Abubakkar Abdullah

20p-0045

PDC

BCS-7F

-------------------

TASK 1:

List of run-time routines in OpenMP:

omp_get_thread_num(): Returns the thread number of the calling thread.
omp_get_num_threads(): Returns the number of threads in the current team.
omp_set_num_threads(): Sets the number of threads in the current team.
omp_get_max_threads(): Returns the maximum number of threads that can be used by a parallel region.
omp_get_num_procs(): Returns the number of processors available to the program.
omp_get_wtime(): Returns the wall-clock time in seconds.
omp_get_wtick(): Returns the precision of the timer.

Convert Parallel Code To Serial Code:

Parallel:

```
#include <stdio.h>
#include <omp.h>
int main() {
#pragma omp parallel
{ int thread_id = omp_get_thread_num();
printf("Hello, World! This is thread %d\n", thread_id);
}
return 0;
}
```

Serial:

```
#include <stdio.h>
int main() {
    int thread_id = 0; // Thread ID in the serial version
    printf("Hello, World! This is thread %d\n",
thread_id);
    return 0;
}
```

TASK 2:

1. The following code adds two arrays of size 16 together and stores answer in result array.

```
#include <stdio.h>
int main() {
int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, ..., 16};
int array2[16] = {16, ..., 13, 12, 11, 10, 9, 8, 7, 6, 5,
4, 3, 2, 1};
int result[16];
for (int i = 0; i < 16; i++) {
result[i] = array1[i] + array2[i];
}
for (int i = 0; i < 16; i++) {
printf("%d ", result[i]);
}
printf("\n");

return 0;
}
```

2. Convert it into Parallel, such that only the addition part is parallelized.

3. The display loop at the end displays the result. Modify the code such that this is also parallel, but only thread of id 0 is able to display the entire loop. The others should not do anything. When making it parallel, make sure its the old threads and new threads are not created. What output do you see?

ANS:
PARALLELIZING AND DISPLAYING BY THREAD “0”

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

int main() {
    int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16};
    int array2[16] = {16, 15, 14, 13, 12, 11, 10, 9, 8,
7, 6, 5, 4, 3, 2, 1};
    int result[16];

    #pragma omp parallel for
    for (int i = 0; i < 16; i++) {
        result[i] = array1[i] + array2[i];
    }

    // Display loop (only executed by thread 0)
    #pragma omp parallel
    {
        #pragma omp master
        {
            for (int i = 0; i < 16; i++) {
                printf("%d ", result[i]);
            }
            printf("\n");
        }
    }

    return 0;
}
```

TASK 3:

1. Modify the code of Task 2 and do the job in half of the threads.

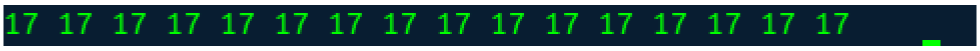
ANS:

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

int main() {
    int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16};
    int array2[16] = {16, 15, 14, 13, 12, 11, 10, 9, 8,
7, 6, 5, 4, 3, 2, 1};
    int result[16];

    // Set the number of threads to half of the available
threads
    int num_threads = omp_get_max_threads() / 2;
    omp_set_num_threads(num_threads);

    #pragma omp parallel for
    for (int i = 0; i < 16; i++) {
        result[i] = array1[i] + array2[i];
    }

    // Display loop (only executed by thread 0)
    #pragma omp parallel
    {
        #pragma omp master
        {
            for (int i = 0; i < 16; i++) {
                printf("%d ", result[i]);
            }
            printf("\n");
        }
    }

    return 0;

}
```

OUTPUT:

.
.
.
.
.
..

```
./a.out
```



```
17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
```

.
.
.

TASK 4:

To parallelize the given code using OpenMP with 16 threads, you can use the **#pragma omp parallel for**,

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

int main() {
    int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16};
    int array2[16] = {16, 15, 14, 13, 12, 11, 10, 9, 8,
7, 6, 5, 4, 3, 2, 1};
    int result1 = 0, result2 = 0;

    #pragma omp parallel for num_threads(16)
reduction(+:result1)
    for (int i = 0; i < 16; i++) {
        result1 += array1[i];
    }

    if (result1 > 10) {
        #pragma omp parallel for num_threads(16)
reduction(+:result2)
        for (int i = 0; i < 16; i++) {
            result2 += array2[i];
        }
    }

    printf("%d\n", result2);
    return 0;
}
```

If you want to remove reduction clause then add #PRAGMA ATOMIC instead of it,

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

int main() {
    int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16};
    int array2[16] = {16, 15, 14, 13, 12, 11, 10, 9, 8,
7, 6, 5, 4, 3, 2, 1};
    int result1 = 0, result2 = 0;

    #pragma omp parallel for num_threads(16)
    for (int i = 0; i < 16; i++) {
        #pragma omp atomic
        result1 += array1[i];
    }

    if (result1 > 10) {
        #pragma omp parallel for num_threads(16)
        for (int i = 0; i < 16; i++) {
            #pragma omp atomic
            result2 += array2[i];
        }
    }

    printf("%d\n", result2);
    return 0;
}
```

TASK 5:

Codes generated are given below:

```
#include <stdio.h>

void function1() {
    printf("Inside Function 1\n");
}

void function2() {
    printf("Inside Function 2\n");
}

void function3() {
    printf("Inside Function 3\n");
}

int main() {
    for (int i = 0; i < 1000000; i++) {
        function1();
        function2();
        function3();
    }

    return 0;
}
```

For Compilation, this below command is used:

```
gcc -pg -o gprof_test gprof_test.c
./gprof_test
gprof ./gprof_test > gprof_output.txt
```
