# Actividad de programación Multiprocesadores:
# Resolviendo problemas con OpenMP

José Ramón Romero Chávez (A01700318)
Tecnológico de Monterrey, Campus Querétaro
*A01700318@itesm.mx*

15 de octubre del 2018

## Resumen

Este documento se escribirá de manera puntual la implementación de los algoritmos:

- Montecarlo: para la generación del valor de Pi a través de la generación de valores aleatorios.
- Count Sort: algoritmo de ordenación por conteo

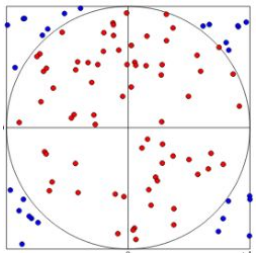
Todo esto haciendo uso de la librería OpenMP para el lenguaje de programación C, la cual nos permite sacar ventaja de sistemas multiprocesador, adicional a esto se realizará el mismo procedimiento en modo monoprocesador con el fin de analizar las diferencias y comparar su rendimiento.

## 1. Introducción

### MonteCarlo

$\pi$ (pi) es la relación entre la longitud de una circunferencia y su diámetro en geometría euclidiana. Es un número irracional y una de las constantes matemáticas más importantes. Se emplea frecuentemente en matemáticas, física e ingeniería.

El algoritmo MonteCarlo, está basado en la idea de la distribución estándar que se puede generar al lanzar de manera aleatoria “dardos” dentro de un círculo de radio 1.



```
number_in_circle = 0;
for (toss = 0; toss < number_of_tosses; toss++) {
    x = random double between -1 and 1;
    y = random double between -1 and 1;
    distance_squared = x*x + y*y;
    if (distance_squared <= 1) number_in_circle++;
}
pi_estimate = 4 * number_in_circle/((double) number_of_tosses);
```

**Count Sort**

La idea básica del algoritmo es que por cada elemento dentro de un arreglo se debe contar la cantidad de elementos que sean menor que el mismo, tomando en cuenta ese número se debe de insertar en un arreglo en la posición contada, es necesario validar escenarios donde existan elementos repetidos.


```
number_in_circle = 0;
for (toss = 0; toss < number_of_tosses; toss++) {
    x = random double between -1 and 1;
    y = random double between -1 and 1;
    distance_squared = x*x + y*y;
    if (distance_squared <= 1) number_in_circle++;
}
pi_estimate = 4 * number_in_circle/((double) number_of_tosses);
```


OpenMP es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen el comportamiento en tiempo de ejecución.

# 2. Implementación

La implementación fue realizada en una computadora con las siguientes características:

- MacBook Pro (Retina, 13-inch, Early 2015)
  - macOS Mojave 10.14 (18A391)
  - Processor Name: Intel Core i5
  - Processor Speed: 2.7 GHz
  - Number of Processors: 1
  - Total Number of Cores: 2 with hyperthreading
  - L2 Cache (per Core): 256 KB
  - L3 Cache: 3 MB
  - Memory: 8 GB 1867 MHz DDR3
  - clang --version
    - Apple LLVM version 10.0.0 (clang-1000.11.45.2)
    - Target: x86_64-apple-darwin18.0.0
    - Thread model: posix

*Versión Multiprocesador MonteCarlo*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#define N                10


  struct timeval startTime, stopTime;
  int started = 0;

  void start_timer() {
   started = 1;
   gettimeofday(&startTime, NULL);
  }

  double stop_timer() {
   long seconds, useconds;
   double duration = -1;

   if (started) {
     gettimeofday(&stopTime, NULL);
     seconds  = stopTime.tv_sec  - startTime.tv_sec;
     useconds = stopTime.tv_usec - startTime.tv_usec;
     duration = (seconds * 1000.0) + (useconds / 1000.0);
     started = 0;
   }
   return duration;
  }




long double montecarlo(int number_of_tosses){
```

```
  int number_in_circle = 0;
  int toss;


        #pragma  omp  parallel  for  shared(number_of_tosses)
reduction(+:number_in_circle) private(toss)
      for (toss = 0; toss < number_of_tosses; toss++) {
         double x= (double)rand()/RAND_MAX*2.0-1.0;//float in
range -1 to 1
          double y=(double)rand()/RAND_MAX*2.0-1.0;//float in
range -1 to 1
        double distance_squared= (x*x) + (y*y);
        //printf("%f - %f\n", x,y);
        if (distance_squared<=1){
          number_in_circle++;
        }
      }

                                      long      double      pi=
4*number_in_circle/((double)number_of_tosses);
      return pi;


}



int main(int argc, char const *argv[]) {
  srand ( time ( NULL));
  int number_of_tosses=1000000;
  ///
  int i=0;
   double ms;

   printf("starting...\n");
   ms = 0;
   long double pi=0;
   for (i = 0; i < N; i++) {
     start_timer();
     pi=montecarlo(number_of_tosses);
     ms += stop_timer();
```

```
   }
   printf("%Lf\n", pi);
   printf("avg time = %.5lf ms\n", (ms / N));

  return 0;
}
```

### *Versión Multiprocesador Count Sort*

```
/*
    José Ramón Fernando Romero Chávez
    A01700318

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#define SIZE 1000000
#define N               1
#define DISPLAY         100
#define MAX_VALUE   100
```

```
struct timeval startTime, stopTime;
int started = 0;

void start_timer() {
 started = 1;
 gettimeofday(&startTime, NULL);
}

double stop_timer() {
 long seconds, useconds;
 double duration = -1;

 if (started) {
   gettimeofday(&stopTime, NULL);
   seconds  = stopTime.tv_sec  - startTime.tv_sec;
   useconds = stopTime.tv_usec - startTime.tv_usec;
   duration = (seconds * 1000.0) + (useconds / 1000.0);
   started = 0;
 }
 return duration;
}

void random_array(int *array, int size) {
 int i;

 srand(time(0));
 for (i = 0; i < size; i++) {
```

```
   array[i] = (rand() % 100) + 1;
 }
}

void fill_array(int *array, int size) {
 int i;

 for (i = 0; i < size; i++) {
   array[i] = (i % MAX_VALUE) + 1;
 }
}

void display_array(char *text, int *array) {
 int i;

 printf("%s = [%4i", text, array[0]);
 for (i = 1; i < DISPLAY; i++) {
   printf(",%4i", array[i]);
 }
 printf(", ... ]\n");
}




void CountSort(int a[], int n) {
  int i, j, count;
```

```
  int * temp = malloc(n * sizeof(int));
   #pragma omp parallel for shared(a, n, temp) private(j, i,
count)
  for (i = 0; i < n; i++) {
    count = 0;
    for (j = 0; j < n; j++){
      if (a[j] < a[i]){
        count++;
      }
      else if (a[i] == a[j] && j < i){
        count++;
      }
    }
    temp[count] = a[i];
  }
  memcpy(a, temp, n * sizeof(int));
  free(temp);
}




int main(int argc, char* argv[]) {
 int i, *array;
 double ms;

 array = (int*) malloc(sizeof(int) * SIZE);
 random_array(array, SIZE);
 display_array("before", array);
```

```
 printf("starting...\n");
 ms = 0;
 for (i = 0; i < N; i++) {
   start_timer();
   CountSort(array, SIZE);
   ms += stop_timer();
 }
 display_array("after", array);
 printf("avg time = %.5lf ms\n", (ms / N));

 free(array);
 return 0;
}
```

# 3. Resultados.

## Montecarlo

MacBook-Pro-de-Ramon-2:MonteCarlo ramonromero$ ./single.out

starting...

3.142788

avg time = 20.22080 ms

MacBook-Pro-de-Ramon-2:MonteCarlo ramonromero$ ./multi.out

starting...

3.128072

avg time = 18.05260 ms

$S = \frac{T_s}{T_m}$ $S = \frac{20.22080\ ms}{18.052\ ms}$ $S = 1,12$

## Count Sort

La evaluación promedio de los resultados en `results.txt` es la siguiente.

$S = \frac{T_s}{T_m}$ $S = \frac{535.31900 ms}{243.072 ms}$ $S = 2.20$

***Como conclusión se puede afirmar que existe una gran mejora al paralelizar ciertos algoritmos con grandes entradas, sin embargo, como fue el caso de Montecarlo, la generación aleatoria de números, es un cuello de botella para la optimización de dicho algoritmo***

## 3. Referencias

[1]"Pi", En.wikipedia.org, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Pi. [Accessed: 17- Sep- 2018].

[2] [BRESHEARS] pp. 31 & 32.

[3]"Amdahl's law", En.wikipedia.org, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Amdahl%27s_law. [Accessed: 15- Sep- 2018].

[4]"Wolfram|Alpha", M.wolframalpha.com, 2018. [Online]. Available: http://m.wolframalpha.com/. [Accessed: 15- Sep- 2018].