Laboratorio de Arquitectura e Ingeniería de Computadores

PRÁCTICA IV

COMPUTACIÓN PARALELA (SISTEMAS SMP)

Ingeniería de Computadores

Pedro Barquín Ayuso
Miguel Ballesteros García

-Código:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <sys/time.h>
#include <errno.h>

#define n 100   //Definicion del numero de divisiones
#define nh 8    //numero de hebras

//globales
long double sumaGlobal;          //Suma total
long double h= 1.00F / n;  //Anchura
int nb=n/nh;               //numero de divisiones

pthread_t threads[nh];           //array de hilos

//Funcion que calcula
long double fy(int i, float h) {
      if (i == 0)
      {
            return 1.0F;
      }
      if (i == (int)n)
      {
            return 0.5F;
      }
      else
      {
            return (long double)(1.0 / (1 + (i*h)*(i*h)));
      }
}

void trabajar(void* arg){//función para trabajar con los hilos
en el caso normal
      int i=0;
      int f=0;
      long double suma = 0.0F;
      int number=(int)arg;

      i=nb*number;//asignacion del rango
      f=nb*number+nb;

      //Bucle que realiza las sumas
      for (i ; i < f; i++)
      {
            suma += (fy(i, h) + fy(i + 1, h));
      }

      //iniciamos el mutex
      int pthread_mutex_lock(pthread_mutex_t *mutex);
//Bloqueamos el mutex
      sumaGlobal=sumaGlobal+suma;
```

```
	int pthread_mutex_unlock(pthread_mutex_t *mutex);
//Desbloqueamos el mutex
	pthread_exit(NULL);//salida del hilo
}

void trabajarE(void* arg){//funcion para trabajar con los hilos
en el caso especial
	int i=0;
	int f=0;
	long double suma = 0.0F;
	int number=(int)arg;

	if(number==0){//asignacion del rango
		i=0;
		f=16;
	}else{
		i=16+12*(number-1);
		f=16+12*number;
	}

	//Bucle que realiza las sumas
	for (i ; i < f; i++)
	{
		suma += (fy(i, h) + fy(i + 1, h));
	}

	//hacer lo del mutex y escribir en la variable global
	int pthread_mutex_lock(pthread_mutex_t *mutex);
//Bloqueamos el mutex
	sumaGlobal=sumaGlobal+suma;
	int pthread_mutex_unlock(pthread_mutex_t *mutex);
//Desbloqueamos el mutex
	pthread_exit(NULL);//salida del hilo
}

int main(int argc, char *argv[]) {

	pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
//creamos un mutex de manera estÃ¡tica
	int i,r,trabajadores=0; //Divisiones / contador
trabajadores
	long double h2 = 0.0f;		//variable para H/2
	int status;		//variable para el estado de la
creacion de los hilos
	int inicial=0, final=0;		//Valores de los rangos
para los hilos

	struct timeval t_inicio, t_fin, t_dif; //variables para el
timepo
	struct timezone tz;

	gettimeofday(&t_inicio,&tz);	//INICIO contador tiempo

	//inicializamos global
	sumaGlobal = 0.0F;
```

```
      //calculo de las divisiones
      h = 1.00F / n;

      //calulo del numero de bloques por hebra
      nb=n/nh;

      //Creacion de los hilos dependiendo del casp
      if(n==100 && nh==8){  //caso raro ya que con 100 y 8 no se
puede hacer una division exacta
            final=16;
            for (trabajadores = 0; trabajadores < nh;
trabajadores++)
            {
                        status =
pthread_create(&threads[trabajadores], NULL, trabajarE, (void
*)trabajadores);
                        if (status != 0)
                {
                        fprintf(stderr, "status %d: %d\n", status,
strerror(status));
                }
            }
      }
      else if(nh==1){//en el caso de que sea un solo hilo entra
en esta seccion
            for (r=0 ; r < n; r++)
            {
            sumaGlobal += (fy(r, h) + fy(r + 1, h));//con un solo
hilo se realiza el bucle aqui
            }
      }
      else{           //caso normal
            for (trabajadores = 0; trabajadores < nh;
trabajadores++)
            {
                        status =
pthread_create(&threads[trabajadores], NULL, trabajar, (void
*)trabajadores);
                        if (status != 0)
                {
                        fprintf(stderr, "status %d: %d\n", status,
strerror(status));
                }
            }
      }


      //esperar hasta que terminen de trabajar los hilos y hayan
guardado en la variable global o en caso de ser un hilo solo no
hacer nada
      if(nh!=1){
            for (i=1; i<nh; i++) {
                  pthread_join(threads[i], NULL);
            }
      }
```

```
	//calcula H/2 que es el multiplicador
	h2 = h / 2.0F;

	//Multiplicacion 4 cuatro para obtener aproximacion de
(pi/4) a pi
	sumaGlobal = sumaGlobal*h2;
	sumaGlobal = 4.0f*sumaGlobal;
	printf("El resultado es: %.19Lf \n", sumaGlobal);

	// Tiempo fin
	gettimeofday(&t_fin,&tz);

	// Calculo del tiempo transcurrido
	if (t_inicio.tv_usec>t_fin.tv_usec)
	{
		t_fin.tv_usec += 1000000;
		--t_fin.tv_sec;
	}

	t_dif.tv_sec=t_fin.tv_sec-t_inicio.tv_sec;
	t_dif.tv_usec=t_fin.tv_usec-t_inicio.tv_usec;

	printf ("Tiempo de calculo = %ld segundos, %ld
microsegundos.\n", t_dif.tv_sec, t_dif.tv_usec);//muestra
contador tiempo
	return 0;
}
```

-Resultados:

	100	10000	100000	10000000
1	77 mics	301 mics	2288 mics	240329 mics
2	659 mics	821 mics	1953 mics	131104 mics
4	604 mics	743 mics	1145 mics	70792 mics
8	2002 mics	1978 mics	1039 mics	36825 mics

Como vemos en la tabla, con 100 divisiones va aumentando el tiempo al aumentar el número de hebras porque son tan pocas divisiones que se tarda más tiempo en crear los hebras que en realizar las operaciones. Vemos que también sucede esto con 10000 divisiones y es con 100000 divisiones cuando se empieza a apreciar que van disminuyendo los tiempos según aumentamos el número de hebras debido a que el tiempo en realizar las operaciones supera al de creación de las hebras. Todo esto se puede apreciar mejor con 10000000 de divisiones donde vemos que el tiempo prácticamente se divide por las hebras que usemos, pues el tiempo de creación de las hebras es despreciable comparado con el tiempo que se tarda en efectuar las operaciones para obtener el número pi.