High Performance Computing Lab : 03

Implementation of schedule, nowait, reduction, ordered and collapse.

PRN : 2019BTECS00055

Name : Aditya Rajendra Manapure

Q1: Analyse and implement a Parallel code for the below program using OpenMP.

Parallel Code:

```
#include <iostream>
#include <omp.h>
#include <vector>
using namespace std;

void sort(vector<int> arr, int n)
{
    int i, j;
#pragma omp parallel for
    for (i = 0; i < n - 1; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                cout << "i=" << i << " j=" << j << " Thread No: " <<
omp_get_thread_num() << endl;
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
void sort_des(vector<int> arr, int n)
{
    int i, j;
```

```
#pragma omp parallel for
    for (i = 0; i < n; ++i)
    {
        for (j = i + 1; j < n; ++j)
        {
            cout << "i=" << i << " j=" << j << " Thread No: " <<
omp_get_thread_num() << endl;
            if (arr[i] < arr[j])
            {
                int a = arr[i];
                arr[i] = arr[j];
                arr[j] = a;
            }
        }
    }
}
int main()
{
    // fill the code;
    int n;
    scanf("%d", &n);
    vector<int> arr1(n), arr2(n);
    int i;
    for (i = 0; i < n; i++)
    {
        cin >> arr1[i];
    }
    for (i = 0; i < n; i++)
    {
        cin >> arr2[i];
    }
    sort(arr1, n);
    sort_des(arr2, n);
    int sum = 0;
    cout << endl;
    for (i = 0; i < n; i++)
        cout << arr1[i] << " ";
    cout << endl;
    for (i = 0; i < n; i++)
    {
        sum = sum + (arr1[i] * arr2[i]);
    }
    printf("%d", sum);
```

```
    return 0;
}
```

```
C:\Users\adity\OneDrive\Documents\College\sem 7\HPC\HPC Lab\Assignment-3>g++ -fopenmp Sorting_sum.cpp

C:\Users\adity\OneDrive\Documents\College\sem 7\HPC\HPC Lab\Assignment-3>a.exe
sum : 3574789
```

Q2: Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculate the execution time or use GPROF)
i. For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads.
ii. Explain whether or not the scaling behaviour is as expected.

Sequential Code:

```
#include <iostream>
#include <vector>
#include <omp.h>
#include <iomanip>
using namespace std;
#define M 250
#define N 250
int main()
{
    int **m1 = new int *[M];
    int **m2 = new int *[M];
    int **sum = new int *[M];

    for (int i = 0; i < M; i++)
    {
        m1[i] = new int[N];
        m2[i] = new int[N];
        sum[i] = new int[N];
    }

    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            m1[i][j] = 10;
            m2[i][j] = 20;
        }
```

```
    }

    int i, j;
    double stime = omp_get_wtime();
    // cout << setprecision(20) << stime << endl;
    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
        {
            sum[i][j] = m1[i][j] + m2[i][j];
        }
    }
    double etime = omp_get_wtime();
    double time = etime - stime;
    // cout << setprecision(20) << etime << endl;
    cout << "\nTime taken: " << setprecision(10) << time;
    return 0;
}
```

Execution:

Ts = 0.00099992752

Parallel Code:

```
#include <iostream>
#include <vector>
#include <omp.h>
#include <iomanip>
using namespace std;

#define M 250
#define N 250
int main()
{
    int **m1 = new int *[M];
    int **m2 = new int *[M];
    int **sum = new int *[M];

    for (int i = 0; i < M; i++)
    {
        m1[i] = new int[N];
        m2[i] = new int[N];
        sum[i] = new int[N];
```

```
    }

    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            m1[i][j] = 10;
            m2[i][j] = 20;
        }
    }

    int i, j, tid;
    omp_set_num_threads(4);
    double stime = omp_get_wtime();
    // cout << setprecision(20) << stime << endl;
#pragma omp parallel for collapse(2)
    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
        {
            sum[i][j] = m1[i][j] + m2[i][j];
            // cout << "Thread id: " << omp_get_thread_num() << " i:" << i
<< " j:" << j << "\n";
        }
    }

    double etime = omp_get_wtime();

    double time = etime - stime;

    cout << "\nTime taken: " << setprecision(20) << time;
    return 0;
}
```

Execution:

N	250	500	750	1000
Ts	0.0009999275208	0.001000165939	0.001999855042	0.003000020981
Tp	0.0009999275208	0.0009999275208	0.001000165939	0.00200009346
Speedup	1	1.00024	1.99	1.49

Scaling behaviour is not as expected as we should be getting constant speedup.

Q3. For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following:
i. Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.
ii. Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.
iii. Demonstrate the use of nowait clause.

Sequential Code:

```
#include <iostream>
#include <omp.h>
#include <iomanip>
using namespace std;

#define N 200

int main()
{
    int *v = new int[N];
    int *sum = new int[N];

    int scalar = 10;
    int i;
    for (i = 0; i < N; i++)
        v[i] = 1;

    double stime, etime;
    stime = omp_get_wtime();
    for (i = 0; i < N; i++)
    {
```

```
        sum[i] = v[i] + scalar;
        cout << sum[i] << " ";
    }
    etime = omp_get_wtime();

    // for (i = 0; i < N; i++)
    //     printf("%d ", v[i]);
    double time_taken = (etime - stime);
    cout << "\nTime Taken: " << setprecision(10) << time_taken;
    return 0;
}
```

Ts = 0.01799988747

Parallel:

```
#include <iostream>
#include <omp.h>
#include <iomanip>
using namespace std;

#define N 200

int main()
{
    int *v = new int[N];
    int *sum = new int[N];

    int scalar = 10;
    int i;
    for (i = 0; i < N; i++)
        v[i] = 1;

    double stime, etime;
    stime = omp_get_wtime();

#pragma omp parallel for schedule(static) //static or dynamic
    for (i = 0; i < N; i++)
    {
        sum[i] = v[i] + scalar;
        cout << sum[i] << " ";
    }
```

```
    etime = omp_get_wtime();

    // for (i = 0; i < N; i++)
    //     printf("%d ", v[i]);
    double time_taken = (etime - stime);
    cout << "\nTime Taken: " << setprecision(10) << time_taken;
    return 0;
}
```

STATIC:

Chunk size	4	8	12
Tp	0.01699995995	0.02199983597	0.01799988747

DYNAMIC:

Chunk size	4	8	12
Tp	0.01799988747	0.01700019836	0.01899981499