Implementarea transformatei Fourier pe GPU

Moraru Radu-Andrei
Matache Alexandru
Blogu Toma-Adrian

1. Introducere

Transformata Fourier este o metoda utilizata foarte frecvent in procesarea semnalelor, inclusiv a imaginilor. Foarte multe aplicatii grafice o utilizeaza pentru aplicarea de filtre, generarea de date noi, etc. De la inceputul anilor 2000, placile video si platformele de programat pentru ele au avansat foarte rapid, oferind un potential foarte inalt de paralelizare. Astfel, ele sunt vazute ca niste platforme de executat operatii de tip SIMD (Single instruction, multiple data). Transformata Fourier rapida este un algoritm optimizat pentru obtinerea frecventelor care compun o imagine. In implementarea sa, transformata Fourier rapida 2D aplica un set de operatii identice pe linii, coloane si pixelii imaginii procesate. Din acest motiv, utilizarea unui procesor grafic pentru calcularea sa reprezinta un mod rapid de a procesa imaginea.

2. Algoritmul FFT

Pentru simplificarea ecuatiilor, definim:

$$W_{N/2} = e^{-\frac{2j\pi}{N/2}} \quad F'(u) = NF(u)$$

Formula transformatei fourier a unui semnal discret este

$$F'(u) = \sum_{x=0}^{N-1} f(x)\, W_N^{xu}$$

Putem extrage valorile functiei f pentru x par si impar cu functiile

$f^e(x) = f(2x)$ si $f^o(x) = f(2x+1)$

Formula transformatei devine astfel

$$\sum_{x=0}^{N/2} f^e(x)\ W_{N/2}^{xu} + W_N^u \sum_{x=0}^{N/2} f^o(x)\ W_{N/2}^{xu}$$

Cele doua sume reprezinta transformata Fourier a partitiilor lui f, deci formula poate fi definita printr-o recurenta

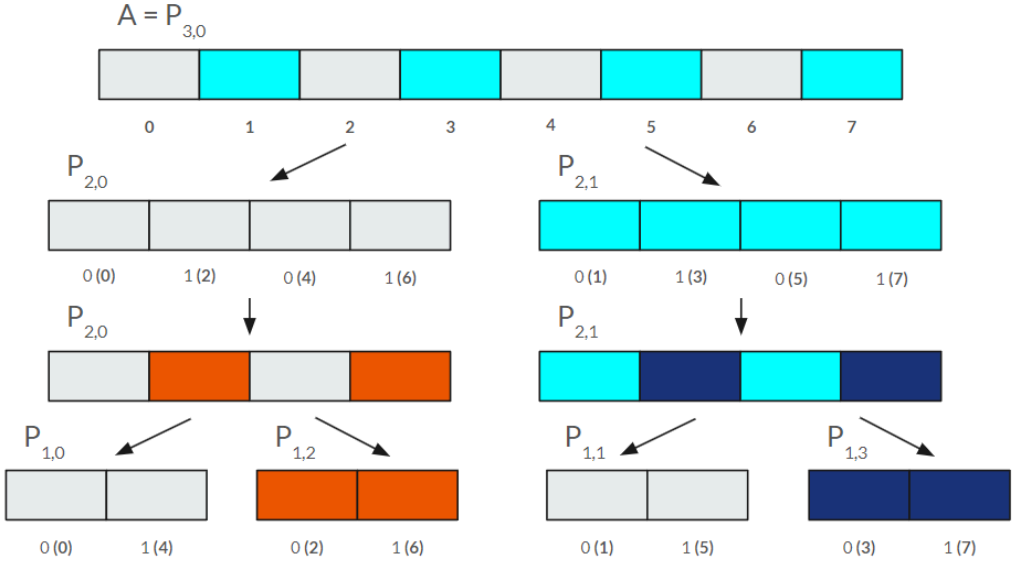
$$F'(u) = F'^e(u) + W_N^u F'^o(u)$$

Pornind de la definitia recurenta a transformatei Fourier a semnalului poate fi dezvoltat un algoritm Divide et Impera sau de programare dinamica pentru optimizarea operatiei.

3. Partitionarea semnalului pentru FFT

Consideram un semnal discret A. O prima solutie ar fi calcularea recursiva a transformatei Fourier pentru indecsii pari si cei impari ai semnalului. Totusi, aceasta solutie are complexitate O(n) pentru spatiu, lucru care este

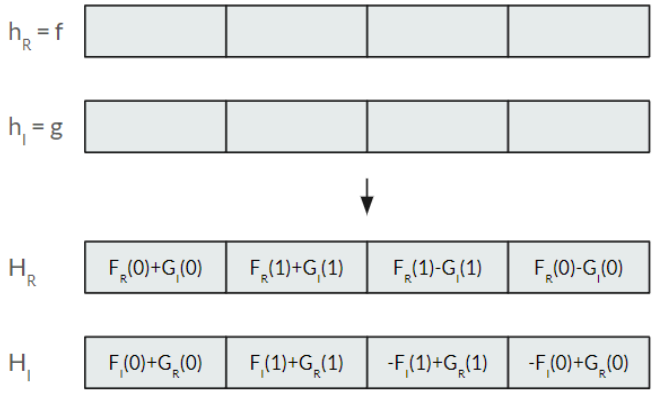
costisitor in special pentru semnale de dimensiuni mari. Pentru o solutie de programare dinamica definim partitiile dupa relatia de echivalenta $P_{i,p}[u] \equiv A[n]$, unde i este numarul iteratiei curente a algoritmului, $p = n \bmod N/2^i$ este numarul partitiei, si $u = n \operatorname{div} N/2^i$. La iteratia i, salvam in A[n] transformata Fourier a partitiei $P_{i,p}$ in u. Deoarece partitiile se injumatatesc la fiecare iteratie, la iteratia $\log(N)$ $A = P_{\log(N),0}$, deci algoritmul va calcula transformata Fourier a lui A.



4. Formula de recurenta pentru solutia DP

Rescriem formula de recurenta de la capitolul 2 pentru partitii
$P_{i,p}[u] = P_{i-1,p}[u] + W_{2^i}^u P_{i-1,p+N/2^i}[u]$
Daca o transformam pentru indexarea in matricea A, avem
$A[n] \leftarrow A[n + uN/2^i] + W_{2^i}^u A[n + uN/2^i + N/2^i]$
care este aplicata la fiecare iteratie a algoritmului. Doua observatii importante sunt ca operatiile trebuie rulate simultan pentru a evita potentiale race conditions, si ca indexarea in A se face modulo N.

5. Frequency Compression - Tangling

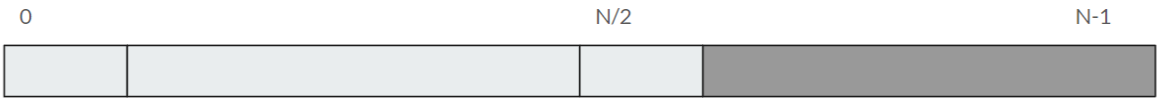


Deoarece pentru transformata Fourier se lucreaza cu numere complexe, valorile unui rand trebuie memorate in doi vectori, unul pentru partea reala si unul pentru partea imaginara. Inainte de prelucrare imaginea contine doar valori reale. Putem lua cate doua perechi de randuri din imagine f(x) si g(x) si sa definim h(x) = f(x) + jg(x). Transformata Fourier este liniara, deci H(x) = F(x) + jG(x). Deoarece pentru transformata Fourier valoarea de pe pozitia n este conjugata valorii de pe pozitia N-n, putem sa recuperam valorile lui F si G din H.

$$
\begin{aligned}
F(u)_R &= \tfrac{1}{2}(H(u)_R + H(N-u)_R) \\
F(u)_I &= \tfrac{1}{2}(H(u)_I - H(N-u)_I) \\
G(u)_R &= \tfrac{1}{2}(H(u)_I + H(N-u)_I) \\
G(u)_I &= -\tfrac{1}{2}(H(u)_R - H(N-u)_R)
\end{aligned}
$$

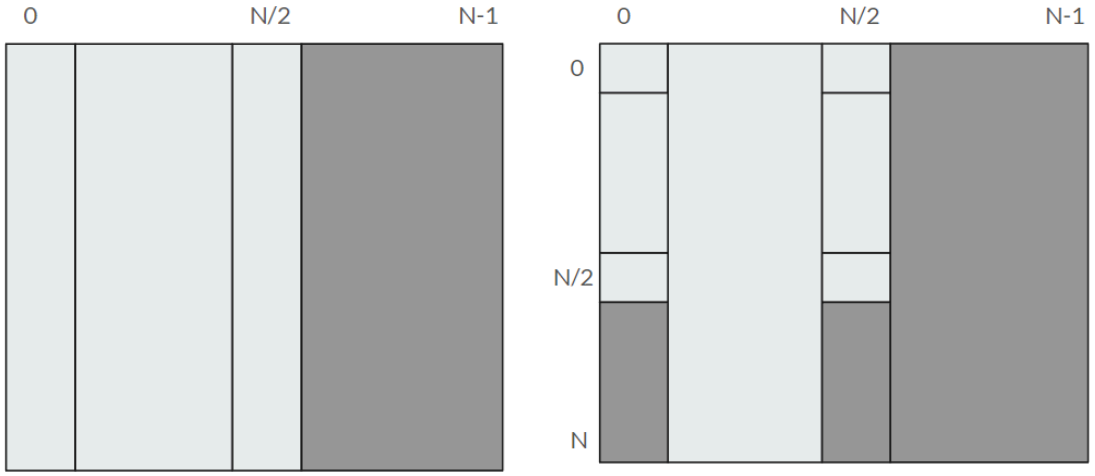
6. Frequency Compression - Packing

Pentru fiecare vector F, in loc sa fie salvate valorile transformatei Fourier a lui F in cate doua variabile, putem folosi proprietatea conjugatelor si sa salvam in prima jumatate partea reala si in a doua jumatate partea imaginara. Valorile de la indecsii 0 si N/2 sunt cojugatele proprii (valori reale), deci nu vor fi considerate pentru descompunere.



7. Rezultate

Dupa procesarea pe linii, coloanele 0 si N/2 vor avea valori reale proprii, iar coloanele din prima jumatate vor contine parti reale si cele din a doua jumatate vor contine valori imaginare.



Pentru calculul transformatei Fourier pe coloane vom imperechea coloanele 0 si N/2 si le calculam cu tangling si packing, iar pe restul le imperechem n cu N-n si rezultatul va fi transformata Fourier normala a unui semnal complex.

8. Implementare si rezultate

Pentru proiect am implementat algoritmul atat pe CPU cat si pe GPU. Implementarea pe CPU are o varianta single-threaded si una multi-threaded, dar care nu da rezultate considerabil mai bune.
Pentru interfatarea cu procesorul grafic am ales platforma CUDA de la nVidia. Perechile de linii sunt procesate fiecare pe cate un thread, iar thread-urile sunt organizate in blocuri intr-un grid. Daca dimensiunea imaginii depaseste numarul de thread-uri, acestea sunt impartite pe mai multe blocuri. Sincronizarea pentru evitarea de data races sau cresterea memoriei utilizate este facuta cu Grupuri cooperative.
Testele au fost rulate pe un procesor intel i7-8700K iar procesorul grafic a fost un nVidia Geforce GTX 1080 Ti. Pentru o imagine de 2048x2048 pixeli, algoritmul rulat pe CPU a avut viteza medie de 9.18 secunde, iar cel rulat pe GPU de 0.12.