Concurrent and Distributed Systems
Concurrent Problem Solving

Mitre Flavia Antonia

January 18, 2020

Third Year
Group: C.E.N. 3.1
Lecturer: Costin Bădică
Teaching assistant: Cristinel Ungureanu

1 Problem statement

1.1 Second problem

Develop and implement a program to multiply 2 matrices of size 1024x1024 using divide-et-impera.
The multiplication of matrices will be realized in a concurrent way using executors and the number of threads will be given by the number of available virtual CPU's.

2 Implementation

2.1 Second problem

For solving the first problem with fine synchronization I've created the classes:

- **Main**

  This class contains the main of the program.

  Initializing parallel and sequential matrix multiplications(a*b), and print the time elapsed for each one.
  If there are found errors in the computation print the errors count.
  Populating the matrices with rand values between [-500.0,500.0).
  This class computes: parallel multiplication, serial multiplication and also checks for differences between serial and parallel multiplication.

- **ParallelMatrixMultiplication**

  Final class used for parallel matrix multiplication, using Executors of MultiplyTask threads.
  Contains: First matrix operand, Second matrix operand, Result matrix.
  Exec is an executor of a fix number of threads, given by the number of available processors.
  In the constructor pass a and b matrices ,and instantiate the c matrix.
  Return the result matrix.
  Check for differences.
  Method multiplyRecursive(0, 0, 0, 0, 0, 0, a.length) is using MultiplyTask threads.

- **MultiplyTask**

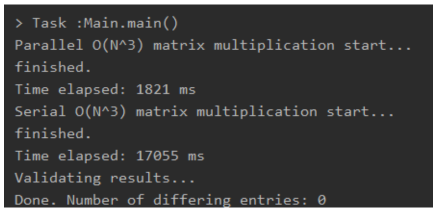
  Base class used for parallel matrix multiplication, using Executors and FutureTask classes.
  Contains: First matrix operand, Second matrix operand, Result matrix and Variables used for matrix segmentation.
  In the implementation of the run method, for every available processor, an element is produced.
  The constructor contains the variables used for matrix segmentation.
  The run is overrided here.

- **Sequentializer**

  Utility thread class used to simplify a FutureTask in MultiplyTask.
  Contains the variables for the first and the second tasks.
  They are used in the constructor.
  The run method runs boths tasks.

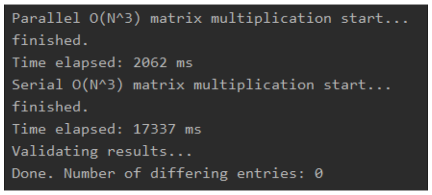
3 Experimental Data

3.1 Second problem



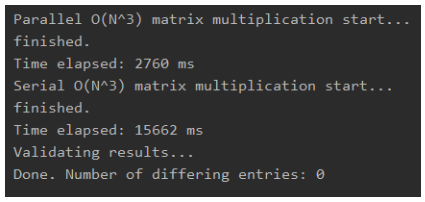
```
> Task :Main.main()
Parallel O(N^3) matrix multiplication start...
finished.
Time elapsed: 1821 ms
Serial O(N^3) matrix multiplication start...
finished.
Time elapsed: 17055 ms
Validating results...
Done. Number of differing entries: 0
```

Figure 1: First output



```
Parallel O(N^3) matrix multiplication start...
finished.
Time elapsed: 2062 ms
Serial O(N^3) matrix multiplication start...
finished.
Time elapsed: 17337 ms
Validating results...
Done. Number of differing entries: 0
```

Figure 2: Second output



```
Parallel O(N^3) matrix multiplication start...
finished.
Time elapsed: 2760 ms
Serial O(N^3) matrix multiplication start...
finished.
Time elapsed: 15662 ms
Validating results...
Done. Number of differing entries: 0
```

Figure 3: Third output

4 Conclusions

In computer science, divide and conquer is an algorithm design paradigm based on multi-branched recursion. A divide-and-conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

This divide-and-conquer technique is the basis of efficient algorithms for all kinds of problems, such as sorting (e.g., quicksort, merge sort), multiplying large numbers (e.g. the Karatsuba algorithm), finding the closest pair of points, syntactic analysis (e.g., top-down parsers), and computing the discrete Fourier transform (FFT).