Parallel Computing Project PHASE2

**Ezgi Karahan, 200201002**

**Ali Fatih Uzun, 200201003**

**Ahmet Bayraktar, 200201043**

**Alper Baydar, 200201052**

1. **Include Headers:**

In the first step, we included the necessary header files. This helps us define the libraries to be used in C++ and includes fundamental features required by the program.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. **Global Variables**

Global variables contain shared data between different sections of the program. In this step, we defined key variables such as matrix dimensions, random matrix, unit matrix, and result matrix.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. **Display Matrix Function:**

We defined a function to display the matrix on the screen. This function allows us to visualize the contents of the matrix at any point in the program.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. **Compute Element Function**

We implemented a function to compute a single element of the result matrix using a separate thread.

This function is executed by each thread and calculates one element of the result matrix. It takes an index shared between threads, performs its calculations using this index, and adds this value to the result matrix. Additionally, it shows which element each thread updated by printing the value it obtained during its execution. By using mutex, it prevents multiple threads from updating the result matrix at the same time, thus preventing concurrency issues.

**Important points:**

**index:** Index of the matrix element that the thread will process.

**row, col:** Row and column values ​​derived from indexes.

**temporary:** Variable used to temporarily store the calculation result.

**Main Function:** The main function receives the input matrix lengths from the user. Then, it generates the random matrix, the identity matrix, and the result matrix. Finally, it calls the computeElement function by specifying an index for each thread and initializes the threads. It waits for the created threads to complete and prints the result matrix to the screen.

**pthread\_t threads[MAX\_SIZE \* MAX\_SIZE];:** Defines an array that determines the maximum number of threads.

**int threadIndices[MAX\_SIZE \* MAX\_SIZE];:** An array holding the index that each thread will process.

**pthread\_create(&threads[threadCount], NULL, computeElement, &threadIndices[threadCount]);:** Associates each thread with the computeElement function and runs it.

**pthread\_join(threads[i], NULL);:** Waits for all threads to complete.

In this way, each thread calculates a matrix element and pushes it to the screen during these calculations. By using mutex, multiple threads are prevented from updating the result matrix at the same time, thus providing a safe multi-processing environment.

1. **Initialization**

Initialization involves generating random matrices, creating a unit matrix, and initializing the result matrix with -1. This step prepares the data before multithreaded computation.

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Initially, the resulting matrix has the value -1 with all its elements. In the first case, each value is assigned as -1 and this matrix is ​​made available as the place where the calculations will take place. In the first case the result matrix is:

ekran görüntüsü, kare, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

This initial state forms a basis from which each element of the result matrix will be calculated and updated by the workpiece. As calculations are made, each workpiece updates the value at its current location and this situation is printed on the screen. In this way, the initial state of the matrix is ​​observed and a persistence of it updated resulting matrix is ​​created.

1. **Multithreading**

We created threads to compute result matrix elements. Each thread is responsible for computing a specific element, and the program displays the matrix whenever a thread sets a value in the output matrix. The use of mutex ensures thread synchronization and avoids race conitions.

**OUTPUTS:**

The size of the first matrix (5x5) and the size of the identity matrix (3x3) were received from the user.

The first thread calculates the value (0,0) and updates the result matrix, and this status screen is printed. This process continues in a similar way with the other job.

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü, metin, menü içeren bir resim

Açıklama otomatik olarak oluşturuldu

And so on...

Each thread in turn calculates a value and updates the result matrix. The matrix status is printed after each update.

These steps show that each workpiece is worked in the same and different positions and the result matrix is ​​comprehensively updated. Each step begins with the change of the random matrix and the identity matrix in accordance with the changing dimensions, and then programming continues on these matrices according to the work distribution. During these programmings, each workpiece calculates the value at its location and updates the result matrix. The matrix status printed on the screen during these update operations shows the contribution of each workpiece.