Take-Home Exam CUDA_FFT

Parallel Programming & Architectures

Consideration

- ✓ Your code is automatically graded using a script, and therefore, if your file/folder names are wrong you will receive a grade of **zero**. Please read and follow the instructions carefully. Common mistakes include
  - Different file or folder names
  - Different formatting of input or output
  - Not paying attention to case sensitiveness of C++ and Linux
- ✓ Go to the folder ~/the/cuda_fft/ in your home directory on the server and put your codes in this directory and remove any compiled binaries and test cases.
- ✓ Make sure your code compiles and runs without any error **on the server**. Your grade will be **zero** if any compile or runtime error occurs on the server. **Any!**
- ✓ The provided test cases, examples and sample codes (if any) are only to better describe the question. They are **not** meant for debugging or grading. It is your responsibility to think of and generate larger and more complex test cases (if necessary) in order to make sure your software works correctly for all possible scenarios.
- ✓ Start early and don't leave everything to the last minute. Software debugging needs focus and normally takes time.
- ✓ Just leave your final programs on the server. **Don't** email anything!
- ✓ Your grade is divided into several parts. In all cases, if you miss **correctness** (i.e. your code doesn't satisfy desired functionality), you miss other parts (e.g. speed, coding style, etc.) too. This rule is applied separately for each section of a take-home exam. So for example, in cuda_mm, your code might not be correct for M >= x but still you will get your grade for lower M values.
- ✓ Talking to your friends and classmates about this take-home exam and sharing ideas are *OK*. Searching the Internet, books and other sources for any code is also *OK*. However, copying another code is **not OK** and will be automatically detected using a similarity check software. In such a case, grades of the copied parts are **multiplied by -0.5**. Your work must be 100% done only by yourself, and you **should not** share parts of your code with others or use parts of other's codes. Online resources and solutions from previous years are part of the database which is used by the similarity check software.

fft.cu – Fast Fourier Transform

Grading (100):
correctness: 15
speed: 70
You will receive the speed grade only if your code is correct.

One-Dimension N-Point Discrete Fourier Transform (DFT) formula is:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}nk}. k = 0.1. \ldots . N-1$$

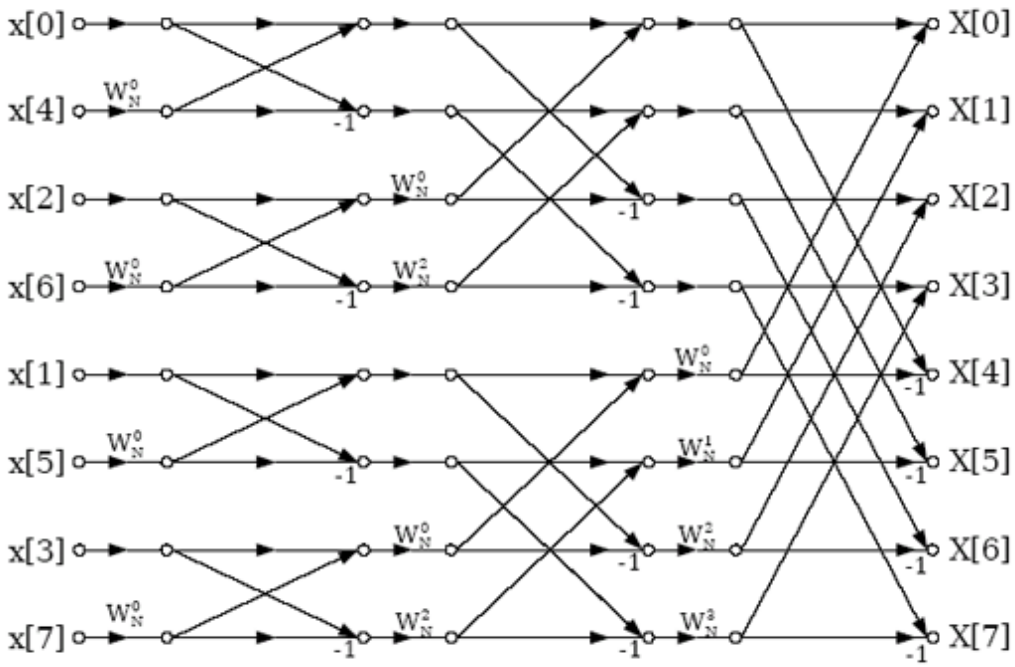
The above algorithm is O(N^2). Assume N=2^M. We can use divide-and-conquer method and compute $X[k]$ by separating $x[n]$ into two parts, odd values of n and even values of n. Read the details here:

- http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html
- https://en.wikipedia.org/wiki/Fast_Fourier_transform
- https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

Using Cooley-Tukey algorithm, the above equation can be re-written as:

$$X[k] = X_1[k] + W_N^k X_2[k]. k = 0.1 \ldots \frac{N}{2} - 1$$

where twiddle factor $W_N = e^{-j\frac{2\pi}{N}}$, X1[k] is N/2 point DFT of even half of x[n] and X2[k] is N/2 point DFT of odd half of x[n]. Figure below shows this computation for N=8. This is called radix-2 decimation in time FFT.



This method is O(N logN). There are logN stages, and in every stage there are N/2 butterfly operations. Each butterfly operation takes 2 complex values as input and generates 2 complex values as output. Read this:

https://en.wikipedia.org/wiki/Butterfly_diagram

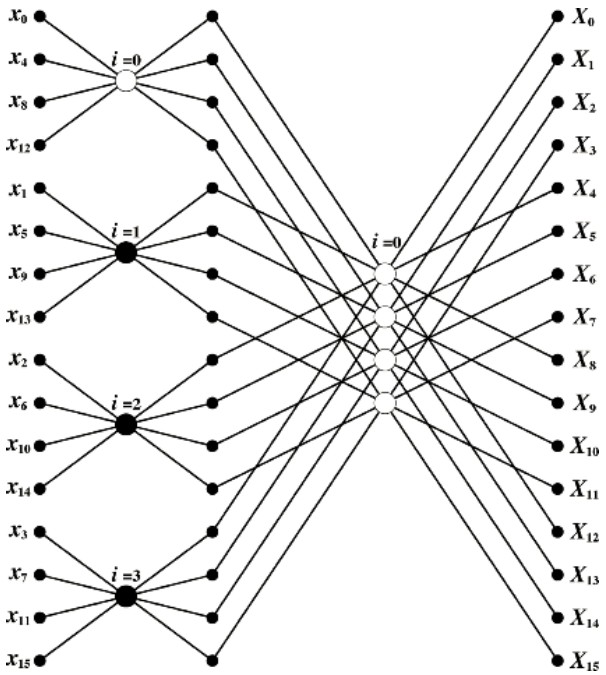
Implement the above FFT algorithm on GPU (gpuKernel function in fft.cu). Note that in each stage, there are N/2 butterfly operations which are all independent and can be executed in parallel. But every stage is dependent on results from previous stage. Note that N=2^M and M = 24, 25, and 26.

Compile: *nvcc -O2 fft_main.cu fft.cu -o fft*

Execute: *./fft 1 M*

Next, optimize the above algorithm using all or some of the following ideas Do not use other algorithms. Only use the following ideas. Read the paper "High Performance Discrete Fourier Transforms on GPUs". Note that the paper uses Stockham algorithm but you should not implement it. Use Cooley-Tukey instead. Use the paper to get inspired about the following ideas.

1. Use shared memory to compute FFT for small values of N, e.g., 1024 or 512.
2. Use hierarchical FFT on small shared memory FFT modules for larger values of N. The algorithm is explained in the above paper.
3. Use a higher radix (4 or 8). The following figure is 16-point FFT using radix 4. In most cases, radix 4 is faster than 2.
4. Use different radix values at different stages of your algorithm. Note that N=2^M and M = 23, 24, 25 and 26. Therefore, for example for N=25, you may decide to execute 12 stages with radix 4 and one last stage with radix 2, or any other combination which you think is faster.



Compile: *nvcc -O2 fft_main.cu fft.cu -o fft*

Execute: *./fft M*

Check correctness of your calculations by comparing the final values from GPU with results given by a serial method in CPU. Use the provided *gpuerrors.h*, *gputimer.h*, *fft_main.cu*, *fft.h* and *fft.cu* files to start your work. Only modify *fft.cu*. Check the speed of your calculations as shown in the provided *fft_main.cu* file.