Instituto Tecnológico de Costa Rica
Centro Académico de Alajuela
IC3101. Arquitectura de Computadores

Laboratorio #2:
Paralelizando Códigos en arquitecturas Multiprocesador

Grupo 2:
Valery Carvajal Oreamuno – Carné: 2022314299
Raquel Gómez Zamora – Carné: 2022099256

Profesor:
Ing. Emmanuel Ramírez Segura

Fecha de entrega
30/05/2023

I Semestre, 2023

ÍNDICE

Objetivo

Objetivo General

Estudiar conceptos de paralelismo y concurrencia en el desarrollo de programas sobre arquitecturas multiprocesador.

Objetivos Específicos

1. Entender cómo paralelizar secciones de código sobre compiladores para determinar cómo mejorar el rendimiento en programas.
2. Programar códigos similares en compiladores paralelos y no paralelos para determinar rendimientos en tiempos de ejecución de programas.
3. Hacer una comparativa de rendimientos (tiempos de ejecución) entre programas diseñados en compiladores paralelos y compiladores que no ofrecen dicha funcionalidad.

Descripción de la solución

1. El código elaborado en C serial.

```c
#include <time.h>   // calcular tiempo
#include <stdio.h>
#include <stdlib.h>

// realiza la operación ( fila^3 + columna^3 ) / 2
int f(int fila, int columna) {
    int res = (fila * fila * fila + columna * columna * columna) / 2;
    return res;
}

int main() {
    int fils = 10000;
    int cols = 10000;

    // reserva memoria de matriz
    int** matriz = (int**)malloc(fils * sizeof(int*));
    for (int i = 0; i < fils; i++) {
        matriz[i] = (int*)malloc(cols * sizeof(int));
    }

    // llena la matriz
    clock_t inicio = clock();              // inicio medición de tiempo
    for (int i = 0; i < fils; i++) {
        for (int j = 0; j < cols; j++) {
            matriz[i][j] = f(i, j);
        }
    }
    clock_t fin = clock();                 // fin medición de tiempo

    // libera la memoria de  matriz
    for (int i = 0; i < fils; i++) {
        free(matriz[i]);
    }
    free(matriz);

    // cálculo del tiempo de ejecución
    double tiempo = (double)(fin - inicio) / CLOCKS_PER_SEC;
    printf("Tiempo de ejecucición: %f segundos\n", tiempo);
    return 0;
}
```

2. El código paralelizado en compilador paralelo de elección 1 (OpenMP).

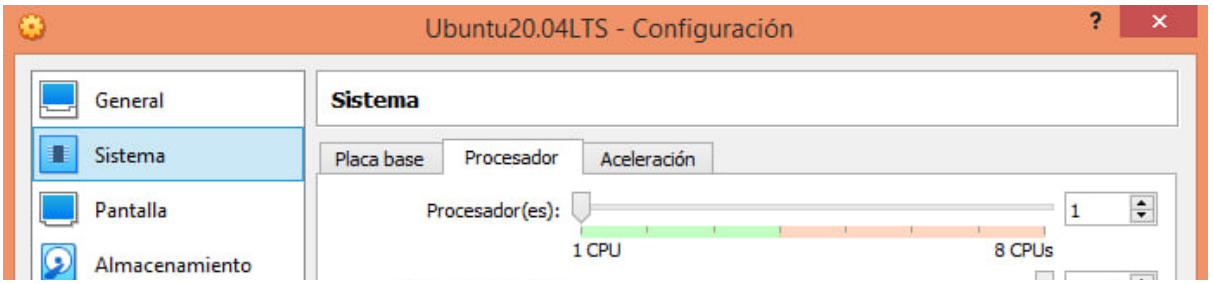
```c
#include <omp.h>  // para usar OpenMP
#include <stdio.h>
#include <stdlib.h>

// realiza la operación ( fila^3 + columna^3 ) / 2
int f(int fila, int columna) {
    int res = (fila * fila * fila + columna * columna * columna) / 2;
    return res;
}

int main() {
    int fils = 10000;
    int cols = 10000;

    // reserva memoria de matriz
    int** matriz = (int**)malloc(fils * sizeof(int*));
    for (int i = 0; i < fils; i++) {
        matriz[i] = (int*)malloc(cols * sizeof(int));
    }

    // llena la matriz
    double inicio = omp_get_wtime();       // inicio medición de tiempo
    #pragma omp parallel for
    for (int i = 0; i < fils; i++) {
        for (int j = 0; j < cols; j++) {
            matriz[i][j] = f(i, j);
        }
    }
    double fin = omp_get_wtime();          // fin medición del tiempo

    // libera la memoria de matriz
    for (int i = 0; i < fils; i++) {
        free(matriz[i]);
    }
    free(matriz);

    // cálculo del tiempo de ejecución
    double tiempo = fin - inicio;
    printf("Tiempo de ejecución: %f segundos\n", tiempo);
    return 0;
}
```

3. El código paralelizado en compilador paralelo de elección 2 (OpenCilk).

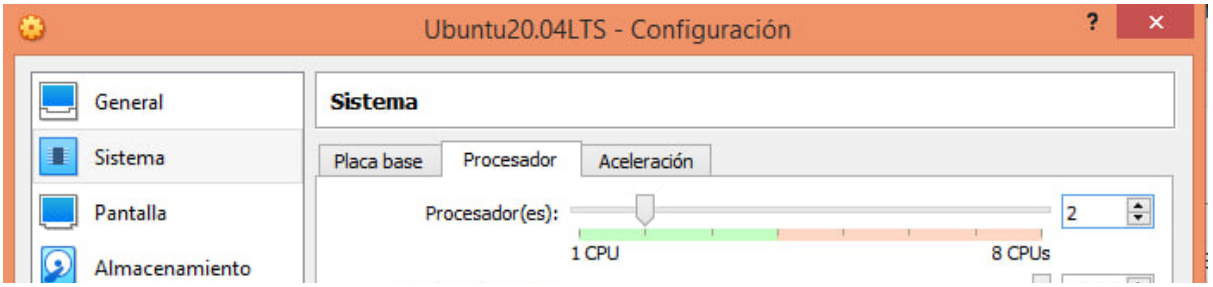
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define cilk_spawn _Cilk_spawn
#define cilk_sync  _Cilk_sync
#define cilk_for   _Cilk_for

// realiza la operación ( fila^3 + columna^3 ) / 2
int f(int fila, int columna) {
    int res = (fila * fila * fila + columna * columna * columna) / 2;
    return res;
}

int main() {
    int fils = 10000;
    int cols = 10000;
    // reserva memoria de matriz
    clock_t inicio = clock();              // inicio medición de tiempo
    int** matriz = (int**) malloc(fils * sizeof(int*));
    for (int i = 0; i < fils; i++) {
        matriz[i] = (int*) malloc(cols * sizeof(int));
    }

    // llena la matriz
    cilk_for (int i = 0; i < fils; i++) { // cilk_for
        for (int j = 0; j < cols; j++) {
            matriz[i][j] = f(i, j);
        }
    }
    clock_t fin = clock();                 // fin medición de tiempo

    // cálculo del tiempo de ejecución
    double tiempo = (double)(fin - inicio) / CLOCKS_PER_SEC;
    printf("Tiempo de ejecución: %f\n", tiempo);

    // libera la memoria de  matriz
    for (int i = 0; i < fils; i++) {
        free(matriz[i]);
    }
    free(matriz);
    return 0;
}
```

4. Imágenes y evidencias de los programas en ejecución en los puntos (1,2,3).

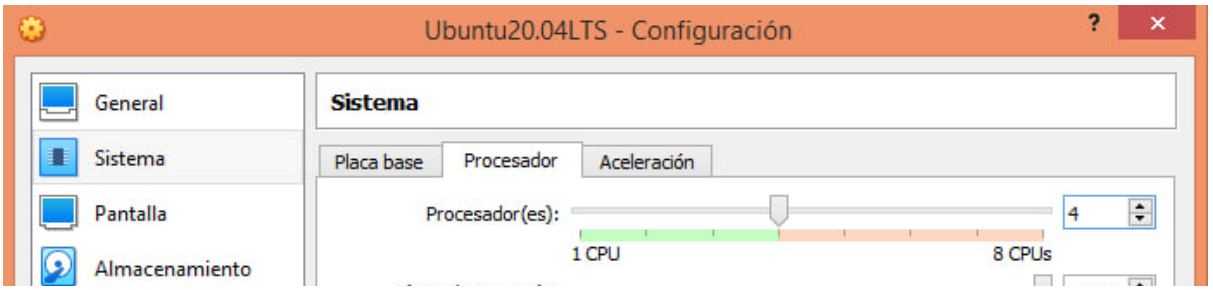
- Cambio de cantidad de procesadores
    - 1 procesador



    - 2 procesadores



    - 4 procesadores



- C serial
    - 1 procesador

```
svctiom@svctiom-VirtualBox:~/TEC/III_Semestre/arqui/lab2$ gcc -o lab2 lab2.c
svctiom@svctiom-VirtualBox:~/TEC/III_Semestre/arqui/lab2$ ./lab2
Tiempo de ejecución: 0.812922 segundos
```

    - 2 procesadores

```
svctiom@svctiom-VirtualBox:~/TEC/III_Semestre/arqui/lab2$ gcc -o lab2 lab2.c
svctiom@svctiom-VirtualBox:~/TEC/III_Semestre/arqui/lab2$ ./lab2
Tiempo de ejecución: 0.806157 segundos
```

    - 4 procesadores

```
svctiom@svctiom-VirtualBox:~/TEC/III_Semestre/arqui/lab2$ gcc -o lab2 lab2.c
svctiom@svctiom-VirtualBox:~/TEC/III_Semestre/arqui/lab2$ ./lab2
Tiempo de ejecución: 0.614709 segundos
```

- OpenMP
    - 1 procesador

```
svctiom@svctiom-VirtualBox:~/Downloads$ gcc -Wall -g -o lab2_OpenMP lab2_OpenMP.
c -fopenmp
svctiom@svctiom-VirtualBox:~/Downloads$ ./lab2_OpenMP
Tiempo de ejecución: 1.639824 segundos
```

    - 2 procesadores

```
svctiom@svctiom-VirtualBox:~/Downloads$ gcc -Wall -g -o lab2_OpenMP lab2_OpenMP.
c -fopenmp
svctiom@svctiom-VirtualBox:~/Downloads$ ./lab2_OpenMP
Tiempo de ejecución: 0.486752 segundos
```

    - 4 procesadores

```
svctiom@svctiom-VirtualBox:~/Downloads$ gcc -Wall -g -o lab2_OpenMP lab2_OpenMP.
c -fopenmp
svctiom@svctiom-VirtualBox:~/Downloads$ ./lab2_OpenMP
Tiempo de ejecución: 0.227541 segundos
```

- OpenCilk
    - 1 procesador

```
raquel@raquel-VirtualBox:~/Desktop$ ./lab2_OpenCilk
Tiempo de ejecución: 1.151800
```

    - 2 procesadores

```
raquel@raquel-VirtualBox:~/Desktop$ ./lab2_OpenCilk
Tiempo de ejecución: 0.835336
```

    - 4 procesadores

```
raquel@raquel-VirtualBox:~/Desktop$ ./lab2_OpenCilk
Tiempo de ejecución: 0.711935
```

5. Explicar y hacer comparativa en los tiempos de ejecución de los programas realizados (serial y paralelos). ¿Cuál es el más eficiente y por qué?

Para realizar la comparación en los tiempos de ejecución de los programas realizados, primero, se codificó un programa en C que realiza la función $(fila^3 + columna^3) / 2$ en cada celda de una matriz de tamaño 10.000 x 10.000. Luego, se investigó un poco más sobre los lenguajes de compiladores paralelos que fueron utilizados en este laboratorio (OpenMP y OpenCilk) y según la información encontrada, se realizaron las modificaciones necesarias para utilizar cada uno de manera correcta.

Al ejecutar los programas de las 3 maneras distintas y variando la cantidad de procesadores utilizados se obtuvieron los siguientes resultados:

Tiempos de ejecución en segundos

	1 CPU	**2 CPU**	**4 CPU**
Compilador en C	0.812922	0.806157	0.614709
Compilador paralelo #1 (OpenMP)	1.639824	0.486752	0.227541
Compilador paralelo #2 (OpenCilk)	1.151800	0.835336	0.711935

Según investigación previa, se sabía que OpenMP y OpenCilk trabajan de manera muy similar, esto se debe a que ambos son frameworks de programación paralela que se basan en el modelo *fork-join* donde una tarea pesada se divide en *k* cantidad de hilos con menor peso (*fork*) y luego, se recolectan sus resultados para unirlos en uno solo (*join*).

Sin embargo, en base al cuadro comparativo, a pesar de que con un solo procesador el más eficiente fue el compilador en C, si tomamos todos los datos, podemos ver como con más CPU el programa más eficiente en cuanto a temas de ejecución fue el de OpenMP con una gran diferencia en tiempo, esto gracias a que trabaja de manera paralela ejecutando varios “mini procesos” a la vez, a diferencia del compilador en C que va procesando uno por uno, es decir, trabaja de manera concurrente. También, es relevante notar cómo mientras más procesadores utiliza, el programa se vuelve más eficiente debido a que puede ejecutar varias tareas a la vez y en este caso, al ser un problema con varios subproblemas en él, el paralelismo es la mejor opción. Entonces, se puede concluir que en programas complejos o pesados, el paralelismo siempre va a ser más eficiente que la concurrencia.

Lecciones Aprendidas

Durante la ejecución de este laboratorio se logró poner en práctica lo aprendido acerca de los lenguajes de compiladores paralelos mencionados en clase OpenMP y OpenCilk, esto permitió ampliar el conocimiento sobre ellos y mejorar el desarrollo de los mismos. Aparte, se practicó la codificación en el lenguaje de programación C y el uso del ambiente virtual Oracle VM VirtualBox que fueron necesarios en el proceso de este laboratorio.

Bibliografía

Encontrar el tiempo de ejecución de UN programa en C. (n.d.). Techie Delight | Ace your Coding Interview. https://www.techiedelight.com/es/find-execution-time-c-program/

JArmstrong. (2018, October 15). OpenMP on Ubuntu. Medium. https://medium.com/swlh/openmp-on-ubuntu-1145355eeb2

OpenMP compilers & tools. (2022, November 22). OpenMP. https://www.openmp.org/resources/openmp-compilers-tools/

Platzi: Cursos online profesionales de tecnología. (n.d.). Platzi: Plataforma de aprendizaje profesional online. https://platzi.com/tutoriales/1189-algoritmos-2017/2010-paraleliza-tu-codigo-en-c-con-openmp/

Course, T. C. (n.d.). C program to find time taken by a program or function to execute in seconds. https://www.techcrashcourse.com/2016/02/c-program-to-find-execute-time-of-program.html#:~:text=To%20find%20the%20execution%20time,elapsed%20since%20the%20program%20started

Install OpenCilk. (n.d.-b). https://www.opencilk.org/doc/users-guide/install/