ALGORITMI PARALELI ȘI DISTRIBUIȚI
Tema #1 Parallel Fast Fourier Transform

Termen de predare: 27-Oct-2019 23:55

Responsabili Temă: **Cristian Chilipirea, Dorinel Filip**

Objectives

Your goal is to implement a parallel, scalable with the number of threads, version of the Fourier Transform and Fast Fourier Transform algorithms, in C using the Pthread API.

Fourier Transform

A Fourier Transform is an extremely useful mathematical tool. It decomposes a function into the frequencies that make it up. In other words, it converts a signal from its original domain, time to a representation in the frequency domain.
The Fourier Transform has multiple uses, especially in digital signal processing, where it can be used for problems as complex as separating voice from music. Other interesting uses span from image compression to applications in financial analysis.
The formula of the Fourier transform is:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi}\, \mathrm{d}x \tag{1}$$

Because computers haven't figured out how to work with infinity, here we use the discrete Fourier transform:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi i}{N} kn} \tag{2}$$

The above formula needs to be calculated for all k values from 0 to $N-1$ with x_i representing the set of original values.

Fast Fourier Transform

As you might have concluded from the formula above, computing the Discreet Transform will have a complexity of $O(N^2)$.
The Fast Fourier Transform uses the symmetry of the function required to calculate the Fourier Transform to reuse much of the underlying computation and leads to time complexity of $O(N \log N)$
There is a lot of documentation available on the Fast Fourier Transform.
Here are a few places to start:

- YouTube 3Blue1Brown
- YouTube MIT OpenCourseWare
- YouTube Barry Van Veen
- YouTube LeiosOS

We recommend you to start your implementation from the C implementation available on Rosetta code.

- Rosetta code FFT.

You are allowed to use any sequential implementation, but consider that we verified the Rosetta implementation is correct, it can be parallelized, and we assume execution times based on it. We also expect the results to be within 0.001 from the ones on Rosetta.
!!! Keep in mind that the code on Rosetta only if the number of elements is a power of 2.

Helpful hints

Fourier Transform simple $O(N^2)$:

- Search for all possible read-write, write-write conflicts and treat them.
- If needed, feel free to use thread-local auxiliary variables to reduce thread lock contention.
- We recommend you to start the $numThreads$ threads only once. $numThreads$ is a program parameter.
- Make sure you use complex numbers and operations from complex.h as it will make your life a lot simpler. Most operations $(+, -, *)$ are defined for complex numbers.
- You may want to read the Rosetta code implementation for Fast Fourier transform before you implement the simple version.
- There is no need to parallelize reads and writes.

Fast Fourier Transform $O(N * log(N))$:

- You can use the Rosetta version or implement your own iterative version instead.
- We only require and test for 1, 2, and 4 threads.
- The cases for 1, 2, and 4 threads can be dealt with separately.
- You can look into using a recursion unrolling technique to reach an easier to parallelize version.
- If you do choose to do recursion unrolling (similar but different from loop unrolling), because of the thread limit you can stop after only two recursions.
- Think about data dependencies before you start coding.
- Remember that threads can be started from anywhere in your code, not just main.
  Just don't forget to call join and only use the parameter set number of threads.

Running, input/output

Your program will be run in the following way:

```
./homeworkFT inputValues.txt outputValues.txt numThreads
./homeworkFFT inputValues.txt outputValues.txt numThreads
```

Please assume the paths may be very long (up to 500 characters for each of the paths).

The input file format:

```
N // number of values
value1 // values are of type double
value2
...
valueN
```

The output file format:

```
N // number of complex type values represented as pairs of doubles
Re-value1 Img-value1 // values are of type double
Re-value2 Img-value2
...
Re-valueN Img-valueN
```

In the scaffolding resources, we offered two useful programs:

- compareOutputs compares two output files. Unlike diff it accepts a difference of 0.001 between values (in case there are differences created from the order of floating point operations).
- inputGenerator is the generator we use to create all input files with which we test the homework. It generates a file with N random numbers based on the given random seed.

To run them:

```
./compareOutputs file1 file2
./inputGenerator N file RANDOM_SEED
```

Points

The homework needs to be uploaded on the checker system. The points are as follows:

- 10 points - Correct Sequential Fourier Transform implementation.
- 40 points - Scalable parallel Fourier Transform implementation.
- 50 points - Scalable parallel Fast Fourier Transform implementation for N as a power of 2.

Any homework that does not compile gets 0 points.

Submission and Grading

Your archive should contain **only** the files “homeworkFT.c” and “homeworkFFT.c”.

The Homework will be graded based on the correctness and scalability obtained running the homework on a dedicated grading system for POLI and for ATM.

Any attempt to cheat/abuse the verification system will result in 0 points for all homeworks.