UNIVERSIDAD DEL VALLE DE GUATEMALA
Paralela, sección 20

Corto # 5
Diseño

Diego García - 22404
Joaquin Campos – 22155

Guatemala, Agosto 2025

1. Diseño de la Idea

Descripción de la Simulación:

El objetivo de la simulación es modelar el proceso de asignación de pacientes a doctores en un hospital. En este hospital, se tiene un número determinado de pacientes y doctores. Cada paciente debe ser atendido por un doctor, y el tiempo que cada paciente tiene que esperar depende de la cantidad de pacientes que su doctor tenga asignados.

En este escenario, el hospital tiene una capacidad limitada de doctores, por lo que los pacientes se distribuyen entre ellos. Los pacientes se asignan de manera cíclica a los doctores disponibles, es decir, el primer paciente se asigna al primer doctor, el segundo al segundo doctor, y así sucesivamente.

Cada doctor tiene un número de tareas (pacientes) que debe atender. El tiempo de espera de cada paciente es proporcional al número de pacientes que tiene asignados su doctor.

Partes Ejecutadas en Paralelo y Aplicación de OpenMP:

1. **`parallel for`:**
   La primera parte que se ejecutará en paralelo será la asignación de pacientes a doctores y el cálculo de sus tiempos de espera. El proceso de **asignación de pacientes** es independiente para cada uno de ellos, lo que permite paralelizar el bucle de asignación. Utilizaremos **`#pragma omp parallel for`** para dividir la tarea de asignar pacientes entre los hilos disponibles.

El cálculo del tiempo de espera se realiza en paralelo para cada paciente, ya que no depende de otros pacientes, solo de la cantidad de pacientes asignados a cada doctor.

Aplicación:

a. Utilizaremos `reduction(+:total)` para acumular el tiempo de espera de todos los pacientes sin generar condiciones de carrera, asegurando que la suma de tiempos de espera se realice de forma segura.

```
c
Copy
#pragma omp parallel for reduction(+:total) shared(data)
for (int i = 0; i < numPacientes; i++) {
    total += data[i];
```

```
}
```

2. **`sections`:**
   En la simulación también vamos a paralelizar la parte del cálculo del máximo y mínimo tiempo de espera entre los pacientes, lo cual se puede dividir en dos tareas independientes. Usamos **`#pragma omp parallel sections`** para ejecutar el cálculo del máximo y el mínimo en paralelo. Estas dos tareas no tienen dependencias entre ellas, lo que permite que se realicen en paralelo sin interferencia.

Aplicación:

a. Cada sección tendrá su propio cálculo de `max` y `min`, lo cual permite que los hilos trabajen de forma independiente en estas dos operaciones.

```
c
Copy
#pragma omp parallel sections
{
    #pragma omp section
    {
        max = data[0];
        for (int i = 1; i < numPacientes; i++) {
            if (data[i] > max) {
                max = data[i];
            }
        }
    }

    #pragma omp section
    {
        min = data[0];
        for (int i = 1; i < numPacientes; i++) {
            if (data[i] < min) {
                min = data[i];
            }
        }
    }
}
```

3. **`firstprivate`:**
   Las variables **`max`** y **`min`** deben ser privadas para cada hilo en la paralelización, ya que cada hilo necesita su propio valor inicial antes de que se realicen modificaciones. Usaremos la cláusula **`firstprivate`** para asegurarnos de que cada hilo reciba una copia inicial de estas variables antes de entrar en la sección paralela.

Aplicación:

   a. **`firstprivate(max, min)`** asegura que cada hilo tenga una copia privada y correctamente inicializada de estas variables antes de que se modifiquen dentro de las secciones.

```
c
Copy
#pragma omp parallel firstprivate(max, min)
```

4. **`shared`:**
   Las variables **`data[]`** (el array con los pacientes), **`total`**, **`max`** y **`min`** deben ser compartidas entre todos los hilos, ya que los resultados de estas variables deben ser accesibles por todos los hilos para la actualización de los resultados finales. La cláusula **`shared`** garantiza que estas variables sean accesibles por todos los hilos sin necesidad de crear copias locales.

Aplicación:

   a. Las variables **`data[]`**, **`total`**, **`max`** y **`min`** se compartirán entre los hilos, lo que permitirá que todos los hilos tengan acceso a la misma información para calcular los resultados de la simulación.

```
c
Copy
#pragma omp parallel shared(data, total, max, min)
```

5. **`reduction`:**
   Para evitar condiciones de carrera y asegurar que los resultados de las operaciones acumulativas (como el cálculo de la suma total de los tiempos de espera) sean correctos, utilizaremos la cláusula **`reduction`**. La cláusula **`reduction(+:total)`** permite que cada hilo realice su propia copia de la variable `total` y, al final, combine los resultados de todos los hilos de manera segura.

Aplicación:

a. En el cálculo de la suma de los tiempos de espera, la cláusula **`reduction(+:total)`** garantiza que los hilos no interfieran entre sí al sumar sus resultados parciales.

```
c
Copy
#pragma omp parallel for reduction(+:total) shared(data)
for (int i = 0; i < numPacientes; i++) {
    total += data[i];
}
```

Variables Compartidas y Privadas:

- **Variables Compartidas** :
  - **`data[]`**: El array de pacientes que debe ser accesible por todos los hilos.
  - **`total`**, **`max`**, **`min`**: Las variables globales que almacenan la suma total de los tiempos de espera, el valor máximo y el valor mínimo, respectivamente.
- **Variables Privadas**:
  - **`i`**: El índice de los bucles `for`, que será privado para cada hilo.
  - **`promedio`**: El promedio calculado, que será privado para cada hilo hasta que se calcule el total.
  - **`max`**, **`min`** (en cada hilo): Cada hilo tendrá su propia copia privada antes de la actualización de estos valores en la sección paralela.