

Homework 2: OpenMP Usage and Code Optimization

Nombre: Ing. Andrés Gómez Jiménez
Nombre: Ing. Randy Céspedes Deliyore

Carne: 200935203
Carne: 201054417

1 OpenMP Optimizations

A branch named *Homework_2* in the repository named *grupo1-HPEC-2020* that could be find here: https://github.com/rscd27p/group1_HPCE_2020/tree/Homework_2.

The purpose of this homework is to evaluate different OpenMP optimizations for the calculation of pi in figure 1.

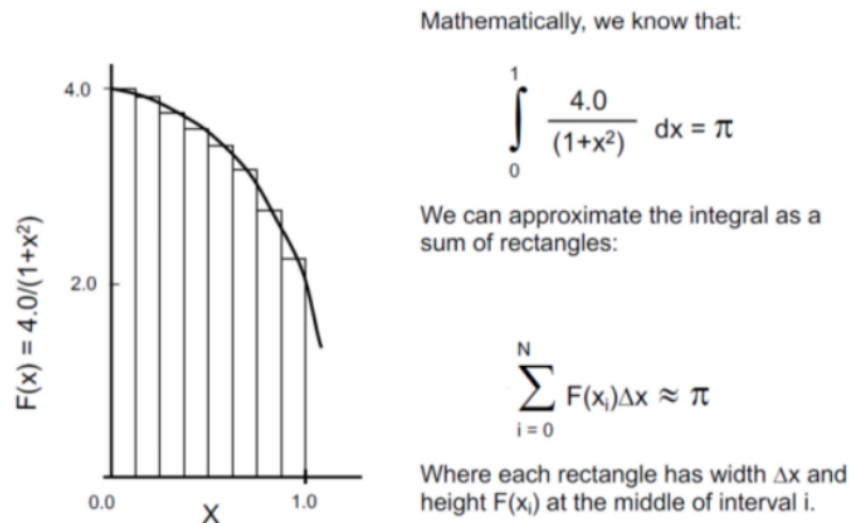


Figure 1: Pi Computational Mathematical Restriction

The code below was provided to calculate the value of pi.

```
// Pi Function declaration
double pi(uint32_t num_steps){
    // Declare Internal Variables
    uint32_t i;
    double x, pi, sum = 0.0;
    double start_time, run_time;
    // Calculate step size
    step = 1.0 / (double)num_steps;
    // get start time
    start_time = omp_get_wtime();
    // main for loop calculation
    for (i = 1; i <= num_steps; i++) {
        x = (i - 0.5) * step;
        sum = sum + 4.0 / (1.0 + x * x);
    }
    // obtain pi value
    pi = step * sum;
}
```

```
run_time = omp_get_wtime() - start_time;
// print results
printf("pi with %d steps is %lf in %lf seconds\n", num_steps, pi, run_time);
return pi;
}
```

Each one of the functions will be ran for 100 million times and they were tested in a desktop PC with the specifications on table 1.

Table 1: Computer Specifications for testing OpenMP optimizations

Computer Specifications	
Resource	Value
CPU	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
Memoria	16 GB DDR3 - 1650 MHz

The results of running the compiled code are show in figure 2.

```
taylorcespedes@taylorcespedes-desktop: ~/Documents/HPEC/...
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework
_2/OpenMP/src$ ./pi
pi with 100000000 steps is 3.141593 in 0.376120 seconds
```

Figure 2: Results of Running PI computation code without optimizations

1.1 Parallel Construct with Private and Reduction Clauses

The first optimization utilized was using the *parallel* construc, which based on the OpenMP manual [1] allows the system to create a team of threads to execute the code. Initially, is used with the loop construct using the command *#pragma omp for* with the following clauses:

- **Private:** for the x variable. Based on **web:openmp'clauses** the private clauses cause the variable or variable list to be private between threads.
- **Reduction:** for the sum variable. Based on **web:openmp'clauses** it causes the variable or list of variables to be private between threads and causes the variable to be subject of a reduction optimization at the end of the parallel region.

The code below was modified using the clauses above and saved in the file named *open_omp_private.c*.

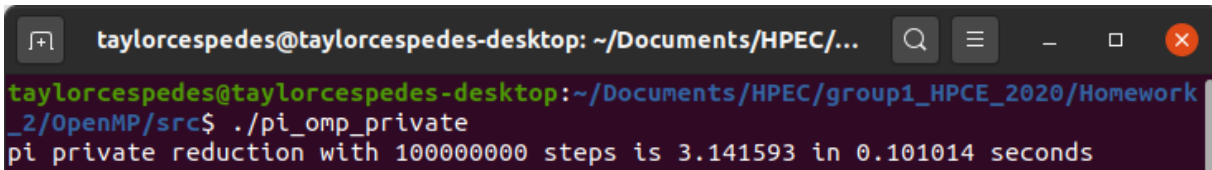
```
double pi_omp_private(uint32_t num_steps){
// Declare Internal Variables
uint32_t i;
double x, pi, sum = 0.0;
double start_time, run_time;
// Calcualte step size
step = 1.0 / (double)num_steps;
// get start time
start_time = omp_get_wtime();
// Make for loop parallel with the parallel construct
#pragma omp parallel
{
// Set reduction pragma to optimize sum operation and make the x variable private between
threads
// the reduction operation makes the variable private and causes the system to perform a
reduction optimization at the end of the parallel region
#pragma omp for reduction(+:sum) private (x)
for (i = 1; i <= num_steps; i++) {
x = (i - 0.5) * step;
}
```

```

        sum = sum + 4.0 / (1.0 + x * x);
    }
}
// Calculates Pi
pi = step * sum;
run_time = omp_get_wtime() - start_time;
// Prints Results
printf("pi private reduction with %d steps is %lf in %lf seconds\n", num_steps, pi, run_time);
return pi;
}

```

The results of can be seen in figure 3.



```

taylorcespedes@taylorcespedes-desktop: ~/Documents/HPEC/...
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework
_2/OpenMP/src$ ./pi_omp_private
pi private reduction with 100000000 steps is 3.141593 in 0.101014 seconds

```

Figure 3: Results of Running PI computation code with the parallel OpenMP construct, and the private and reduction clauses.

1.2 Teams OpenMP Construct

The OpenMP *Teams* construct is use to create a group or *team* of threads that will be used to execute a job in parallel [2]. This construct is being used in combination with the following constructs:

- **Distribute:** the distribute construct is used to evenly distribute the execution among the group of threads within each team.
- **Parallel:** this constricture is used with the same parameters than the ones used in the *open_omp_private.c* file on the previous section.

For this particular execution the following clauses part of the *Teams* construct are being used:

- **num_teams:** this clause is used to specify an used configurable number of teams.
- **thread_limit:** this constricture is used with the same parameters than the ones used in the *open_omp_private.c* file on the previous section.

There is a graphical representation of the code behaviour can be seen in figure 4. Each one of the *blue* lines in the figure represents a thread within each one of the teams, represeted by a square.

The code below was modified using the clauses above and saved in the file named *open_omp_teams.c*.

```

double pi_opm_teams(uint32_t num_steps, uint32_t teams_number, uint32_t max_num_threads){
    // Declare Internal Variables
    uint32_t i;
    double x, pi, sum = 0.0;
    double start_time, run_time;
    // Calcualte step size
    step = 1.0 / (double)num_steps;
    // get start time
    start_time = omp_get_wtime();
    // Make for loop parallel with the teams construct and a thread_limit
    #pragma omp teams distribute num_teams(teams_number) thread_limit(max_num_threads) reduction(+:
    sum) private (x)
    // Set reduction pragma to optimize sum operation and make the x variable private between threads
    // the reduction operation makes the variable private and causes the system to perform a
    reduction optimization at the end of the parallel region
    for (i = 1; i <= num_steps; i++) {

```

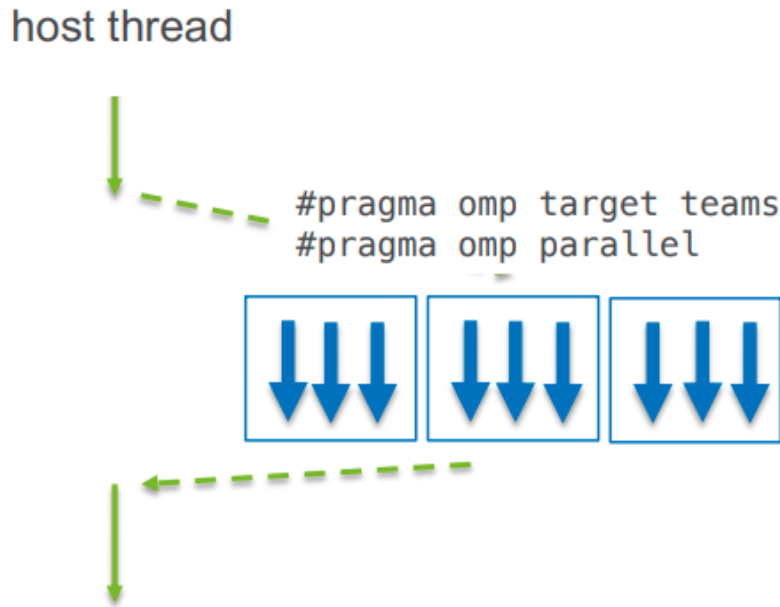


Figure 4: Running Teams and Parallel Constructs together. [2]

```

    x = (i - 0.5) * step;
    sum = sum + 4.0 / (1.0 + x * x);
}
pi = step * sum;
run_time = omp_get_wtime() - start_time;
printf("pi teams implementation with %d number of teams, %d steps is %lf in %.12lf seconds\n",
    teams_number, num_steps, pi, run_time);
return pi;
}

```

The results of running the code can be seen in figure 5.

```

taylorcespedes@taylorcespedes-desktop: ~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src$ ./pi_omp_teams -h
usage: ./pi_omp_teams [-h] [-a] -t teams_number -n thread_number
-t Maximum number of teams.
-n Maximum number of threads.
-a displays the information of the authors of the program.
-h displays the usage message to let the user know how to execute the application.
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src$ ./pi_omp_teams -t 8 -n 8
Configured to use -Teams number: 8 | -Max thread number: 8
Current settings -Teams used number: 8 | -Threads used: 1
pi teams implementation with 100000000 steps is 3.141593 in 0.099364836002 seconds
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src$

```

Figure 5: Running Teams and Parallel Constructs together

1.3 OpenMP Multiple Threads

The final type of optimization is designed using the command *num.threads* of the OpenMP library within the parallel construct. Also the same type of clauses from the *pi_omp_private* code are being used. This code can be seen in the section below:

```

double pi_opm_threads(uint32_t num_steps, uint32_t requested_threads){
    // Define internal Variables
    double x, pi, sum = 0.0;
    double start_time, run_time;
    step = 1.0 / (double)num_steps;
}

```

```

start_time = omp_get_wtime();
// Open MP Parallel pragma with num_threads
#pragma omp parallel num_threads(requested_threads)
{
    // Combine multiple threads with reduction for sum and use x variable private between
    threads
    #pragma omp for reduction(+:sum) private (x)
    for (uint32_t i = 1; i <= num_steps; i++){
        x = (i - 0.5) * step;
        sum = sum + 4.0 / (1.0 + x * x);
    }
}
// Calculate Pi Value
pi = step * sum;
run_time = omp_get_wtime() - start_time;
// Print Results
printf("pi teams implementation with %d steps is %lf in %.12lf seconds using %d threads\n",
num_steps, pi, run_time, requested_threads);
return pi;
}

```

The results of running the code can be seen in figure 6.

```

taylorcespedes@taylorcespedes-desktop: ~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src$ ./pi_omp_threads

*** Starting Pi calculation with 1 threads ***
pi teams implementation with 10000000 steps is 3.141593 in 0.367770012002 seconds using 1 threads

*** Starting Pi calculation with 2 threads ***
pi teams implementation with 10000000 steps is 3.141593 in 0.182626730995 seconds using 2 threads

*** Starting Pi calculation with 3 threads ***
pi teams implementation with 10000000 steps is 3.141593 in 0.124738178012 seconds using 3 threads

*** Starting Pi calculation with 4 threads ***
pi teams implementation with 10000000 steps is 3.141593 in 0.095181590994 seconds using 4 threads

*** Starting Pi calculation with 5 threads ***
pi teams implementation with 10000000 steps is 3.141593 in 0.123523715010 seconds using 5 threads

*** Starting Pi calculation with 6 threads ***
pi teams implementation with 10000000 steps is 3.141593 in 0.119694180990 seconds using 6 threads

*** Starting Pi calculation with 7 threads ***
pi teams implementation with 10000000 steps is 3.141593 in 0.107883832010 seconds using 7 threads

*** Starting Pi calculation with 8 threads ***
pi teams implementation with 10000000 steps is 3.141593 in 0.095497247996 seconds using 8 threads
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src$

```

Figure 6: Running code with parallel construct with num_threads clause

1.4 Discussion

In figure 7 the comparison of using all the optimizations can be observed.

The default program without optimization runs in approximately 381 ms. The initial optimization called private provided a reduction to 107ms with a speedup of 3.56. In this case the number of threads to use was not configured and let the library configure this automatically.

When using 8 teams with a maximum number of threads of 8 as well, the execution time went down to a similar value of 103 ms, with a speedup of 3.70. Based on [2] the main difference between the *Teams* and the *Parallel* constructs, is the fact that there is no boundary at the end of the execution of independent teams running groups

of threads. This means that each team can finish its execution independently and they the groups of teams merge after each execution is completed.

When manually controlling the number of threads, the best result of 96 ms is obtained when selecting 4 threads, with a speedup of 3.97.

```
taylorcespedes@taylorcespedes-desktop: ~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src$ ./pi
pi with 100000000 steps is 3.141593 in 0.381383 seconds
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src$ ./pi_omp_private
pi private reduction with 100000000 steps is 3.141593 in 0.107131 seconds
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src$ ./pi_omp_teams -t 8 -n 8
Configured to use -Teams number: 8 | -Max thread number: 8
Current settings -Teams used number: 8 | -Threads used: 1
pi teams implementation with 100000000 steps is 3.141593 in 0.102716427995 seconds
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src$ ./pi_omp_threads

*** Starting Pi calculation with 1 threads ***
pi implementation with parallel clause 100000000 steps is 3.141593 in 0.382931404980 seconds using 1 threads

*** Starting Pi calculation with 2 threads ***
pi implementation with parallel clause 100000000 steps is 3.141593 in 0.191038539022 seconds using 2 threads

*** Starting Pi calculation with 3 threads ***
pi implementation with parallel clause 100000000 steps is 3.141593 in 0.128298096999 seconds using 3 threads

*** Starting Pi calculation with 4 threads ***
pi implementation with parallel clause 100000000 steps is 3.141593 in 0.096358170995 seconds using 4 threads

*** Starting Pi calculation with 5 threads ***
pi implementation with parallel clause 100000000 steps is 3.141593 in 0.123551131983 seconds using 5 threads

*** Starting Pi calculation with 6 threads ***
pi implementation with parallel clause 100000000 steps is 3.141593 in 0.126180488005 seconds using 6 threads

*** Starting Pi calculation with 7 threads ***
pi implementation with parallel clause 100000000 steps is 3.141593 in 0.108766049991 seconds using 7 threads

*** Starting Pi calculation with 8 threads ***
pi implementation with parallel clause 100000000 steps is 3.141593 in 0.103514465009 seconds using 8 threads
taylorcespedes@taylorcespedes-desktop:~/Documents/HPEC/group1_HPCE_2020/Homework_2/OpenMP/src$
```

Figure 7: Comparison of running all codes

References

- [1] OpenMP Architecture Review Board, *OpenMP Application Program Interface Manual*, 2013. [Online]. Available: <https://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf>.
- [2] C. Bertoni, *Openmp 4.5 device offloading details = argonne national laboratory*, Online; accessed 22-August-2020, 2020. [Online]. Available: https://www.alcf.anl.gov/sites/default/files/2020-01/OpenMP45_Bertoni.pdf.