

Parallel and Distributed Lab Computing Digital Assessment III

Name: Atharva Manoj Dagaonkar

Registration No: 20BCE0891

Slot: L27 + L28

Faculty: Dr. K. Murugan

Question

Write a OpenMP program for Matrix multiply (specify run of a GPU card, large scale data Complexity of the problem need to be specified).

Aim

To perform matrix multiplication using OpenMP

GPU Processing

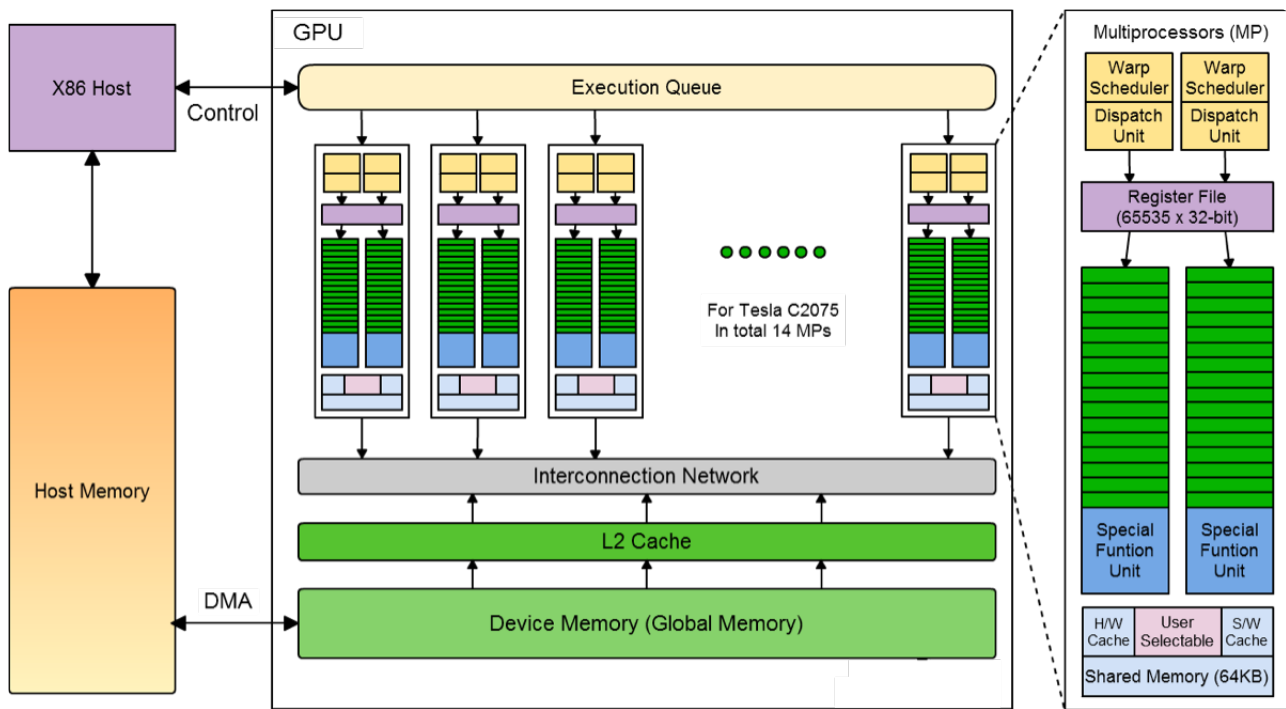
Since version 4.0 , OpenMP supports heterogeneous systems. OpenMP uses `TARGET` construct to offload execution from the host to the target device(s), and hence the directive name. In addition, the associated data needs to be transferred to the device(s) as well. Once transferred, the target device owns the data and accesses by the host *during the execution* of the target region is forbidden.

A host/device model is generally used by OpenMP for offloading:

- normally there is only one single host: e.g. CPU
- one or multiple target devices *of the same kind*: e.g. coprocessor, GPU, FPGA, ...
- unless with unified shared memory, the host and device have separate memory address space

The `TARGET` construct transfers the control flow to the device is sequential and synchronous, and it is because OpenMP separates offload and parallelism. One needs to explicitly create parallel regions on the target device to make efficient use of the device(s).

CPUs and GPUs were designed with different goals in mind. While the CPU is designed to excel at executing a sequence of operations, called a thread, as fast as possible and can execute a few tens of these threads in parallel, the GPU is designed to excel at executing many thousands of them in parallel. GPUs were initially developed for highly-parallel task of graphic processing and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control. More transistors dedicated to data processing is beneficial for highly parallel computations; the GPU can hide memory access latencies with computation, instead of relying on large data caches and complex flow control to avoid long memory access latencies, both of which are expensive in terms of transistors.



Procedure

1. Import the required header files.
2. Initialize the number of rows and columns as N.
3. N can be a large number for large scale processing.
4. Initialize the matrices with required values.
5. Perform parallelised matrix multiplication using pragma omp parallel for. The shared data structures are the matrices A,B,C. Threads will run parallelly.
6. Display the time elapsed by using timeofday function.
7. Display the result of matrix multiplication.

Complexity

The initialization for matrices is done in $O(n^2)$ time.

The multiplication of matrices is done in cubic time $O(n^3)$

Parallelizing these kind of processes is important to reduce time needed for computing large sized data.

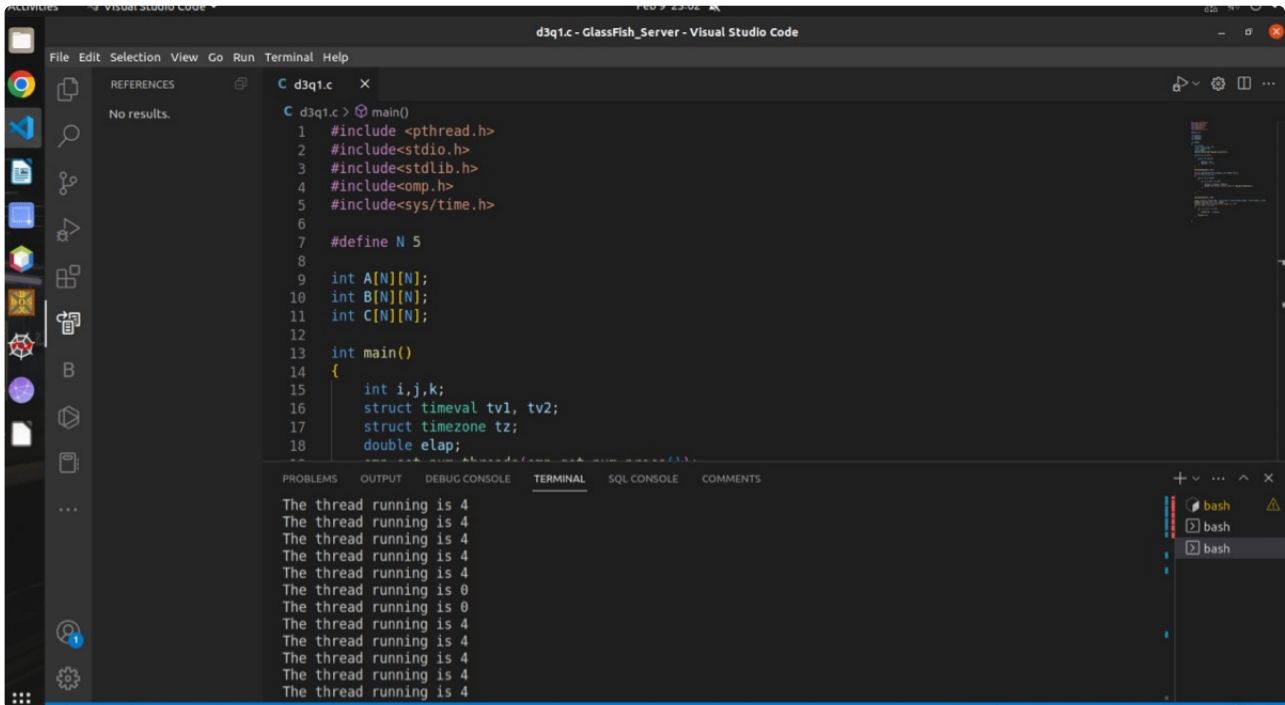
Code

```

#include <pthread.h>
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
#include<sys/time.h>
#define COUNT 5
int A[COUNT][COUNT];
int B[COUNT][COUNT];
int C[COUNT][COUNT];
int main()
{
    int i,j,k;
    struct timeval tv1, tv2;
    struct timezone tz;
    double elap;
    omp_set_num_threads(omp_get_num_procs());
    for(i = 0; i< COUNT;i++)
    {
        for(j = 0; j<COUNT;j++)
        {
            A[i][j] = i+j;
            B[i][j] = i - j;
        }
    }
    gettimeofday(&tv1, &tz);
    #pragma omp parallel for private(i,j,k) shared (A,B,C)
    for ( i = 0; i < COUNT; i++)
    {
        for(j = 0; j< COUNT;j++)
        {
            for ( k = 0; k < COUNT; k++)
            {
                C[i][j] += A[i][k] * B[k][j];
                printf("The thread running is %d \n", omp_get_thread_num());
            }
        }
    }
    gettimeofday(&tv2, &tz);
    elap = (double) (tv2.tv_sec - tv1.tv_sec) + (double)(tv2.tv_usec - tv1.tv_usec) *
    1.e-6;
    printf("\nElapsed time = %f", elap);
    printf("\nThe matrix multiplication output is: \n");
    for ( i = 0; i < COUNT; i++)
    {
        for ( j = 0; j < COUNT; j++)
        {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }
}

```

Output

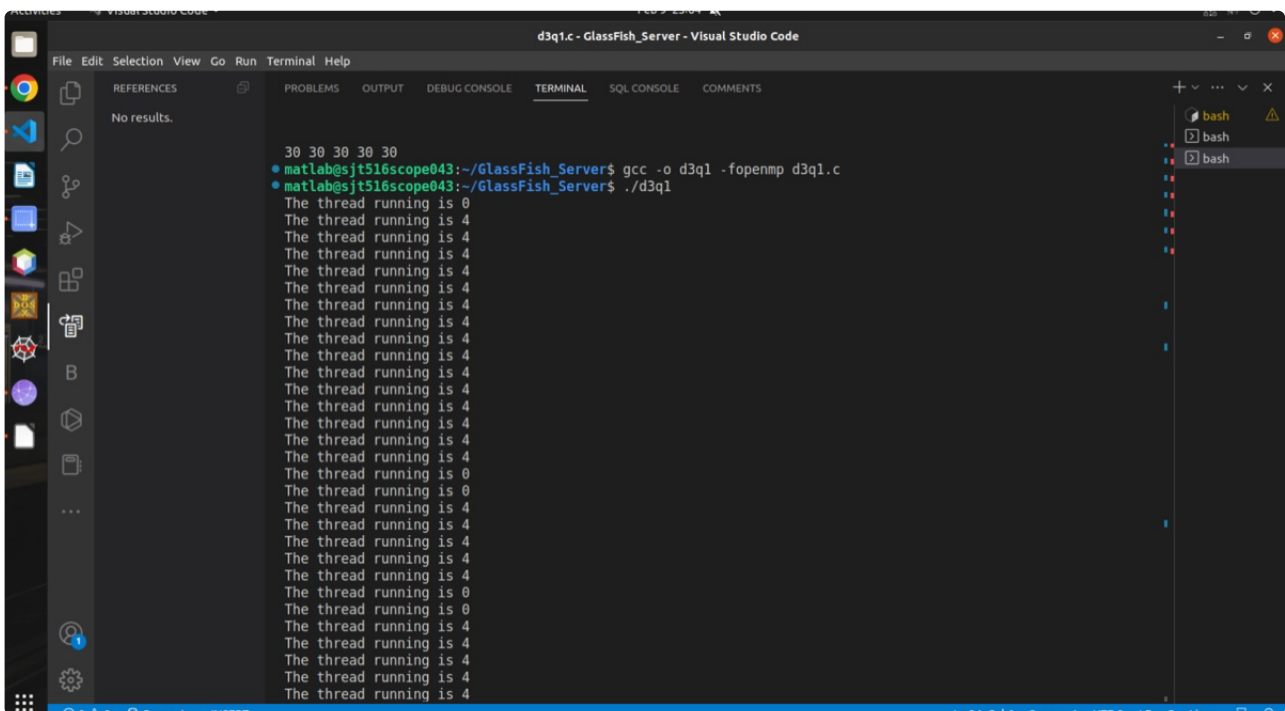


The screenshot shows the Visual Studio Code editor with the file `d3q1.c` open. The code is a C program that uses `pthread` to create multiple threads. The terminal output shows the program running and printing "The thread running is 4" for most threads, and "The thread running is 0" for some threads.

```
C d3q1.c > main()
1 #include <pthread.h>
2 #include<stdio.h>
3 #include<stdlib.h>
4 #include<omp.h>
5 #include<sys/time.h>
6
7 #define N 5
8
9 int A[N][N];
10 int B[N][N];
11 int C[N][N];
12
13 int main()
14 {
15     int i,j,k;
16     struct timeval tv1, tv2;
17     struct timezone tz;
18     double elap;
```

Terminal Output:

```
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 0
The thread running is 0
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
```



The screenshot shows the Visual Studio Code editor with the file `d3q1.c` open. The terminal output shows the program being compiled with `gcc -o d3q1 -fopenmp d3q1.c` and then executed with `./d3q1`. The output shows the program running and printing "The thread running is 4" for most threads, and "The thread running is 0" for some threads.

```
30 30 30 30 30
matlab@sjt516scope043:~/GlassFish_Server$ gcc -o d3q1 -fopenmp d3q1.c
matlab@sjt516scope043:~/GlassFish_Server$ ./d3q1
The thread running is 0
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 0
The thread running is 0
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 0
The thread running is 0
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 0
The thread running is 0
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
The thread running is 4
```


The image shows a screenshot of the Visual Studio Code interface. The Explorer pane on the left displays the project structure for 'GLASSFISH_SERVER', including directories like 'bin', 'glassfish', 'javadb', 'META-INF', 'mq', and files like 'd3q1', 'd3q1.c', and 'README.txt'. The Terminal pane on the right shows the output of a Java program. The output consists of 18 lines of 'The thread running is 6', followed by 'Elapsed time = 0.015855', and then a matrix multiplication result. The matrix is a 10x10 grid of integers. The status bar at the bottom indicates the active file is 'd3q1.c' and the current directory is '/GlassFish_Server\$'.