TEC | Tecnológico de Costa Rica

High Performance Embedded Systems

Homework 2
OpenMP

Lecturer:
MSc. Mauricio Caamaño Bedoya

2019

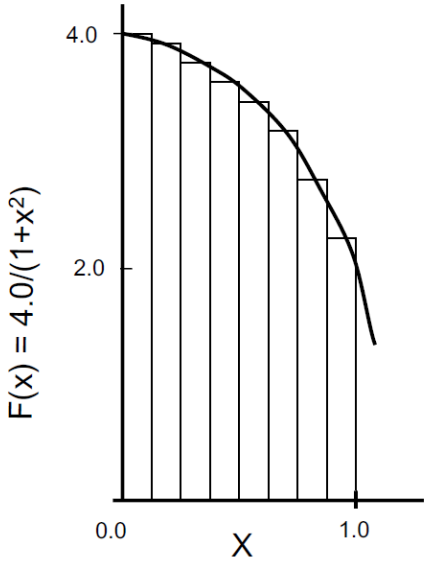
Motivation

OpenMP is an API that allows programmers to exploit data- and task-level parallelism on multicore platforms with shared memory. One of the key features of OpenMP is that programs can be parallelized incrementally from an initial sequential version. Once developers identify the regions of code that can benefit from parallel execution (e.g., hotspots), the code can be annotated with OpenMP directives to define how computations can be distributed across different cores and what data are accessed within each core.

Objective

With this assignment, the student will experiment with OpenMP directives to take advantage of multicore capabilities with data- and task-level parallelism.

Description



Mathematically, we know that:

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

We can approximate the integral as a sum of rectangles:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i.

The provided serial PI program (pi.c) approximates a definitive integral the result of which should equal pi.

```
static long num_steps = 100000;
double step;
int main ()
{     int i; double x, pi, sum = 0.0;

      step = 1.0/(double) num_steps;

      for (i=0; i< num_steps; i++){
            x = (i+0.5)*step;
            sum = sum + 4.0/(1.0+x*x);
      }
      pi = step * sum;
}
```

Using OpenMP, it is possible to achieve parallelization defining a parallel region:

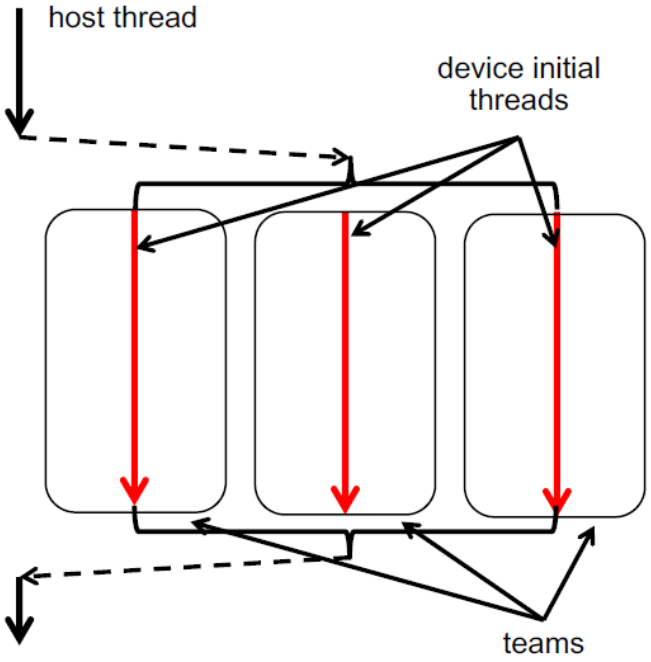
```
#include <omp.h>
static long num_steps = 100000; double step;
void main ()
{     int i; double x, pi, sum = 0.0;

      step = 1.0/(double) num_steps;
      #pragma omp parallel
      {
            #pragma omp for reduction(+:sum)
            #pragma omp for private(x)
            for (i=0; i< num_steps; i++){
                  x = (i+0.5)*step;
                  sum = sum + 4.0/(1.0+x*x);
            }
      }
      pi = step * sum;
}
```

Activities

1. Modify the provided pi.c to a new program **pi_omp_private.c** to add parallelism as shown in the second code above, using the *reduction* and *private* constructs. Add comments into your code explaining each of the *omp* constructs used:
   a. `#pragma omp for reduction(+:sum)`
   b. `#pragma omp for private(x)`

2. Modify the code to a new program **pi_omp_teams.c** using the *teams* and *distribute* constructs to add worksharing execution as shown in the next figure:



   Add comments into your code explaining each of the *omp* constructs used:

   a. `#pragma omp teams`
   b. `#pragma omp distribute`

3. Modify the code to a new program **pi_omp_thread.c** to run four times sequentially, each time using one more thread than the previous, up to 4 threads. The execution should look like the pi_loop binary provided:

```
$ ./pi_loop
num_threads = 1
pi is 3.141593 in 2.108124 seconds and 1 threads
num_threads = 2
pi is 3.141593 in 1.060070 seconds and 2 threads
num_threads = 3
pi is 3.141593 in 0.709118 seconds and 3 threads
num_threads = 4
pi is 3.141593 in 0.561934 seconds and 4 threads
```

Hints:

- Use the functions `omp_set_num_threads(n)` and `omp_get_num_threads()` to set and obtain the number of threads, respectively.
- You may use *single* and *reduction* constructs, for example:

```
…
#pragma omp parallel
{
      #pragma omp single
      printf(" num_threads = %d", omp_get_num_threads());

      #pragma omp for reduction(+ : sum)
      for (i = 1; i <= num_steps; i++) {
      …
```

Deliverables

1. Include modified codes with the functionality requested:
   - **pi_omp_private.c**
   - **pi_omp_teams.c**
   - **pi_omp_threads.c**
2. Modify makefile included in order to include all code to compile with a single ‘make’.

Evaluation criteria

Item	Description	Percentage
pi_omp_private.c	Functionality of pi_omp_private	20%
	Comments in pi_omp_private explaining OpenMP constructs	10%
pi_omp_teams.c	Functionality of pi_omp_teams	20%
	Comments in pi_omp_teams explaining OpenMP constructs	10%
pi_omp_threads.c	Functionality of pi_omp_threads	30%
Makefile	Modified to compile all codes with ‘make’	10%
TOTAL		**100%**

Delivery date

Monday, August 19th 11:59 p.m.