Report: Multi-computing using OMP

I made two programs in my Visual Studio Solution. One of them is dedicated to the Matrix calculation and the other one to the Integral calculation.

To try out the code, make sure you run the right program in entry.

I) Matrix calculation

This programs runs the calculation of the following operation:

N defines the size of the Matrix and Array (if the affected value is increased, more calculations are required)

2 vectors A and B of N dimension

1 Matrix C of NxN dimensions

R is the resulted vector

The operation is:

R = A + B*C

Results

With N = 5000

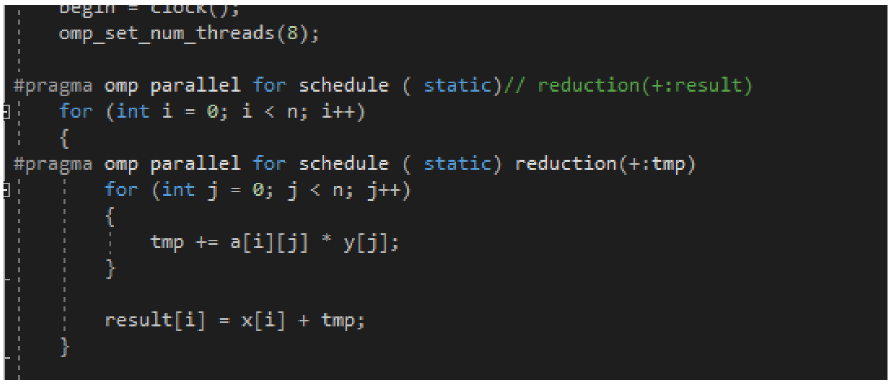
Without Multi-computing

Average treatment time: 0.064s

With Multi-computing

Average treatment time: 0.032s

For this exercise, I had a hard time configuring the algorithm because resulting times were almost the same with or without using Multi-computing. However, I was able to get good performance with multi-computing using “schedule”. I also use a reduction to avoid calculation issues



```
    begin = clock();
    omp_set_num_threads(8);

#pragma omp parallel for schedule ( static)// reduction(+:result)
    for (int i = 0; i < n; i++)
    {
#pragma omp parallel for schedule ( static) reduction(+:tmp)
        for (int j = 0; j < n; j++)
        {
            tmp += a[i][j] * y[j];
        }

        result[i] = x[i] + tmp;
    }
```

II) Integral calculation

This programs runs the calculation of the following operation:

N defines the number of iterations of the integral calculation

The function fcalc that calculated integral

Results

With N = `10 000 000`

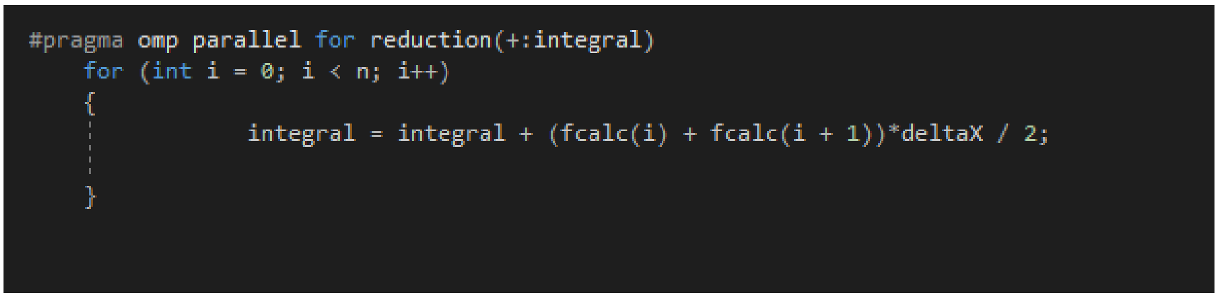
Without Multi-computing

Average treatment time: 2s

With Multi-computing

Average treatment time: 0.5s

For this exercise, it was easier to configure multi-computing. Using the word “reduction”, I was able to avoid calculation issues. We can notice a big difference with the use of multi-computing.



```
#pragma omp parallel for reduction(+:integral)
    for (int i = 0; i < n; i++)
    {
                integral = integral + (fcalc(i) + fcalc(i + 1))*deltaX / 2;

    }
```