Date: 05/09/2022

High Performance Computing Lab

Assignment - 3

PRN: 2019BTECS00029
Name: Harshal Kodgire

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

Sequential Code :

```
// C Program to find the minimum scalar product of two vectors
// (dot product)
#include <stdio.h>
#include <time.h>
#define n 100000
int sort(int nums[]) {
    int i, j;
    for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - i - 1; j++)
            if (nums[j] > nums[j + 1]) {
                int temp = nums[j];
                nums[j] = nums[j + 1];
                nums[j + 1] = temp;
            }
}

int sortDesc(int nums[]) {
    int i, j;
    for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++) {
            if (nums[i] < nums[j]) {
```

```
            int a = nums[i];
            nums[i] = nums[j];
            nums[j] = a;
        }
    }
  }
}
int main() {
  int nums1[n], nums2[n];
  for (int i = 0; i < n; i++) {
    nums1[i] = 10;
  }
  for (int i = 0; i < n; i++) {
    nums2[i] = 20;
  }
  clock_t t = clock();
  sort(nums1);
  sortDesc(nums2);
  t = clock() - t;
  double time = ((double)t) / CLOCKS_PER_SEC;
  printf("Time taken (seq): %f\n", time);
  int sum = 0;
  for (int i = 0; i < n; i++) {
    sum = sum + (nums1[i] * nums2[i]);
  }
  printf("%d\n", sum);
  return 0;
}
```

Parallel Code :

```
// C Program to find the minimum scalar product of two
// vectors(dot product)
```

```
#include <omp.h>
#include <stdio.h>
#include <time.h>
#define n 100000
int sort(int nums[]) {
    int i, j;
    for (i = 0; i < n; i++) {
        int turn = i % 2;
#pragma omp parallel for
        for (j = turn; j < n - 1; j += 2)
            if (nums[j] > nums[j + 1]) {
                int temp = nums[j];
                nums[j] = nums[j + 1];
                nums[j + 1] = temp;
            }
    }
}
int sort_des(int nums[]) {
    int i, j;
    for (i = 0; i < n; ++i) {
        int turn = i % 2;
#pragma omp parallel for
        for (j = turn; j < n - 1; j += 2) {
            if (nums[j] < nums[j + 1]) {
                int temp = nums[j];
                nums[j] = nums[j + 1];
                nums[j + 1] = temp;
            }
        }
    }
}
int main() {
    int nums1[n], nums2[n];
```

```
    // int i;
    for (int i = 0; i < n; i++) {
        nums1[i] = 10;
    }
    for (int i = 0; i < n; i++) {
        nums2[i] = 20;
    }
    clock_t t;
    t = clock();
    sort(nums1);
    sort_des(nums2);
    t = clock() - t;

    double time_taken = ((double)t) / CLOCKS_PER_SEC;
    printf("Time taken (seq): %f\n", time_taken);
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum = sum + (nums1[i] * nums2[i]);
    }
    printf("%d\n", sum);
    return 0;
}
```

```
D:\WCE_ENGINEERING\BTECH_SEM1\HPC Lab>g++ -fopenmp Se

D:\WCE_ENGINEERING\BTECH_SEM1\HPC Lab>a.exe
Time taken (seq): 16.613000
20000000

D:\WCE_ENGINEERING\BTECH_SEM1\HPC Lab>g++ -fopenmp Par

D:\WCE_ENGINEERING\BTECH_SEM1\HPC Lab>a.exe
Time taken (seq): 8.613000
20000000
```

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 calculating 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 behavior is as expected.

Sequential :

```c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 1000
void add(int **a, int **b, int **c) {
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         c[i][j] = a[i][j] + b[i][j];
      }
   }
}
```

```c
void input(int **a, int num) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            a[i][j] = num;
        }
    }
}
void display(int **a) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}
int main() {
    int **a = (int **)malloc(sizeof(int *) * N);
    int **b = (int **)malloc(sizeof(int *) * N);
    int **c = (int **)malloc(sizeof(int *) * N);
    for (int i = 0; i < N; i++) {
        a[i] = (int *)malloc(sizeof(int) * N);
        b[i] = (int *)malloc(sizeof(int) * N);
        c[i] = (int *)malloc(sizeof(int) * N);
    }
    input(a, 1);
    input(b, 1);
    double start = omp_get_wtime();
    add(a, b, c);
    double end = omp_get_wtime();
    // display(c);
    printf("Time taken (seq): %f\n", end - start);
}
```

N	250	500	750	1000	2000
Time	0.001	0.0015	0.00251	0.0031	0.0150

Parallel：

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 250
void add(int **a, int **b, int **c) {
#pragma omp parallel for
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
}
void input(int **a, int num) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            a[i][j] = num;
        }
    }
}
void displayMatrix(int **a) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", a[i][j]);
        }
        printf("\n");
```

```
    }
}

int main() {
    int **a = (int **)malloc(sizeof(int *) * N);
    int **b = (int **)malloc(sizeof(int *) * N);
    int **c = (int **)malloc(sizeof(int *) * N);
    for (int i = 0; i < N; i++) {
        a[i] = (int *)malloc(sizeof(int) * N);
        b[i] = (int *)malloc(sizeof(int) * N);
        c[i] = (int *)malloc(sizeof(int) * N);
    }
    input(a, 1);
    input(b, 1);
    double start = omp_get_wtime();
    add(a, b, c);
    double end = omp_get_wtime();
    // display(c);
    printf("Time taken (seq): %f\n", end - start);
}
```

N	250	500	750	1000	2000
Time	0.002	0.001	0.00254	0.0029	0.0034

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

Static Schedule:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 200
int main() {
    int *a = (int *)malloc(sizeof(int) * N);
    int *c = (int *)malloc(sizeof(int) * N);
    int b = 10;
    omp_set_num_threads(6);
    for (int i = 0; i < N; i++) {
        a[i] = 0;
    }
    double itime, ftime, exec_time;
    itime = omp_get_wtime();
#pragma omp parallel for schedule(static, 2)
    for (int i = 0; i < N; i++) {
        c[i] = a[i] + b;
    }
    ftime = omp_get_wtime();
    exec_time = ftime - itime;
    printf("\n\nTime taken is %f\n", exec_time);
}
```

Chunk Size	2	4	6	8
Time	.0019	.0010	0.0009	0.0009

Dynamic Schedule:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#define N 200
int main() {
   int *a = (int *)malloc(sizeof(int) * N);
   int *c = (int *)malloc(sizeof(int) * N);
   int b = 10;
   omp_set_num_threads(6);
   for (int i = 0; i < N; i++) {
      a[i] = 0;
   }
   double itime, ftime, exec_time;
   itime = omp_get_wtime();
#pragma omp parallel for schedule(dynamic, 2)
   for (int i = 0; i < N; i++) {
      c[i] = a[i] + b;
   }
   ftime = omp_get_wtime();
   exec_time = ftime - itime;
   printf("\n\nTime taken is %f\n", exec_time);
}
```

Chunk Size	2	4	6	8
Time	0.00187	0.0017	0.0015	0.0012

NoWait Claue:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 10
```

```
void hello_world() {
    printf("Hello world\n");
}
void print(int i) {
    printf("Value %d\n", i);
}
int main() {
#pragma omp parallel
    {
#pragma omp for nowait
        for (int i = 0; i < N; i++) {
            print(i);
        }
        hello_world();
    }
}
```

Output without nowait:

```
Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 3
$ g++ -fopenmp Q3_NoWait.cpp

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 3
$ ./a.exe
Value 2
Value 3
Value 7
Value 0
Value 1
Value 9
Value 6
Value 5
Value 8
Value 4
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
```

Output with NoWait:

```
Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 3
$ g++ Q3_NoWait.cpp

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 3
$ ./a.exe
Value 0
Value 1
Value 2
Value 3
Hello world
Value 5
Hello world
Value 8
Hello world
Hello world
Value 4
Hello world
Value 7
Hello world
```