Universidad Nacional de San Agustin
Esc. Profesional de Ciencia de la Computación

Algoritmos Paralelos

Tema: Laboratorio 5

Profesor: Alvaro Henry Mamani Aliaga
Alumno: Diego André Ranilla Gallegos

17 de Abril del 2017

1. Multiplicación Matriz - Vector con Pthreads

Para esta multiplicación vamos a emplear una matriz $A = (a_{ij})$ de mxn y un vector columna x de dimensión n de la forma $x = (x_0, x_1, ..., x_{n-1})^T$.

El producto de la multiplicación $Ax = y$ es un vector columna de dimensión m de la forma $y = (y_0, y_1, ..., y_{m-1})^T$

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int thread_count,M,N;
int **A, *X,*y;

int main(int argc, char * argv[]){
  thread_count = strtol(argv[1],NULL,10);
  M = strtol(argv[2],NULL,10);
  N = strtol(argv[3],NULL,10);

  //Generar matriz A y vector X e y
  .......

  long thread;
  pthread_t* thread_handles;
  thread_handles = malloc(thread_count*sizeof(pthread_t));

  for(thread = 0; thread < thread_count; thread++)
    pthread_create(&thread_handles[thread],NULL,multiplicacion,(
    void*)thread);

  for(thread = 0; thread < thread_count; thread++)
    pthread_join(thread_handles[thread],NULL);//*

  free(thread_handles);
  free(X);
  free(y);

  for( i=0 ; i < M ; i++){
    free(A[i]);
  }
  free(A);
  return 0;
}

void* multiplicacion(void *rank){
  long my_rank = (long)rank;
  int i,j;
  int local_m = M/thread_count;
  int first_row = my_rank*local_m;
  int last_row = (my_rank+1)*local_m -1;

  for(i=first_row;i<=last_row;i++){
    y[i] = 0.0;
    for (j = 0; j < N; j++){
      y[i] +=A[i][j]*X[j];
    }
  }
  return NULL;
}
```

Listing 1: Multiplicación matriz-vector

Para efectuar la multiplicación, la matriz A, los vectores x e y, m y n deben ser variables globales en otras palabras estas variables van hacer usadas por los distintos threads, por otro lado para que se efectúe la multiplicación vamos a repartir entre los threads las filas de la matriz A para que se encarguen de calcular la multiplicación y almacenar la respectiva respuesta en los distintos elementos de la matriz y.

2. Calculo de PI con Pthreads

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int thread_count,N=10000000;
double sum = 0.0;

void* calcularPi(void *rank);

int main(int argc, char * argv[]){
  if(argc !=2){
    printf("Debe ejecutar el programa con los siguientes valores en
     orden:\n");
    printf("[numero de hilos]\n");
    return 0;
  }

  thread_count = strtol(argv[1],NULL,10);

  long thread;
  pthread_t* thread_handles;
  thread_handles = malloc(thread_count*sizeof(pthread_t));

  for(thread = 0; thread < thread_count; thread++)
    pthread_create(&thread_handles[thread],NULL,calcularPi,(void*)
    thread);

  for(thread = 0; thread < thread_count; thread++)
    pthread_join(thread_handles[thread],NULL);//*

  free(thread_handles);

  sum = sum*4;
  printf("%f\n", sum);

  return 0;

}

void* calcularPi(void *rank){
  long my_rank = (long)rank;
  double factor;
  long long i;
  long long my_n = N/thread_count;
  long long my_first_i = my_n * my_rank;
  long long my_last_i = my_first_i + my_n;

  if(my_first_i %2 == 0)
    factor = 1.0;
  else
```

```
49      factor = -1.0;
50
51    for(i=my_first_i; i<my_last_i;i++,factor=-factor)
52      sum += factor/(2*i+1);
53    return NULL;
54  }
```

Listing 2: Calculo de PI

En este caso a comparación del punto anterior, se diferencian en cuanto a la integridad de los variables, en el punto anterior solo se leía los datos de los vectores para realizar la multiplicación y solo se modificaba los datos del vector y pero no había choques entre hilos sobre ese vector, ya que los hilos estaban organizados. Ahora en este caso un bloque de memoria va ser modificado de manera constante para hallar la respuesta final (la variable (sum), en este ejercicio usaremos una de las tantas formulas que hay para calcular π, en esta formula se empleará una variable N, que determinará hasta cuanto se calculará π, para paralelizar esto vamos a dividir los intervalos de cálculo entre los hilos, intervalos de N/t para cada hilo (t es el total de hilos), usando valores N muy altos vemos que el valor π mejora.

3. Busy Waiting y Mutex

Realizar pruebas y cambios de la tabla 4.1 del libro (Busy Waiting y Mutex).

Comparaciones usando $N = 10^8$, valores en segundos:

Threads	Busy-Wait	Mutex
1	0.889919	0.000285
2	5.914324	0.000376
4	31.866259	0.000311
8	50.334322	0.000589
16	83.348372	0.000383
32	90.003473	0.000785
64	300.343222	0.001626

Tabla 1: Busy Waiting y Mutex

4. Realizar experimentos y replicar los cuadros 4.5

	Matrix Dimension		
	$8,000,000$ x 8	8000 x 8000	8 x $8,000,000$
1	0.030972	0.246471	0.031805
2	0.028494	0.261604	0.061445
4	0.067492	0.508067	0.121055

Tabla 2: Tabla de resultados MUltiplicación Matrix-vector