IMPLEMENTATION OF DFT IN CPU AND GPU

Shengfeng Chen
Reference

- CUDA by Example: An Introduction to General-Purpose GPU Programming
- https://batchloaf.wordpress.com/2013/12/07/simple-dft-in-c/
Discrete Fourier Transform

- It converts a finite list of equally spaced samples of a function into the list of coefficients of a finite combination of complex sinusoids, ordered by their frequencies, that has those same sample values.
Discrete Fourier Transform

\[ X[k] = \sum_{n=0}^{N-1} x[n]e^{-\frac{j2\pi kn}{N}}, \quad k = 0 : N - 1 \]

De Moivre’s Theorem:

\[ e^{j\theta} = \cos(\theta) + jsin(\theta) \]
Discrete Fourier Transform

\[ X[k] = X_{re}[k] + jX_{im}[k] \]

\[ X_{re}[k] = \sum_{n=0}^{N-1} x[n] \cos\left(\frac{2\pi kn}{N}\right) \]

\[ X_{im}[k] = -\sum_{n=0}^{N-1} x[n] \sin\left(\frac{2\pi kn}{N}\right) \]
DFT in Matlab

- Time domain - Input sequence
- Frequency domain - Magnitude response
- Frequency domain - Phase response
DFT in C

\[
X_{re}[k] = \sum_{n=0}^{N-1} x[n] \cos\left(\frac{2\pi kn}{N}\right)
\]

\[
X_{im}[k] = -\sum_{n=0}^{N-1} x[n] \sin\left(\frac{2\pi kn}{N}\right)
\]

- for (k = 0; k < N; k++)
  - { 
    - Xre[k] = 0;
    - Xim[k] = 0;
    - for (n = 0; n < N; n++)
      - { 
        - Xre[k] += x[n] * \cos(n * k * TWOPI / N);
        - Xim[k] -= x[n] * \sin(n * k * TWOPI / N);
      } 
  }
CPU output
DFT Matrix Form

\[
\begin{bmatrix}
X[0] \\
X[1] \\
X[2] \\
\vdots \\
X[N-1]
\end{bmatrix} = \begin{bmatrix}
1 & 1 & 1 & 1 & \cdots & 1 \\
1 & W & W^2 & W^3 & \cdots & W^{N-1} \\
1 & W^2 & W^4 & W^6 & \cdots & W^{2(N-1)} \\
1 & W^3 & W^6 & W^9 & \cdots & W^{3(N-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & W^{N-1} & W^{2(N-1)} & W^{3(N-1)} & \cdots & W^{(N-1)(N-1)}
\end{bmatrix} \begin{bmatrix}
x[0] \\
x[1] \\
x[2] \\
\vdots \\
x[N-1]
\end{bmatrix}
\]

where \( W = e^{-j2\pi/N} \) and \( W = W^{2N} = 1 \).
A two-dimensional arrangement of blocks and threads

<table>
<thead>
<tr>
<th>block 0</th>
<th>thread 0</th>
<th>thread 1</th>
<th>thread 2</th>
<th>.</th>
<th>.</th>
<th>thread N-1</th>
<th>sum (block 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>block 1</td>
<td>thread 0</td>
<td>thread 1</td>
<td>thread 2</td>
<td>.</td>
<td>.</td>
<td>thread N-1</td>
<td>sum (block 1)</td>
</tr>
<tr>
<td>block 2</td>
<td>thread 0</td>
<td>thread 1</td>
<td>thread 2</td>
<td>.</td>
<td>.</td>
<td>thread N-1</td>
<td>sum (block 2)</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>block N-1</td>
<td>thread 0</td>
<td>thread 1</td>
<td>thread 2</td>
<td>.</td>
<td>.</td>
<td>thread N-1</td>
<td>sum(block N-1)</td>
</tr>
</tbody>
</table>
Sum of each block

Reference: *CUDA by example* (Chapter 5: Dot Product)
DFT in Cuda

```c
__global__ void dft(double *x, double *Xre, double *Xim){
    __shared__ double cache[2*threadsPerBlock];
    int n = threadIdx.x, k = blockIdx.x, cacheIndex = threadIdx.x;
    // matrix computation for Xim and Xre
    double temp1 = 0, temp2 = 0;
    while (n < msg && k < msg){
        temp1 += x[n] * cos(n * k * PI2 / msg);
        temp2 -= x[n] * sin(n * k * PI2 / msg);
        n += msg; k += msg;
    }
    cache[cacheIndex] = temp1;
    cache[cacheIndex + blockDim.x] = temp2;
    __syncthreads();
    // summation of each block
    int i = blockDim.x/2;
    while(i != 0){
        if (cacheIndex < i){
            cache[cacheIndex] += cache[cacheIndex + i];
            cache[blockDim.x + cacheIndex] += cache[blockDim.x + cacheIndex + i];
        }
        __syncthreads();
        i /= 2;
    }
    // load cache value into Xre and Xim
    if (cacheIndex == 0){
        Xre[blockIdx.x] = cache[0];
        Xim[blockIdx.x] = cache[blockDim.x];
    }
}
```
GPU Output
Elapsed time

Data length 128

- GPU: 9.500e-05 (128 threads)
- CPU: 4.035e-03

Speed up of 42.5
Application

- OFDM modulation/demodulation
Conclusion

- Implementation of DFT in GPU has significant improvement in terms of speedup. The larger size of data shows even more advantage in GPU than in CPU.
- It can be useful for many applications that need to process large data with DFT/FFT and obtain considerably speed up.
Question?