Programación Paralela

Tarea 1

Alumno: Cristhian Wiki Sánchez Sauñe

Ejercicio 1

Por cuestiones de practicidad, diseñe un algoritmo en lugar de un suedocódigo en ambos ejemplos.

Dados dos procesadores y la tarea de actualizar la cuenta de un banco.

1. Diseñe un seudocódigo (enumere los pasos necesarios) para la solución de este problema utlizando el paradigma de memoria compartida

    i. Ambos procesos comienzan haciendo una lectura del objecto **cuenta**, y lo almacenan en la variable **x** e **y**

    ii. Antes de actualizarse con los sumandos, ambos procesos consultan si el objeto cuenta a sido modificado.

        a) En caso sea verdadero, la variable (**x** o **y**) del proceso se actualiza con el nuevo valor del objeto **cuenta** y se le añade el valor de su sumando. Y finalmente procede a actualizar el valor del objeto **cuenta** de forma definitiva llamando a la función **escribirCuenta()**.

        b) En caso sea falso, a la variable (**x** o **y**) del proceso se le añade su respectivo sumando, y se procede a actualizar el valor del objeto **cuenta** llamando a la función **escribirCuenta()**

    iii. Esperamos a que los procesos terminen para finalizar el programa.

2. Diseñe un seudocódigo (enumere los pasos necesarios) para la solución de este problema utlizando el paradigma de memoria distribuida

    i. Ambos procesos hacen una copia del objeto inicial **cuenta**, y los almacenan en sus respectivas variables (**x** e **y**)

    ii. Cada proceso actualiza sus variables con sus respectivos sumandos

    iii. Finalmente actualizan el valor del objeto inicial **cuenta** (para ambos procesos se tiene el mismo valor) con el valor del item anterior, usando la función **escribirCuenta()**

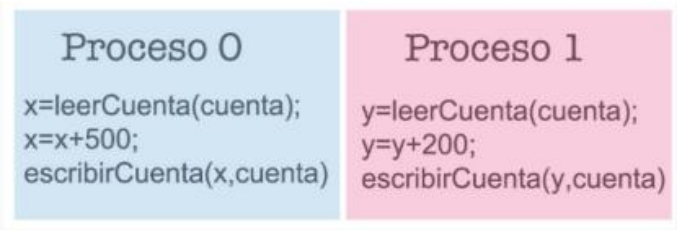
    iv. Con estos valores individuales, procedemos a sumarlos y posteriormente restamos 1 vez el valor inicial del objeto **cuenta** del item **i**

    v. Finalizamos ambos procesos y el programa habrá terminado

3. Nombre dos principales diferencias entre estas soluciones aplicadas al problema en cuestión

    a. Con la primera solución, el primer proceso necesita conocer el estado del segundo proceso en todo momento

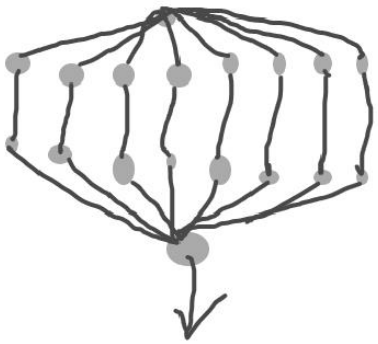
    b. Con la primera solucón nuestros resultandos son menos prospensos a errores pero aumenta la dificultad al momento de coordinar y programar los procesos.



Ejercicio 2

1. Diagrame en un árbol, el proceso de la suma de **n** enteros en paralelo.
Se tiene n=16 y disponibles 8 procesadores.

En este caso vamos a sumar 16 números. Como tenemos 8 procesadores, vamos a distribuir 8 pares de números a cada procesador (para que realicen su respectiva suma). Finalmente unimos estas sumas resultado de cada procesador, obteniendo el resultado total de la suma de nuestros 16 números.
Y aunque el cada subproceso funciona en un recurso de datos compartido, son independientes entre sí. Estamos usando la taxonomía llamada SIMD, donde ejecutamos solo una instrucción (sumar) en múltiples datos de forma paralela.



A continuación se muestra el algoritmo naive y el algoritmo mediante hilos.

Programación_Paralela native_sum.cpp

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_NO_OF_ELEMENTS 16

static long long int sum;
static int arr[MAX_NO_OF_ELEMENTS];

int main() {
    for (int i = 0; i < MAX_NO_OF_ELEMENTS; i++)
        arr[i] = i + 1;

    clock_t start, end;
    double cpu_time_taken;

    start = clock();

    // sumando

    for (int i = 0; i < MAX_NO_OF_ELEMENTS; i++)
        sum += arr[i];

    end = clock();
    cpu_time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("La suma total es: %lld\n", sum);

    printf("El tiempo total es %lf\n seg", cpu_time_taken);

    return 0;
}
```

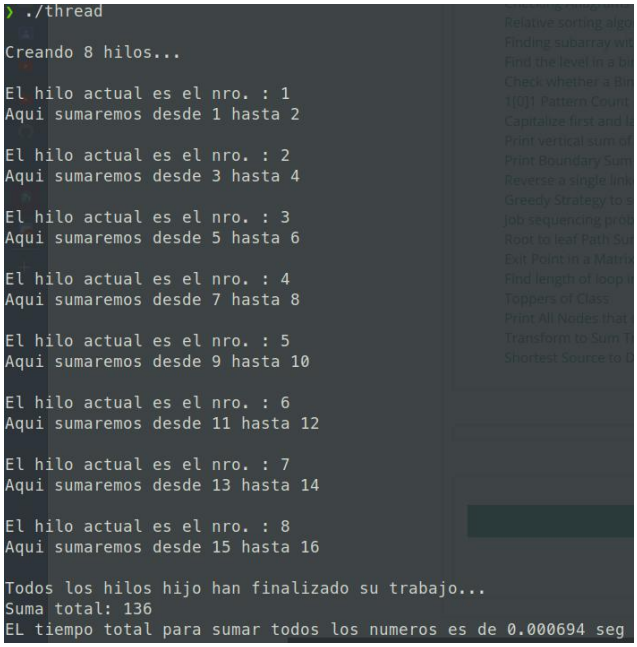
Programación_Paralela / thread_sum.cpp

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

#define MAX_NO_OF_THREADS 8
#define MAX_NO_OF_ELEMENTS 16

typedef struct arg_data {
    int thread_number;
} arg_data;

// Compartimos los datos en cada hilo para que trabajen concurrentemente
// sobre el arreglo que va a ser sumado
static int arr[MAX_NO_OF_ELEMENTS];
// Variable suma que almacenará la suma total
static long long int sum;

void* worker_sum(void* arg) {
    arg_data* current_thread_data = (arg_data*)arg;

    printf("\nEl hilo actual es el nro. : %d\n", current_thread_data->thread_number);

    int endpart = (current_thread_data->thread_number) * (MAX_NO_OF_ELEMENTS / MAX_NO_OF_THREADS);
    int startpart = endpart - (MAX_NO_OF_ELEMENTS / MAX_NO_OF_THREADS);

    printf("Aqui sumaremos desde %d hasta %d\n", arr[startpart], arr[endpart - 1]);

    long long int current_thread_sum = 0;
    for (int i = startpart; i < endpart; i++) {
        current_thread_sum += arr[i];
    }
    sum += current_thread_sum;

    return NULL;
}
```

```cpp
int main() {
    // el arreglo consistira de los MAX_NO_OF_ELEMENTS primeros enteros,
    // desde 1 hasta MAX_NO_OF_ELEMENTS
    for (int i = 0; i < MAX_NO_OF_ELEMENTS; i++) arr[i] = i + 1;

    // objetos pthread
    pthread_t id[MAX_NO_OF_THREADS];

    // data argumento para enviar en las funciones worker
    arg_data arg_arr[MAX_NO_OF_THREADS];

    // numero total de hilos que crearemos
    int no_of_threads = MAX_NO_OF_THREADS;
    printf("\nCreando %d hilos...\n", no_of_threads);

    clock_t start, end;
    double cpu_time_taken;

    start = clock();

    int thread_no = 1;

    //creando los hilos hijo
    for (thread_no = 1; thread_no <= MAX_NO_OF_THREADS; thread_no++) {
        arg_arr[thread_no - 1].thread_number = thread_no;
        pthread_create(&id[thread_no - 1], NULL, worker_sum, &arg_arr[thread_no - 1]);
    }

    //uniendo los hilos uno por uno
    for (int i = 1; i <= MAX_NO_OF_THREADS; i++) pthread_join(id[i - 1], NULL);

    end = clock();
    cpu_time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("\nTodos los hilos hijo han finalizado su trabajo...\n");

    printf("Suma total: %lld\n", sum);
```

También se midió el tiempo que demoran ambos algoritmos.

```
❯ ./native
La suma total es: 136
El tiempo total es 0.000002
 seg%
```



Ahora, se puede ver que el algoritmo ingenuo ha tomado menos tiempo, y parece que el algoritmo mediante hilos no es en absoluto útil, ya que ha tomado más tiempo. Pero ese no es el caso. Multithreading tiene un tiempo adicional debido a la creación de subprocesos, unión de subprocesos, etc. Desde entonces, aquí el programa era bastante simple y no requería un alto cálculo, es por eso que el algoritmo naive ha tomado menos tiempo aquí. Pero no será el caso si hace cálculos altos, como operar en cualquier dato grande donde no necesite que la ejecución ocurra secuencialmente. En tales casos, multithreading es extremadamente útil, ya que puede utilizar su CPU correctamente.

2. ¿Cuál es la forma más eficiente de hacer el cálculo si se tienen 5 procesadores?

3. ¿Qué método de paralelismo utiliza en cada caso?

Em la primera pregunta como ya se dijo, estamos usando la taxonomía llamada SIMD, donde ejecutamos solo una instrucción (sumar) en múltiples datos de forma paralela.