Universidad Nacional de San Antonio Abad del Cusco

Departamento Académico de Informática

ALGORITMOS PARALELOS Y DISTRIBUIDOS

Práctica Nº 4

EJERCICIOS DE APLICACION

1. **OBJETIVO.**
   - Aplicar la programación paralela con OpenMP a la solución de diversos problemas que demandan mucho cómputo.
   - Utilizar adecuadamente las directivas de OpenMP para paralelizar un determinado problema
2. **INTRODUCCION.**

   A menudo la computación paralela se aplica a algoritmos de cálculo intensivo, en cuyo caso el objetivo principal suele ser acelerar el cálculo para obtener la respuesta en un tiempo inferior al que se tendría con un algoritmo secuencial. En otras ocasiones, el objetivo es poder abordar problemas de mayor dimensión, los cuales exceden la capacidad de memoria de un único computador.

   En ésta práctica presentamos ejemplos de aplicación de la programación paralela con OpenMP.
3. **DESARROLLO DE LA PRÁCTICA**

   3.1. PRODUCTO PUNTO.

   El producto punto o escalar entre dos vectores se realiza de acuerdo con la figura

$$\mathbf{A} \cdot \mathbf{B} = \begin{bmatrix} A_1 & A_2 & \dots & A_n \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_n \end{bmatrix} = A_1B_1 + A_2B_2 + \dots + A_nB_n$$

Si se quiere una solución paralela utilizando OpenMP, es decir que n hilos cooperen en la solución, se analiza primero como dividir las tareas entre los n hilos. La solución es que cada hilo trabaje con diferentes elementos de los vectores A y B y cada uno obtenga resultados parciales. Por ejemplo, si los vectores son de 15 elementos y se tienen 3 hilos, el producto punto se puede dividir como:

$$A \cdot B = A_0 \cdot B_0 + A_1 \cdot B_1 + A_2 \cdot B_3$$

Y cada uno realiza los siguientes cálculos:

$$A_1 \cdot B_1 = a_0 \cdot b_{0'} + a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3 + a_4 \cdot b_4$$

$$A_2 \cdot B_2 = a_5 \cdot b_5 + a_6 \cdot b_6 + a_7 \cdot b_7 + a_8 \cdot b_8 + a_9 \cdot b_9$$

$$A_3 \cdot B_3 = a_{10} \cdot b_{10} + a_{11} \cdot b_{11} + a_{12} \cdot b_{12} + a_{13} \cdot b_{13} + a_{14} \cdot b_{14}$$

Se observa que cada hilo realiza los mismos cálculos, pero sobre diferentes elementos del vector. Así que para asignar diferentes elementos del vector a cada hilo se puede utilizar el constructor for.

```
#include <stdio.h>
#include <iostream>
#include <omp.h>
using namespace std;
double prodpunto(double *a, double *b, int n)
{
      double res = 0;
      int i;
      #pragma omp parallel for reduction(+:res)
      for (i = 0; i < n; i++)
       {
            res += a[i] * b[i];
       }
      return res;
}
int main()
{
      int n=100;
      double *a = new double[n];
      double *b = new double[n];
      //--colocacion de datos al arreglo a
      for (int i = 0; i < n; i++)
            a[i] = i;
      //--colocaciòn de datos al arreglo b
      for (int j = 0; j < n; j++)
            b[j] = j * 10;
      //--calcular producto punto en paralelo
      double pp = prodpunto(a, b, n);
      cout << "El producto punto es: " << pp << endl;
      getchar();
      return(0);

}
```

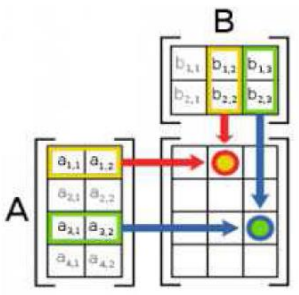
3.2. MULTIPLICACION DE MATRICES.

Los algoritmos basados en arreglos unidimensionales y bidimensionales a menudo son paralelizables de forma simple debido a que es posible tener acceso simultáneamente a todas las partes de la estructura,.

Un ejemplo común es el producto de dos matrices A y B de orden *nxm* y *mxr*. Recordando la fórmula y la forma de obtener un elemento de C:

$$C_{i\,j} = \sum_{k=0}^{n-1} a_{ik} b_{k\,j}$$

para $0 \leq i,\ j < n$



Para lograr dividir este problema se realiza un paralelismo de datos, donde cada hilo trabaje con datos distintos, pero con el mismo algoritmo.

```
#include <stdio.h>
#include "omp.h"
#include <time.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
int main(int argc, char const *argv[])
{
	int i, j;
	srand(time(NULL));
	int n;
	cout << "ingrese el valor de n: ";
	cin >> n;

	//--crea las matrices n x n
	int **matriz_a = new int*[n];
	for (int i = 0; i < n; i++) {
		matriz_a[i] = new int[n];
	}

	int **matriz_b = new int*[n];
	for (int i = 0; i < n; i++) {
		matriz_b[i] = new int[n];
	}

	int **matriz_c = new int*[n];
	for (int i = 0; i < n; i++) {
		matriz_c[i] = new int[n];
	}

	//--llena las matrices a y b con valores aleatorios
	for (i = 0; i < n; i++)
	{
		for (j = 0; j < n; j++)
		{
			matriz_a[i][j] = (rand() % 4) + 1;
			matriz_b[i][j] = (rand() % 4) + 1;
		}
	}
	/* muestra la matriz_a */
	for (i = 0; i < n; i++)
	{
		for (j = 0; j < n; j++)
		{
			printf("%d ", matriz_a[i][j]);
		}
		printf("\n");
	}
	printf("\n");
	/* muestra la matriz_b*/
```

```
        for (i = 0; i < n; i++)
        {
              for (j = 0; j < n; j++)
              {
                    printf("%d ", matriz_b[i][j]);
              }
              printf("\n");
        }

        printf("\n");
        /* realizar la multiplicación en paralelo */
      #pragma omp parallel
        {
              int i, j, k, suma = 0;
           #pragma omp for
              for (i = 0; i < n; i++)
              {
                    for (j = 0; j < n; j++)
                    {
                          for (k = 0; k < n; k++)
                          {
                                suma += (matriz_a[i][k] * matriz_b[j][k]);
                          }
                          matriz_c[i][j] = suma;
                          suma = 0;
                    }
              }
        }
      //-- muestra la matriz_c de resultados
        for (i = 0; i < n; i++) {
              for (j = 0; j < n; j++) {
                    printf("%d ", matriz_c[i][j]);
              }
              printf("\n");
        }

        getchar();
        getchar();
        return 0;
  }
```

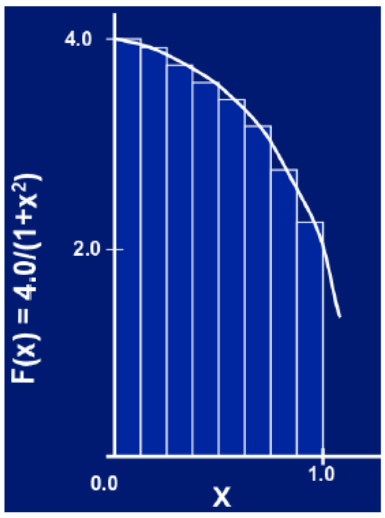
3.3. CALCULO DE PI

El valor de Pi se puede calcular mediante la siguiente integral:

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

Esta puede ser resuelta aproximadamente, sumando las áreas de los rectángulos debajo de la curva como se muestra en la figura, en la que δx es el ancho de cada rectángulo con altura igual a $F(x_i)$ en el medio del intervalo *i*

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



El programa paralelo es el siguiente:

```
#include <math.h>
#include <iostream>
#include <omp.h>
using namespace std;
double empezar, terminar;
int main(int argc, char *argv[])
{
      int n;
      cout << "introduce el número de rectángulos (n > 0): ";
      cin >> n;
      double PI25DT = 3.14159265358979323846264338;
      double h = 1.0 / (double)n;
      double sum = 0.0;

      empezar = omp_get_wtime();
     #pragma omp parallel for shared( sum )
       for (int i = 1; i <= n; i++) {
            double x = h * ((double)i - 0.5);
              sum += (4.0 / (1.0 + x*x));
       }
      double pi = sum * h;
      terminar = omp_get_wtime();
      cout << "El valor aproximado de PI es: " << pi
           << ", con un error de " << fabs(pi - PI25DT) << endl;
      cout << "el tiempo de procesamiento fue: "
             << fixed << terminar - empezar << endl;
      getchar();
      getchar();
      return 0;

}
```

CUESTIONARIO.

1. Cuando este programa se ejecuta para números de rectángulos pequeños, el resultado es aceptable. Pero cuando se ejecuta para números de rectángulos muy grandes, el resultado es erróneo.
   a. ¿A qué se debe este error?
   b. ¿Cómo se puede solucionar?

3.4. HISTOGRAMA DE UNA IMAGEN EN TONO DE GRISES.

El histograma es el número de pixeles de cada nivel de gris encontrado en la imagen. En el programa la imagen es representada por una matriz de NxN cuyos elementos pueden ser mayores o iguales a 0 y menores a NG (tonos de gris) y se cuenta el número de veces que aparece un número (pixel) en la matriz y éste es almacenado en un arreglo unidimensional .

Por ejemplo, para la matriz (imagen) de 5x5 de la figura que almacena valores de entre 0 y 5, en un arreglo unidimensional llamado histograma se almacena en el elemento con índice 0 el número de ceros encontrados, en el elemento con índice 1 el número de unos, en el elemento con índice 2 el número de dos etc..

Imagen

1	3	5	0	1
0	3	4	2	3
1	0	5	3	3
2	0	4	0	1
2	0	5	3	3

histograma

0	1	2	3	4
6	4	3	7	2

Para paralelizarlo, se realiza una descomposición de datos, es decir, se dividirá la matriz de forma que cada hilo trabaje con un número de renglones diferentes y almacene la frecuencia de aparición de cada valor entre 0 y NG de la sub-matriz en un arreglo privado o propio de cada hilo llamado *histop[]*. Después se deben sumar todos los arreglos *histop[]* de cada hilo para obtener el resultado final en el arreglo *histo[]*, tal como se puede ver a continuación.

1	3	5	0	1	Hilo 0
0	3	4	2	3	
1	0	5	3	3	Hilo 1
2	0	4	0	1	
2	0	5	3	3	

histop de hilo 0

0	1	2	3	4
2	2	1	3	1

histop de hilo 1

0	1	2	3	4
4	2	2	4	1

histo[ ]= histop de hilo 0 + histop de hilo1

0	1	2	3	4
6	4	3	7	2

El programa que realiza lo planteado es:

```
#include <stdio.h>
#include "omp.h"
#include <ctime>
#include <stdlib.h>
#include <iostream>

using namespace std;

int main()
{
	int i, j;
	int NG = 5; //--tamaño del histograma
	int N = 10;  //--tamaño de la imagen NxN
	int *histop = new int[NG]; //--histograma local
	int *histo = new int[NG];  //--histograma resultado
	//--inicializa el histograma resultado
	for (i = 0; i < NG; i++)
	   histo[i] = 0;


	//--crea la matriz imagen N x N
	int **IMA = new int*[N];
	for (int i = 0; i < N; i++) {
		IMA[i] = new int[N];
	}
	//-- generacion de la matriz imagen
	srand(time(NULL));
	for (i = 0; i < N; i++)
	{
		for (j = 0; j < N; j++)
		{
		  IMA[i][j] = rand() % 4 + 1;  //--genera un numero entero entre 0 y 4
		  IMA[i][j] = rand() % 4 + 1;
		}
```

```
  }

  /*Calculo del histograma de IMAGEN*/
#pragma omp parallel private(histop) num_threads(2)
  {
        for (i = 0; i < NG; i++)
              histop[i] = 0;
         #pragma omp for private(j)
        for (i = 0; i < N; i++)
              for (j = 0; j < N; j++)
                    histop[IMA[i][j]] ++;
         #pragma omp critical
          {
             for (i = 0; i < NG; i++)
              histo[i] += histop[i];
          }
  }
  //--imprime la matriz de imagen
  cout << "---MATRIZ IMAGEN--" << endl;
  for (i = 0; i < N; i++)
  {
        for (j = 0; j < N; j++)
        {
              printf("%d ", IMA[i][j]);
        }
        printf("\n");
  }
  printf("\n");

  //--imprime histograma
  cout << "--HISTOGRAMA--" << endl;
  for (i = 0; i < NG; i++) {
        cout << i << "-->" << histo[i] << endl;
  }
  cout << endl;
  system("pause");
  return(0);


   }
```

4. TAREA.

4.1. Diseñar un programa paralelo con OpenMP para calcular el resultado de multiplicar una matriz por un vector.

4.2. Escribir un programa paralelo con OpenMP para calcular la integral :

$$\int_b^a \frac{x^3}{3} + 4xdx$$

5. BIBLIOGRAFIA.

- Barbara Chapman, 2008 Using OpenMP. Masachuset Institute Techmnology.
- José E. Román. et. al. 2018. Ejercicios de Programación Paralela con OpenMP y MPI. Editorial de la Universidad Politécnica de Valencia.
- Barry Wilkinson. 2005. Parallel Programming. 2nd.Edition, Pearson Education USA.
- https://www.openmp.org/
- https://mapecode.com/programacion-paralela-con-openmp/
- http://javierferrer.me/paralelizar-c-openmp-introduccion/

Cusco noviembre de 2019.