Cómputo paralelo y Distribuido.
Ejercicios de OpenMP
ITESM
Junio 2015.

1. **Versión instalada de OpenMP.**

```
/*
Parallel and Distributed Computing Class
OpenMP

Practice 1 : First example with OpenMP
Name      :
*/

#include <omp.h>
#include <stdio.h>
int main(){

  printf("Este  es nuestro primer ejemplo en openMP\n");
  #pragma omp parallel
  {
    printf("\nHola mundo\n");
  }
  printf("\nLa version instalada de OpenMP es: %d\n\n",_OPENMP);
  return 0;
}
```

Preguntas de reflexión:

- ¿Cuántas veces se imprime? ¿Porqué?
- ¿Qué versión de OpenMP tienes?

2. **Número de cores disponibles en el equipo, ID actual del thread y número total de threads.**

```
/*
Parallel and Distributed Computing Class
OpenMP

Practice 2 : Getting Number of cores and num of threads
Name      :
*/
#include <omp.h>
#include <stdio.h>
int main(){
  printf("Este es nuestro segundo ejemplo en openMP\n");
```

```
#pragma omp parallel
 {
   //Initializing Parallel Region
   int NCores,tid,NPR,NTHR;
   NCores=omp_get_num_procs();  //get the number of available cores
   tid=omp_get_thread_num();    //get current thread ID
   NPR=omp_get_num_threads();   //get total number of threads. NPR
   NTHR=omp_get_max_threads();  //get number of threads requested.
   if(tid==0)
    {
       printf("%i : Number of available cores\t= %i\n",tid,NCores);
       printf("%i : Number of threads request\t= %i\n",tid,NTHR);
       printf("%i : Numero total de hilos \t= %i\n",tid,NPR);
    }
   printf("%i:Hello multicore user! I am thread %i out of
%i\n",tid,tid,NPR);
 }
 return 0;
}
```

3. **Suma de vectores en paralelo.**

```
/*
Parallel and Distributed Computing Class
OpenMP

Practice 3 :Parallelizing the Sum of 2 vectors
Name      :
*/

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

void Suma_Vec(int* a, int* b, int* c, int size)
{
 int i = 0;
#pragma omp parallel for
 for (i = 0; i < size; ++i)
   {
     c[i] = a[i] + b[i];
   }
}

int main(){
```

```
  int size = 500000;

  int a[size];
  int b[size];
  int c[size];

  printf("\nParallelizing the Sum of 2 vectors\n");


  Suma_Vec(a, b, c, size);
  printf("Sum realized successfully!!\n");
  return 0;
}
```

Preguntas de reflexión:

- ¿Qué sucede cuando se paraleliza el problema con tamaño de 1000 elementos con la máxima cantidad de hilos que soporta la computadora?
- ¿Qué sucede cuando se paraleliza el problema con un tamaño de 1000000 elementos?
  Utiliza la función "*omp_get_wtime()"* para obtener el tiempo de ejecución de la suma de los dos tamaños de vectores de la pregunta anterior.
- ¿Qué sucede si se paraleliza con 2, 4 u 8 threads?. Especificar el thread desde la variable de entorno. Realiza una tabla comparativa.
- ¿Qué instrucción se utiliza para especificar la cantidad de hilos dentro de código?
- ¿Qué sucede si se especifican más threads que cores disponibles en la computadora?
- Realiza una comparativa en tiempos de ejecución en forma secuencial y paralela utilizando un tamaño del vector de 500000 elementos. Explica el resultado.

4. Producto Punto.

a) Prueba el siguiente código con un tamaño de 100000 elementos en cada uno de los vectores. El vector "a" que contenga los número pares y el vector "b" los impares:

```
#pragma omp parallel for
  for (i=0; i < size; i++)
       result += array_a[i] * array_b[i];
```

Ejecuta varias veces el programa, ¿Qué sucede? ¿Porque?

b) Agrega la clausula *reduction*:

```
start = omp_get_wtime();
#pragma omp parallel for reduction(+:result)
 for (i=0; i < size; i++)
   result += (array_a[i] * array_b[i]);
end = omp_get_wtime();
```

1333323473354752

Preguntas de reflexión:

- ¿Qué sucede si se utiliza la clausula *reduction* en la paralelización del ciclo?
- Realiza una comparación entre la versión secuencial y paralela utilizando 2,4,8,... dependiendo de la cantidad máxima de hilos disponibles en tu equipo de cómputo.

5. Multiplicación de Matriz-Vector.

```
/*
Parallel and Distributed Computing Class
OpenMP

Practice 6 : Vector * Matrix
Name      :
*/

#include <stdio.h>
#include <omp.h>
#define SIZE 100


int main ()
{
 // Counters
 int i, j;
 // Working matrix and vectors
 float matrix[SIZE*SIZE], vector[SIZE], result[SIZE];

 // Initializing Matrix and Vector
 printf("\nInitializing Matrix [%d][%d] and Vector[%d] ...\n",
SIZE,SIZE,SIZE);
 for (i=0; i<SIZE; i++){
  vector[i] = i + 1.0;
 }
```

```
 for (i=0; i < SIZE; i++) {
  for (j=0; j < SIZE; j++) {
    matrix[i*SIZE+j] = i * 2.0;
  }
 }

 printf("\nStarting Multiplication Vector * Matrix ...\n");
 double start = omp_get_wtime();

 #pragma omp parallel for
        for (i=0; i < SIZE; i++){
                double sum=0;
                for (j=0;j< SIZE; j++){
                        sum+=matrix[i*SIZE+j] * vector[j];
                        }
                        result[i]=sum;
                        }
 double end = omp_get_wtime();

        for (i=0; i < SIZE; i++)
                printf("\t result=%f",result[i]);
 printf("\nMultiplication Vector * Matrix has FINISHED\n");
 printf("\nExecution Time = %f\n",end - start);
 return 0;
}
```

Preguntas de reflexión:

- Prueba el ejercicio con al menos cinco tamaños de datos y diferente cantidad de threads. ¿Qué sucede?
- Realiza una comparativa del ejercicio anterior con la versión secuencial.

6. **Multiplicación de matriz-matriz.**

Realiza la multiplicación de matriz-matriz (200x200) elementos.
Realiza una comparativa de la versión secuencial con la versión paralela utilizando diferente cantidad de threads.

Contesta las siguientes preguntas:

1.- A que se refiere la variable *shared* dentro de la zona paralelizable
2.- A que se refiere la variable *private* dentro de la zona paralelizable
3.- Que hace la siguiente instrucción:
    a) #pragma omp parallel num_threads(4);