128) submatrices we will schedule 1024 / 128 = 8 tasks.

3.2 Using OpenMP tasks.

We start by adding a pragma directive making a parallel region. Then we add a pragma omp single directive needed for the task scheduling(One thread sets each task but it might not be the one that actually executes the task). Then we add 2 for-loops to iterate through the matrix of submatrices(size will vary depending on the S given). Then we add the pragma omp task directive to schedule the task passing the submatrix coordinates as firstprivate variables for the task. All the other variables are private(i, j, k, sum). Then inside each task we add a formula to the first 2 for-loops to dynamically calculate each submatrix dimensions given the submatrix coordinates.

3.3 Results using various S values and 1024 x 1024 matrices



All measurements were done using the laboratory machine with id: opti3060ws08

3.4 Comments about results

Using value S=16(64 tasks) or S=256(4 tasks) we can see that we have a very good result in terms of time of completion. The interesting thing is that using S=1024(1 task) we can see that the time of completion is about the same with the serial time which makes sense because we use only 1 task as if we are making a serial calculation.

3.5 Results accuracy

For the accuracy of the parallel matrix multiplication results a script was developed that checks if the result matrix of the parallel calculation is equal to the serial one.