CUDA based Convolution and FFT on OPC

Guojin Chen

Department of Computer Science & Engineering
The Chinese University of Hong Kong

June 17, 2020

Outline

Background and problem formulation

- FORWARD LITHOGRAPHY Hopskins diffraction model.*

$$I(x,y) = \sum_{k=1}^{N^2} w_k \left|M(x,y) \otimes h_k(x,y)\right|^2, \quad x,y = 1,2,\dots N \tag{1}$$

- How can we accelerate the large kernel convolve operations.
- A universal way of CPU to GPU acceleration based on FFT algo..
- When can we use this kind of acceleration.

*Gao2014MOSAICMO.

Fast Fourier Transform: Applications

- Optics, acoustics, quantum physics, telecommunications, control systems, signal processing, speech recognition, data compression, image processing.
- DVD, JPEG, MP3, MRI, CAT scan.
- Numerical solutions to Poisson's equation.

FFT

The FFT is one of the truly great computational developments of this [20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT.
-Charles van Loan

Fast Fourier Transform: Brief History

Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.

Runge-König (1924). Laid theoretical groundwork.

Danielson-Lanczos (1942). Efficient algorithm.

Cooley-Tukey (1965). Monitoring nuclear tests in Soviet Union and tracking submarines.
Rediscovered and popularized FFT.

Polynomials: Coefficient Representation

- Polynomial: coefficient representation

$$\begin{aligned} A(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \\ B(x) &= b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1} \end{aligned} \tag{2}$$

- Add: $O(n)$ arithmetic operations.

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)\, x + \cdots + (a_{n-1} + b_{n-1})\, x^{n-1} \tag{3}$$

- Evaluate: $O(n)$ using Horner's method.

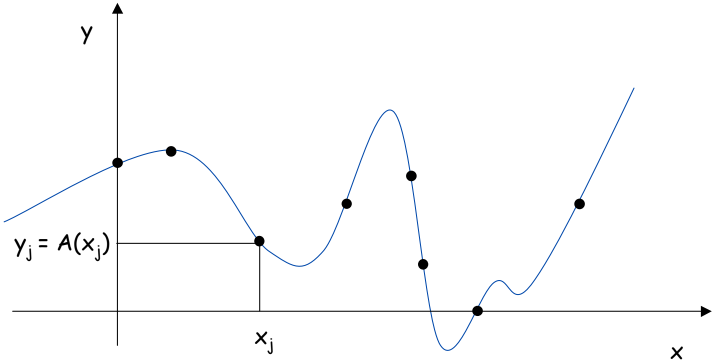
$$A(x) = a_0 + (x\,(a_1 + x\,(a_2 + \cdots + x\,(a_{n-2} + x\,(a_{n-1}))\cdots)) \tag{4}$$

- Multiply (convolve): $O(n^2)$ using brute force.

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j} \tag{5}$$

Polynomials: Point-Value Representation

- **Fundamental theorem of algebra:** [Gauss, PhD thesis] A degree n polynomial with complex coefficients has n complex roots.
- **Corollary:** A degree n-1 polynomial A(x) is uniquely specified by its evaluation at n distinct values of x.



Polynomials: Point-Value Representation

- Polynomial: point-value representation

$$\begin{aligned} A(x) &: (\mathrm{x}_0, y_0), \ldots, (\mathrm{x}_{\mathrm{n}-1}, y_{n-1}) \\ B(x) &: (\mathrm{x}_0, z_0), \ldots, (\mathrm{x}_{\mathrm{n}-1}, z_{n-1}) \end{aligned} \tag{6}$$

- Add: $O(n)$ arithmetic operations.

$$A(x) + B(x) : \quad (\mathrm{x}_0, y_0 + z_0), \ldots, (\mathrm{x}_{\mathrm{n}-1}, y_{n-1} + z_{n-1}) \tag{7}$$

- Multiply: $O(n)$, but need $2n - 1$ points.

$$A(x) \times B(x) : \quad (\mathrm{x}_0, y_0 \times z_0), \ldots, (\mathrm{x}_{2\mathrm{n}-1}, y_{2n-1} \times z_{2n-1}) \tag{8}$$

- Evaluate: $O(n^2)$ using Lagrange's formula.

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j=k} (x - x_j)}{\prod (x_k - x_j)} \tag{9}$$

table

- tradeoff: Fast evaluation or fast multiplication. We want both!

Representation	Multiply	Evaluate
Coefficient	$O(n^2)$	$O(n)$
Point-value	$O(n)$	$O(n^2)$

- Goal. Make all ops fast by efficiently converting between two representations.

$$\begin{tikzcd}
\boxed{a_0, a_1, \ldots, a_{n-1}} \arrow[r, shift left] & \boxed{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})} \arrow[l, shift left]
\end{tikzcd}$$

coefficient representation

point-value representation

Converting Between Two Polynomial Representations: Brute Force

- Coefficient to point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.
- Point-value to coefficient. Given n distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ that has given values at given points.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$O(n^2)$ for matrix-vector multiply

$O(n^3)$ for Gaussian elimination

$\uparrow$

Vandermonde matrix is invertible iff x_i distinct

Coefficient to Point-Value Representation: Intuition

- Coefficient to point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.
- Divide. Break polynomial up into even and odd powers.
  - $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$.
  - $A_{even}(x) = a_0 + a_2x + a_4x^2 + a_6x^3$.
  - $A_{odd}(x) = a_1 + a_3x + a_5x^2 + a_7x^3$.
  - $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$.
  - $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2)$.
- Intuition. Choose two points to be ś1.
  - $A(1) = A_{even}(1) + 1\, A_{odd}(1)$.
  - $A(-1) = A_{even}(1) - 1\, A_{odd}(1)$.

Can evaluate polynomial of degree ≤ n at 2 points by evaluating two polynomials of degree ≤ $\frac{1}{2}$n at 1 point.

Coefficient to Point-Value Representation: Intuition

- Coefficient to point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.
- Divide. Break polynomial up into even and odd powers.
  - $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$.
  - $A_{even}(x) = a_0 + a_2x + a_4x^2 + a_6x^3$.
  - $A_{odd}(x) = a_1 + a_3x + a_5x^2 + a_7x^3$.
  - $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$.
  - $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2)$.
- Intuition. Choose two points to be ś1, śi.
  - $A(1) = A_{even}(1) + 1\, A_{odd}(1)$.
  - $A(-1) = A_{even}(1) - 1\, A_{odd}(1)$.
  - $A(i) = A_{even}(-1) + i\, A_{odd}(-1)$.
  - $A(-i) = A_{even}(-1) - i\, A_{odd}(-1)$.

Can evaluate polynomial of degree $\leq n$ at 4 points by evaluating two polynomials of degree $\leq \frac{1}{2}n$ at 2 points.

Discrete Fourier Transform

- Coefficient to point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.
- Key idea: choose $x_k = \omega^k$ where ω is principal n^{th} root of unity.

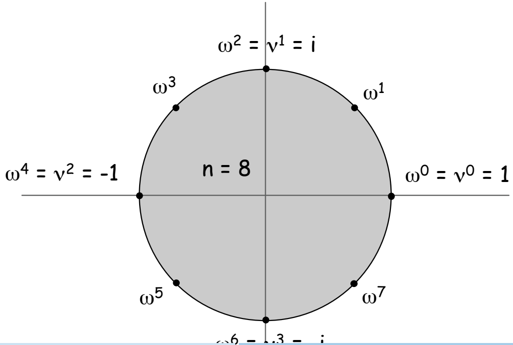
$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Discrete Fourier transform

Fourier matrix F_n

Roots of Unity

- Def. An n^{th} root of unity is a complex number x such that $x^n = 1$.
- Fact. The n^{th} roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i/n}$
- Pf. $(\omega^k)^n = (e^{2\pi i k/n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$
- Fact. The $\frac{1}{2}n^{th}$ roots of unity are: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$ where $\nu = e^{4\pi i/n}$
- Fact. $\omega^2 = \nu$ and $\left(\omega^2\right)^k = \nu^k$



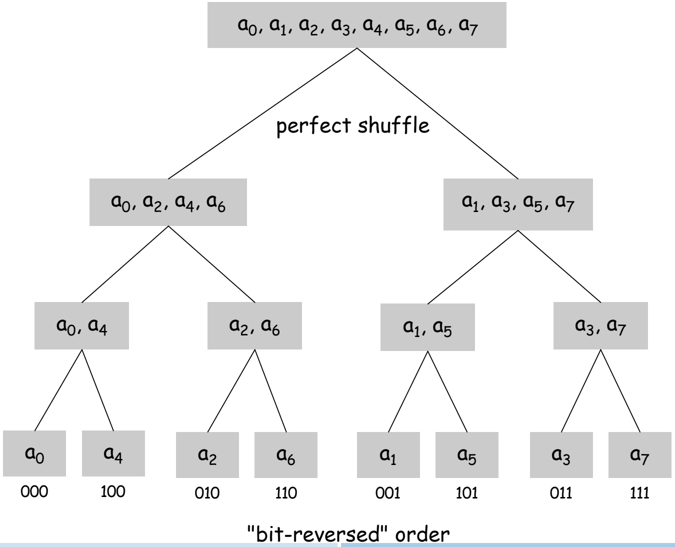
Fast Fourier Transform

- Goal. Evaluate a degree $n-1$ polynomial $A(x) = a_0 + \ldots + a_{n-1}x^{n-1}$ at its n^{th}roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$
- Divide. break polynomial up into even and odd powers.
  - $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n/2-2}\, x^{(n-1)/2}$.
  - $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n/2-1}\, x^{(n-1)/2}$.
  - $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$.
- Conquer. Evaluate degree $A_{\text{even}}(x)$ and $A_{\text{odd}}(x)$ at the $\frac{1}{2}n^{\text{th}}$ roots of unity: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$
- Combine.
  - $A(\omega^k) = A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2$
  - $A(\omega^{k+n}) = A_{even}(\nu^k) \underset{\uparrow}{-} \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2$

$\omega^{k+n} = -\omega^k$

$\boxed{\nu^k = (\omega^k)^2 = (\omega^{k+n})^2}$

Recursion Tree



FFT Algorithm

```
fft(n, $a_0, a_1, \ldots, a_{n-1}$) {
   if (n == 1) return $a_0$

   $(e_0, e_1, \ldots, e_{n/2-1})$ ← FFT(n/2, $a_0, a_2, a_4, \ldots, a_{n-2}$)
   $(d_0, d_1, \ldots, d_{n/2-1})$ ← FFT(n/2, $a_1, a_3, a_5, \ldots, a_{n-1}$)

   for k = 0 to n/2 - 1 {
      $\omega^k$ ← $e^{2\pi ik/n}$
      $y_k$       ← $e_k + \omega^k d_k$
      $y_{k+n/2}$ ← $e_k - \omega^k d_k$
   }

   return $(y_0, y_1, \ldots, y_{n-1})$
}
```

FFT Summary

- Theorem. Theorem. FFT algorithm evaluates a degree $n-1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ steps. assumes n is a power of 2.
- Running time. $\mathrm{T}(2n) = 2\mathrm{T}(n) + O(n) \Rightarrow \mathrm{T}(n) = n \log_2 n$

$$\begin{tikzcd}
\boxed{a_0, a_1, \ldots, a_{n-1}} \arrow[r, shift left, "O(n \log n)"] & \boxed{(\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})} \arrow[l, shift left]
\end{tikzcd}$$

coefficient representation

point-value representation

Inverse FFT: Algorithm

```
ifft(n, $a_0,a_1,\dots,a_{n-1}$) {
    if (n == 1) return $a_0$

    ($e_0,e_1,\dots,e_{n/2-1}$) ← FFT(n/2, $a_0,a_2,a_4,\dots,a_{n-2}$)
    ($d_0,d_1,\dots,d_{n/2-1}$) ← FFT(n/2, $a_1,a_3,a_5,\dots,a_{n-1}$)

    for k = 0 to n/2 - 1 {
        $\omega^k$ ← $e^{-2\pi ik/n}$
        $y_k$       ← ($e_k$ + $\omega^k$ $d_k$) / n
        $y_{k+n/2}$ ← ($e_k$ - $\omega^k$ $d_k$) / n
    }

    return ($y_0,y_1,\dots,y_{n-1}$)
}
```

Inverse FFT Summary

- Theorem. Inverse FFT algorithm interpolates a degree $n-1$ polynomial given values at each of the n^{th} roots of unity in $O(nlogn)$ steps.
- Running time. $\mathrm{T}(2n) = 2\mathrm{T}(n) + O(n) \Rightarrow \mathrm{T}(n) = n\log_2 n$

$$\begin{tikzcd} a_0, a_1, \ldots, a_{n-1} \arrow[r, shift left, "O(n \log n)"] & (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1}) \arrow[l, shift left, "O(n \log n)"] \end{tikzcd}$$

coefficient representation

point-value representation

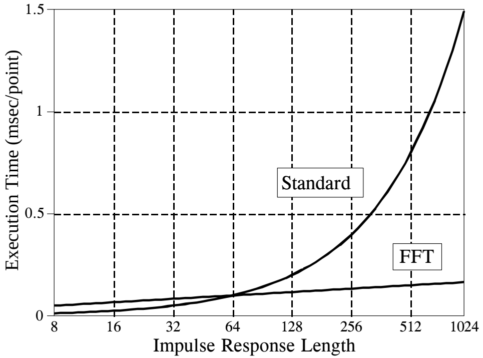
Polynomial Multiplication

- **Theorem.** Can multiply two degree $n-1$ polynomials in $O(nlogn)$ steps.

$$\begin{tikzcd}
\text{coefficient representation} & \text{coefficient representation} \\
\begin{matrix} a_0, a_1, \ldots, a_{n-1} \\ b_0, b_1, \ldots, b_{n-1} \end{matrix} \arrow[d, "\text{FFT}"', "O(n \log n)"] & c_0, c_1, \ldots, c_{2n-2} \\
\begin{matrix} A(x_0), \ldots, A(x_{2n-1}) \\ B(x_0), \ldots, B(x_{2n-1}) \end{matrix} \arrow[r, "\text{point-value multiplication}", "O(n)"'] & C(x_0), C(x_1), \ldots, C(x_{2n-1}) \arrow[u, "\text{inverse FFT}", "O(n \log n)"']
\end{tikzcd}$$

When we need FFT?

- Theorem. Depends on the length of the filter kernel.†
- The time for standard convolution is directly proportional to the number of points in the filter kernel.
- In comparison, the time required for FFT convolution increases very slowly (logarithm)



†smith1997scientist.

FFT: faster with more precision

- Excution time
  - Filter kernels shorter than about 60 points can be implemented faster with standard convolution.
  - Longer filter kernels can be implemented faster with FFT convolution.
  - With FFT convolution, the filter kernel can be made as long as you like, with very little penalty in execution time.
  - 16,000 point filter kernel only requires about twice as long to execute as one with only 64 points.
- Precision
  - The speed of the convolution also dictates the precision of the calculation.
  - Because the round-off error in the output signal depends on the total number of calculations, which is directly proportional to the computation time.
  - FFT convolution can be expected to be an order of magnitude faster, and an order of magnitude more precise.‡

‡**kleinberg2006algorithm**.

Outline

CUDA Acceleration for OPC problems

- FORWARD LITHOGRAPHY Hopskins diffraction model.

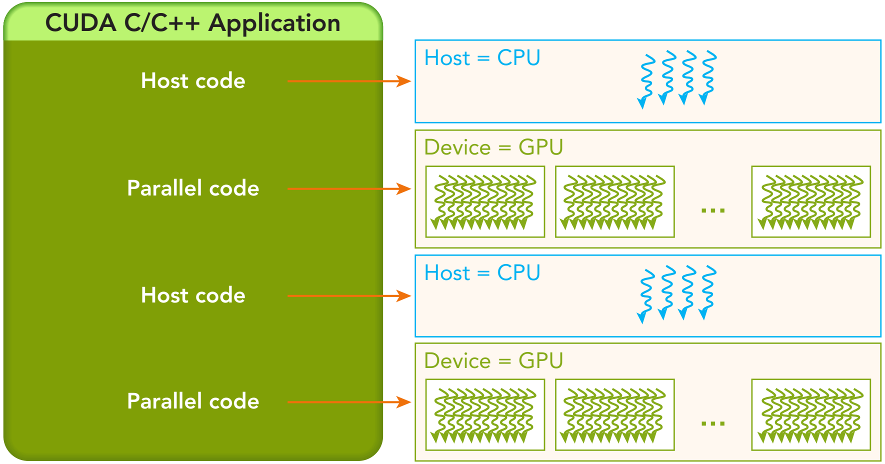
$$I(x,y) = \sum_{k=1}^{N^2} w_k \left| M(x,y) \otimes h_k(x,y) \right|^2, \quad x,y = 1,2,\ldots N \tag{10}$$

- Speedup for Kernel Convolution transform the non-quadratic form based on the properties of convolution.

$$M \otimes H = \sum_{k=1}^{N_h} w_k \cdot (M \otimes h_k) = \sum_{k=1}^{N_h} M \otimes (w_k \cdot h_k) = M \otimes \sum_{k=1}^{N_h} w_k \cdot h_k \tag{11}$$

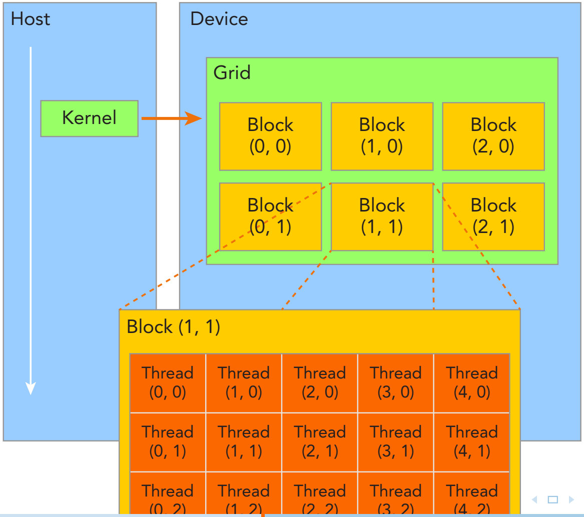
CUDA Programming Model

- CUDA applications arch. Host-Device model.



CUDA Threads Model

- CUDA threads arch. Grid-Block-Thread.



CPU to GPU transform

- Intuition . Eliminate all the **for** loop.
  - FFT $\rightarrow$ CUDA based FFT : eliminate the **for** loop in FFT algo.
  - Kernels $\rightarrow$ Stacked Kernels : eliminate the **for** loop for computing different kernels. Because there is 24 kernels, we can stack theme together and feed into cuda memory to compute together.
- Performance . Nearly 40 times speed up.

Thanks

Thank you.