Lab Assignment: OpenMP Work Sharing & Scheduling

Objective: To perform linear algebra operation to analyse the performance impact of scheduling clauses (static vs. dynamic) and the effect of memory access patterns (row-major vs. column-major) on parallel execution time.

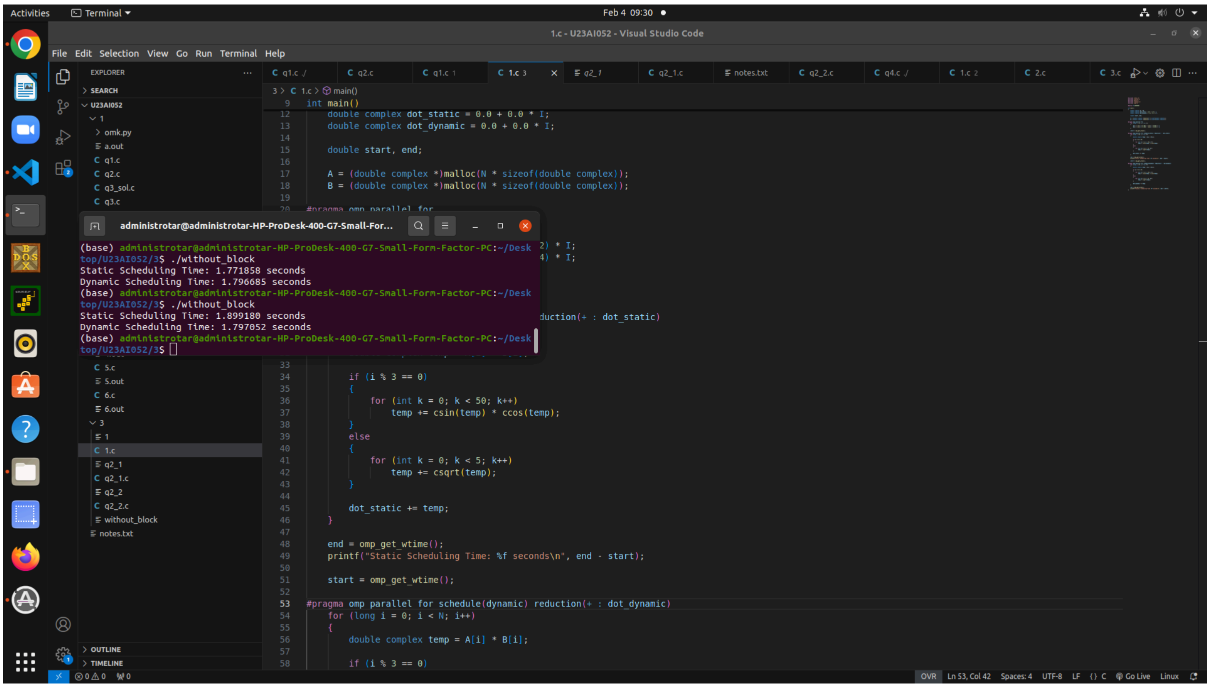
Part 1: Vector-Vector Operations

Task: Implement a program that performs a **Dot Product** of two large vectors. Compare the execution time using static and dynamic scheduling.

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <complex.h>

#include <omp.h>


#define N 20000000


int main()

{

   double complex *A, *B;

   double complex dot_static = 0.0 + 0.0 * I;

   double complex dot_dynamic = 0.0 + 0.0 * I;


   double start, end;


   A = (double complex *)malloc(N * sizeof(double complex));

   B = (double complex *)malloc(N * sizeof(double complex));


#pragma omp parallel for

   for (long i = 0; i < N; i++)

   {

       A[i] = cos(i * 0.001) + sin(i * 0.002) * I;
```

```c
        B[i] = sin(i * 0.003) + cos(i * 0.004) * I;
    }

    start = omp_get_wtime();

#pragma omp parallel for schedule(static, 100) reduction(+ :
dot_static)
    for (long i = 0; i < N; i++)
    {
        double complex temp = A[i] * B[i];

        if (i % 3 == 0)
        {
            for (int k = 0; k < 50; k++)
                temp += csin(temp) * ccos(temp);
        }
        else
        {
            for (int k = 0; k < 5; k++)
                temp += csqrt(temp);
        }

        dot_static += temp;
    }

    end = omp_get_wtime();
    printf("Static Scheduling Time: %f seconds\n", end - start);

    start = omp_get_wtime();
```

```
#pragma omp parallel for schedule(dynamic, 100) reduction(+ :
dot_dynamic)
    for (long i = 0; i < N; i++)
    {
        double complex temp = A[i] * B[i];

        if (i % 3 == 0)
        {
            for (int k = 0; k < 50; k++)
                temp += csin(temp) * ccos(temp);
        }
        else
        {
            for (int k = 0; k < 5; k++)
                temp += csqrt(temp);
        }

        dot_dynamic += temp;
    }

    end = omp_get_wtime();
    printf("Dynamic Scheduling Time: %f seconds\n", end - start);
}
```



Analysis: I have noticed that if we keep the same size of the block then static is faster as it has less overhead at runtime for checking blocking and all. If we don't give block size to static then it will be slower.

Part 2: Matrix-Vector Multiplication (Row vs. Column)

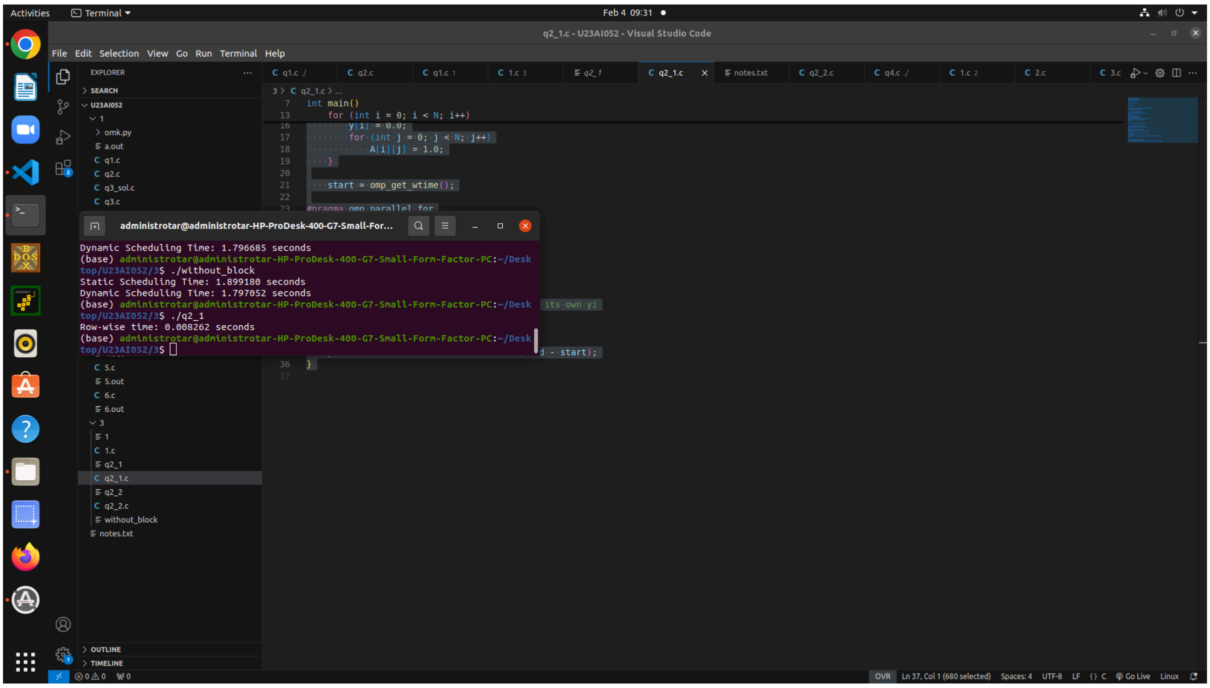
Task: Observe how **spatial locality** and **synchronization** affect performance. C uses row-major ordering, making row-wise distribution much faster.

Program A: Row-Wise Distribution

In this version, each thread calculates a complete element of the result vector. No two threads write to the same memory location.

```
#include <stdio.h>

#include <stdlib.h>

#include <omp.h>


#define N 4000


int main()

{

    static double A[N][N], x[N], y[N];

    double start, end;
```

```c
#pragma omp parallel for
   for (int i = 0; i < N; i++)
   {
       x[i] = 1.0;
       y[i] = 0.0;
       for (int j = 0; j < N; j++)
           A[i][j] = 1.0;
   }

   start = omp_get_wtime();

#pragma omp parallel for
   for (int i = 0; i < N; i++)
   {
       double sum = 0.0;
       for (int j = 0; j < N; j++)
       {
           sum += A[i][j] * x[j];
       }
       y[i] = sum; // each thread writes to its own yi
   }

   end = omp_get_wtime();
   printf("Row-wise time: %f seconds\n", end - start);
}
```

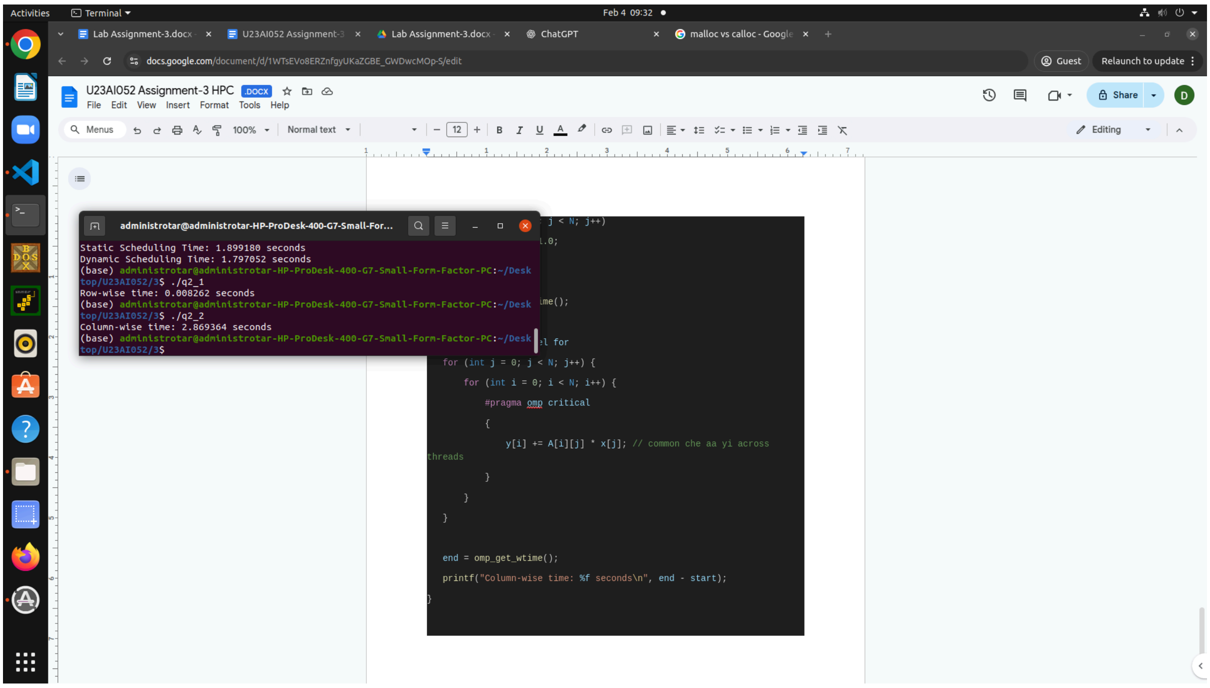


Program B: Column-Wise Distribution

In this version, threads work on columns. Because multiple threads will try to update the same y[i] at the same time, we must use #pragma omp critical to prevent race conditions.

```
#include <stdio.h>

#include <stdlib.h>

#include <omp.h>


#define N 4000


int main() {

   static double A[N][N], x[N], y[N];

   double start, end;


   #pragma omp parallel for

   for (int i = 0; i < N; i++) {

       x[i] = 1.0;

       y[i] = 0.0;
```

```
        for (int j = 0; j < N; j++)

            A[i][j] = 1.0;

    }


    start = omp_get_wtime();


    #pragma omp parallel for

    for (int j = 0; j < N; j++) {

        for (int i = 0; i < N; i++) {

            #pragma omp critical

            {

                y[i] += A[i][j] * x[j]; // common che aa yi across
threads

            }

        }

    }


    end = omp_get_wtime();

    printf("Column-wise time: %f seconds\n", end - start);

}
```



Analysis: I saw that there is great difference in execution time of row and column wise multiplication as cache-line size becomes irrelevant in this context of column wise multiplication. This doesn’t allow us to use spatial locality.