

Informe de Laboratorio

Norman Patrick Harvey Arce

April 22, 2017

Fecha Abril 17
Docente Mg. Alvaro Mamani Aliaga

1 Implementar el problema Matriz-Vector usando PT-threads

```
void *Pth_mat_vect(void* rank)
{
    long my_rank = (long) rank;
    int i, j;
    int local_m = m/num_hilos;
    register int sub = my_rank*local_m*n;
    int my_first_row = my_rank*local_m;
    int my_last_row = (my_rank+1)*local_m - 1;
    double tmp;

    for (i = my_first_row; i <= my_last_row; i++)
    {
        tmp = 0.0;
        for (j = 0; j < n; j++)
            tmp += A[sub++]*x[j];
        y[i] = tmp;
    }
    return NULL;
}
```

Lo que hace este fragmento de código es asignar un número de filas para cada hilo (el proceso de multiplicación y suma). Por lo que el número de procesos debe ser divisor del número de filas(m) y número de columnas(n).

2 Implementar el problema del Calculo de PI

2.1 Busy-Waiting

```
for (thread = 0; thread < num_hilos; thread++)
    pthread_create(&thread_handles[thread], NULL, Thread_sum, (void*)thread);

for (thread = 0; thread < num_hilos; thread++)
    pthread_join(thread_handles[thread], NULL);
sum = 4.0*sum;
GET.TIME(fin);

printf("n=%lld_terminos,\n", n);
printf("Multi-hilo=%%.15f\n", sum);
printf("Tiempo_de_ejecucion=%%.15f\n", fin-inicio);

GET.TIME(inicio);
sum = Serial_pi(n);
GET.TIME(fin);
printf("Con_un_solo_hilo=%%.15f\n", sum);
printf("Tiempo_de_Ejecucion:=%%.15f\n", fin-inicio);
printf("Valor_de_libreria_MATH=%%.15f\n", 4.0*atan(1.0));

free(vals);
free(thread_handles);
```

2.2 Mutex

```
thread_handles = (pthread_t*) malloc (num_hilos*sizeof(pthread_t));
pthread_mutex_init(&mutex, NULL);

GET.TIME(inicio);
sum = 0.0;
for (thread = 0; thread < num_hilos; thread++)
    pthread_create(&thread_handles[thread], NULL, Thread_sum, (void*)thread);

for (thread = 0; thread < num_hilos; thread++)
    pthread_join(thread_handles[thread], NULL);
sum = 4.0*sum;
GET.TIME(fin);

printf("n=%lld_terminos,\n", n);
printf("Resultado=%%.15f\n", sum);
printf("Tiempo_de_ejecucion:=%%.15f\n", fin - inicio);
GET.TIME(inicio);
sum = Serial_pi(n);
GET.TIME(fin);
printf("Con_un_solo_hilo=%%.15f\n", sum);
printf("Tiempo_de_ejecucion:=%%.15f\n", fin - inicio);
printf("pi=%%.15f\n", 4.0*atan(1.0));

pthread_mutex_destroy(&mutex);
free(thread_handles);
```

Cuando son menos hilos que procesadores la diferencia no es mucha en cuanto al tiempo de ejecución, ya que cada hilo entra solo una vez a la sección crítica. Sin embargo, si aumentamos el número de hilos, solamente la versión de Busy-Waiting se verá afectada ya

que cada hilo podría entrar en un proceso de espera y esto afectaría al rendimiento global del algoritmo.

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

Se hizo las pruebas en un computador de 4 núcleos con n=1000.

Tiempos de Ejecución Pi: Busy-Waiting y Mutex, n=100000000		
Hilos	Busy-Wait	Mutex
1	0.02510	0.00221
2	0.00293	0.03879
4	0.00284	0.00527
8	0.005275	0.008358
16	0.01108	0.001864
32	0.01439	0.002451
64	0.02855	0.005865

Table 1: Tiempo de ejecución para el programa PiBusyWaiting y PiMutex

Como se observa, el tiempo de ejecución para Busy-Wait se degrada debido a que el número de hilos sobrepasa al número de procesadores del computador.

4 Implementar la lista enlazada multithreading y replicar las tablas 4.3 y 4.4

4.1 Tabla 4.3

Linked List: 1000 Claves iniciales, 100,000 operaciones, 99.9% Miembro, 0.05% Inserción, 0.05% Borrado

Linked list				
Implementation	1	2	4	8
Read Write Locks	0.0214	0.0142	0.0084	0.0135
One mutex for entire List	0.0201	0.0478	0.0422	0.0498
One mutex per node	0.1467	0.647	0.344	0.268

Table 2: Linked List

4.2 Tabla 4.4

Linked List: 1000 Claves iniciales, 100,000 operaciones, 80% Miembro, 10% Inserción, 10% Borrado

Tiempos de Ejecución Pi: Busy-Waiting y Mutex, n=100000000		
Hilos	Busy-Wait	Mutex
1	0.02510	0.02510
2	0.00293	0.00293
4	0.00284	0.00284
8	0.005275	0.00854
16	0.01108	0.00854
32	0.01439	0.00845
64	0.02855	0.00545

Table 3: Linked List

Como se observa, el tiempo de ejecución para Busy-Wait se degrada

5 Realizar experimentos y replicar los cuadros 4.5

5.1 Tabla 4.5

Run-Times and Efficiencies of Matrix-Vector Multiplication (times are in seconds)

Tiempos de Ejecucion y eficiencia multiplicacion MatrixVector			
Hilos	8000000x8	8000x8000	8x8000000
1	0.28017	0.24407	0.29474
2	0.20329	0.19459	0.23099
4	0.12530	0.11764	0.20357

Table 4: Tiempos de eficiencia de la multiplicacion Matrix Vector con sus respectivas dimensiones

Un evento de Wirtle-miss, ocurre cuando un procesador intenta actualizar una variable que no está en la caché y debe acceder a memoria principal. Cuando ejecutamos la matriz de número de filas=8,000,000 va a tener más eventos de este tipo ya que cada elemento debe ser inicializado.

Finalmente para las matrices de número de columnas = 8,000,000, la diferencia es que el valor X va a ser leído más veces y esto afecta el tiempo de ejecución.

6 Implementar el problema presentado en la sección 4.11 del uso de strtok

```

long my_rank = (long) rank;
int count;
int next = (my_rank + 1) % num_hilos;
char *fg_rv;
char my_line[MAX];
char *my_string;

sem_wait(&sems[my_rank]);

```

```

fg_rv = fgets(my_line, MAX, stdin);
sem_post(&sems[next]);
while (fg_rv != NULL)
{
    printf("Hilo_%ld>_mi_linea_=%s", my_rank, my_line);
    count = 0;
    my_string = strtok(my_line, "\\t\\n");
    while ( my_string != NULL )
    {
        count++;
        printf("Hilo_%ld>_Palabra_%d_=%s\\n", my_rank, count, my_string);
        my_string = strtok(NULL, "\\t\\n");
    }
    if (my_line != NULL) printf("Hilo_%ld>_despues_de_tokenizar ,_mi_linea_=%s\\n", my_rank, my_line);
    sem_wait(&sems[my_rank]);
    fg_rv = fgets(my_line, MAX, stdin);
    sem_post(&sems[next]);
}

```

El problema con la función STRTOK es que no sabe donde se quedó el anterior hilo que entró a usarla. Es decir, no almacena un puntero el cual nos indique desde dónde debemos comenzar con el siguiente hilo. Esta función es insegura cuando trabajamos con multi-hilos; pero tiene su solución. Y es la función: R-STRTOK la cual almacena un puntero dentro de sus parámetros, el cual va a indicar al siguiente hilo que utilice la función, desde dónde debe seguir trabajando.