

به نام خدا



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

برنامه نویسی چندهسته‌ای

دستور کار آزمایشگاه ۸

هدف از این آزمایش، انجام عملیات ریاضی Convolution به کمک gpu است.

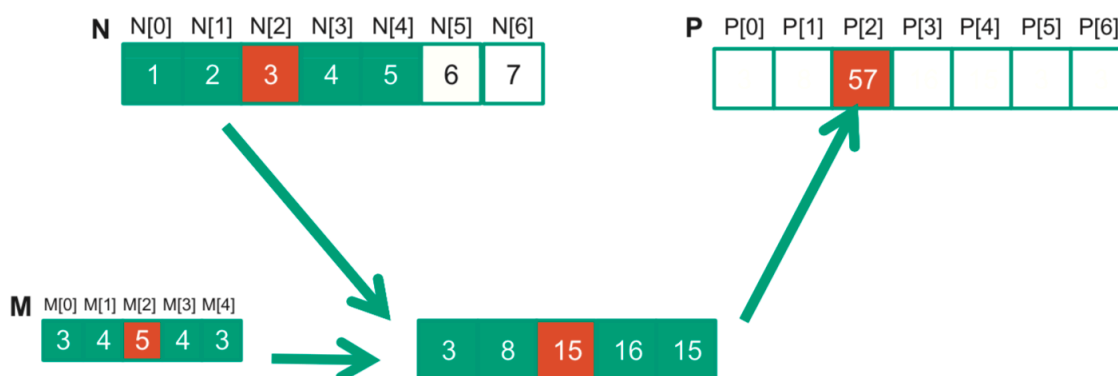
عملیات Discrete convolution به صورت زیر تعریف می‌شود:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

همان‌طور که از رابطه مشخص است این عملیات دو آرایه f و g را به عنوان ورودی گرفته و آرایه دیگری را به عنوان خروجی برمی‌گرداند. از آنجا که معمولاً یکی از این دو آرایه ورودی دارای اندازه محدودتری است می‌توان به مساله به این شکل نگاه کرد که هر درایه در آرایه خروجی، جمع وزن‌دار تعدادی از همسایگان خودش در آرایه بزرگتر است و آرایه کوچکتر معمولاً در نقش وزن‌ها ظاهر شده و اصطلاحاً به آن convolution kernel (mask) گفته می‌شود.

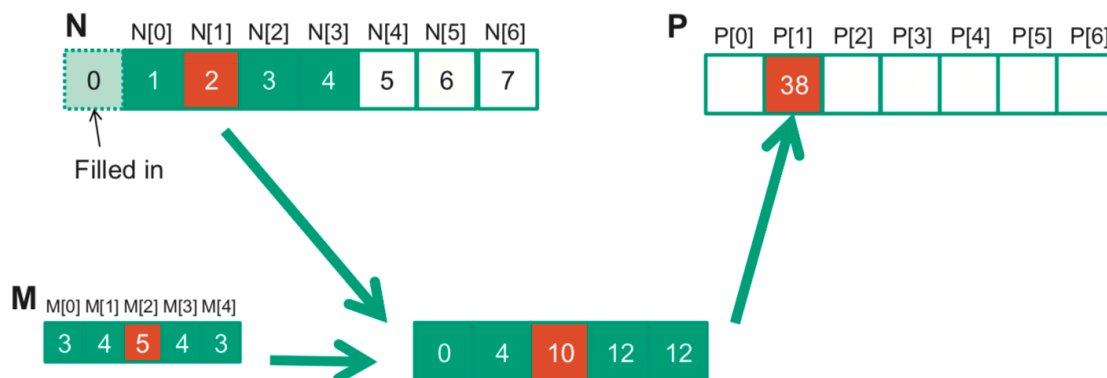
به این ترتیب می‌توان در عمل به مساله به این شکل نگاه کرد که به ما یک آرایه ورودی و یک آرایه از وزن‌ها داده می‌شود و ما در خروجی به ازای هر خانه جمع وزن‌داری از همسایگانش را محاسبه می‌کنیم.

در مثال زیر نحوه انجام عملیات برای به دست آوردن یکی از درایه‌های ماتریس خروجی P به کمک آرایه ورودی N با اندازه ۷ و نیز آرایه convolution kernel به نام M با اندازه ۵ را مشاهده می‌کنید :



$$\begin{aligned}
 P[2] &= N[0]*M[0] + N[1]*M[1] + N[2]*M[2] + N[3]*M[3] + N[4]*M[4] \\
 &= 1*3 + 2*4 + 3*5 + 4*4 + 5*3 \\
 &= 57
 \end{aligned}$$

هم چنین مطابق شکل زیر در صورتی که تعداد کافی المان در آرایه N وجود نداشت به تعداد لازم می توانیم از صفر استفاده نماییم:



$$\begin{aligned}
 P[1] &= 0 * M[0] + N[0]*M[1] + N[1]*M[2] + N[2]*M[3] + N[3]*M[4] \\
 &= 0 * 3 + 1*4 + 2*5 + 3*4 + 4*3 \\
 &= 38
 \end{aligned}$$

- این آرایه وزن‌ها (convolution kernel) در واقع همان تابع $f(-x)$ در رابطه بالا می‌باشد. این موضوع را می‌توانید توجیه کنید؟
- این عملیات کاربرد جدی در زمینه های کاری پردازش سیگنالی دارد. می‌توانید نمونه‌ای از کاربردهای آن را نام ببرید؟

در این آزمایش، می‌خواهیم انجام این عمل در gpu را به کمک چند گام بهینه کنیم.

۱- گام اول: پیاده سازی حالت ساده تابع کرنل

در قطعه کد زیر می‌توانید پیاده سازی حالت ساده‌ای از کرنل را مشاهده نمایید.

```
__global__ void convolution_1 (float *N, float *M, float *P, int Mask_Width, int Width) {  
    int i = blockIdx.x*blockDim.x + threadIdx.x;  
    float Pvalue = 0;  
    int N_start_point = i - (Mask_Width/2);  
    for (int j = 0; j < Mask_Width; j++) {  
        if (N_start_point + j >= 0 && N_start_point + j < Width) {  
            Pvalue += N[N_start_point + j]*M[j];  
        }  
    }  
    P[i] = Pvalue;  
}
```

- توضیح دهید طبق الگوریتم چگونه خانه‌های آرایه P پر می‌شوند. هر نخ چه کاری انجام می‌دهد؟
- این کرنل را برای آرایه N با اندازه ۱۰۰ میلیون (400MB) و آرایه M با اندازه ۱۰ (40B) تست کرده و زمان آن را گزارش نمایید.

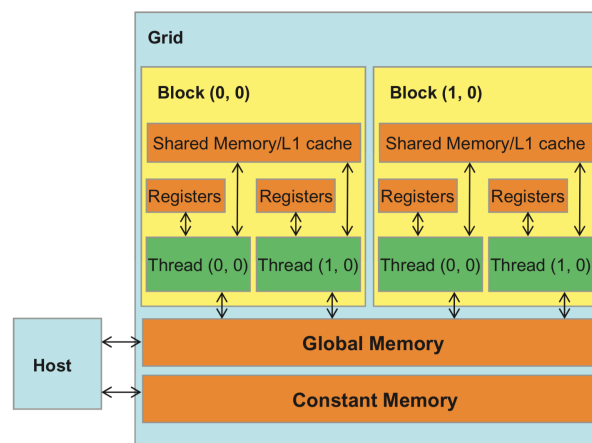
۲- گام دوم: استفاده از حافظه Constant

همان طور که پیش از این نیز گفته شد M به عنوان convolution kernel معمولاً دارای اندازه کوچک است و به ندرت از ۱۰ درایه بیشتر خواهد بود. همچنین این آرایه به کرات توسط هر کدام از نخ‌ها استفاده می‌شود. این دو ویژگی باعث می‌شوند تا این آرایه کاندید خوبی برای حضور در حافظه Constant باشد.

نحوه پیاده سازی کد کرنل، در قطعه کد زیر مشخص است :

```
__global__ void convolution_2(float *N, float *P, int Mask_Width, int Width) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    float Pvalue = 0;
    int N_start_point = i - (Mask_Width/2);
    for (int j = 0; j < Mask_Width; j++) {
        if (N_start_point + j >= 0 && N_start_point + j < Width) {
            Pvalue += N[N_start_point + j] * M[j];
        }
    }
    P[i] = Pvalue;
}
```

- با توجه به تصویر زیر که بیانگر یک شماتیک کلی از gpu و نحوه دسترسی آن به حافظه است می‌توانید توضیح بدهید چرا شرایط بالا، M را به گزینه خوبی برای حضور در حافظه Constant تبدیل می‌کند؟



- به کمک deviceQuery اندازه constant memory را در gpu ی که از آن استفاده می‌کنید گزارش کنید.

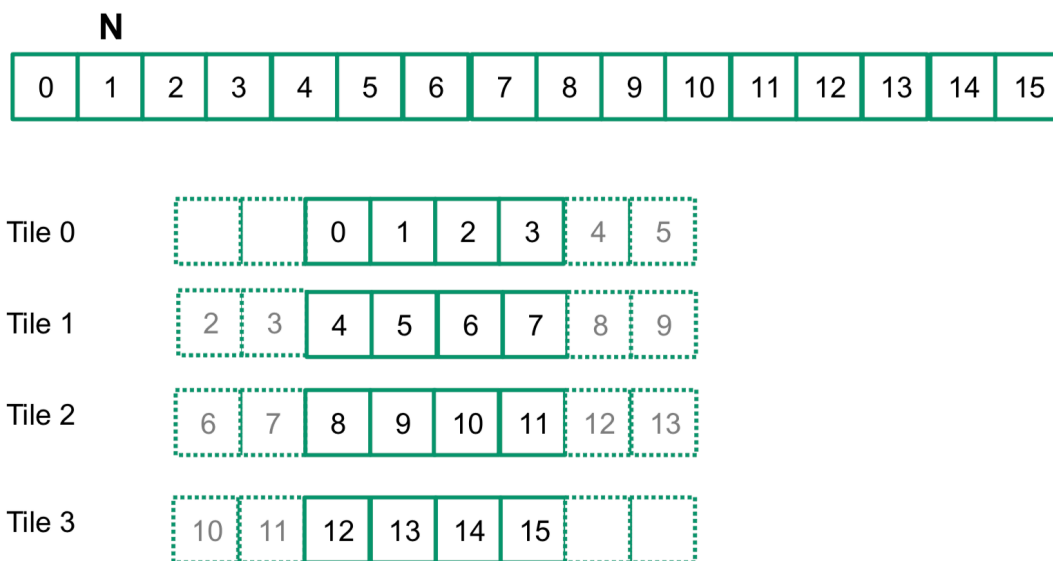
- به کمک Constant Memory و تابع `cudaMemcpyToSymbol(dest, src, size)` و توضیحات بالا کرنل و سایر بخش‌های موردنیاز کدتان را بازنویسی کرده و مجدداً زمان را برای اندازه‌های گفته شده در تست قبلی محاسبه کنید.

۳- گام سوم: استفاده از حافظه shared

در الگوریتم‌ها و سیستم‌های پردازش موازی، همواره یکی از مهم‌ترین گلوگاه‌های سیستم دسترسی به حافظه می‌باشد. به همین دلیل **cache**ها با سطوح متفاوتی در پردازنده‌ها طراحی شده‌اند تا بتوانند در شرایطی که دسترسی به حافظه‌ها محلی است به ما در بهینه‌کردن تعداد دسترسی‌ها به حافظه کمک کنند. در **gpu** به ازای هر **block** می‌توان مقداری از حافظه **shared memory** را برای استفاده محلی توسط نخ‌های آن **block** استفاده نمود. بنابراین همواره توجه داشته باشید که کنترل نحوه استفاده از این حافظه فقط بر عهده برنامه نویس است.

در این مساله دسترسی‌ها به حافظه **N** برای نخ‌های مجاور هر **block**، اشتراک دارد. طبق پیاده‌سازی قبلی، هر نخ به طور جداگانه به دنبال به دست آوردن تعدادی از خانه‌های حافظه **N** است که این حافظه‌ها توسط نخ‌های مجاور نیز از حافظه **global** در خواست می‌شوند. بنابراین با آوردن بخش‌های مورد نیاز از **N** به حافظه **shared** می‌توانیم بار زیادی را از دوش حافظه **global** برداریم.

برای مثال فرض کنید $|N|=16$ و $|M|=5$ است و ما می‌خواهیم به کمک **block**های ۴تایی این محاسبه را انجام دهیم. در شکل زیر می‌توانیم مشاهده کنیم هر کدام از این بلاک‌ها چه بخشی از حافظه را نیاز دارند.



قطعه کد روبرو نحوه انجام این کار در کرنل را نشان می دهد :

```
__global__ void convolution_3(float *N, float *P, int Mask_Width, int Width) {
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    __shared__ float N_ds[TILE_SIZE + MAX_MASK_WIDTH -1];
    int n = Mask_Width/2;

    //filling left side of N_ds
    int halo_index_left = (blockIdx.x - 1)*blockDim.x + threadIdx.x;
    if (threadIdx.x >= blockDim.x - n) {
        N_ds[threadIdx.x - (blockDim.x - n)] =
            (halo_index_left < 0) ? 0 : N[halo_index_left];
    }

    //filling main part of N_ds
    N_ds[n + threadIdx.x] = N[blockIdx.x*blockDim.x + threadIdx.x];

    //filling right side of N_ds
    int halo_index_right = (blockIdx.x + 1)*blockDim.x + threadIdx.x;
    if (threadIdx.x < n) {
        N_ds[n + blockDim.x + threadIdx.x] =
            (halo_index_right >= Width) ? 0 : N[halo_index_right];
    }

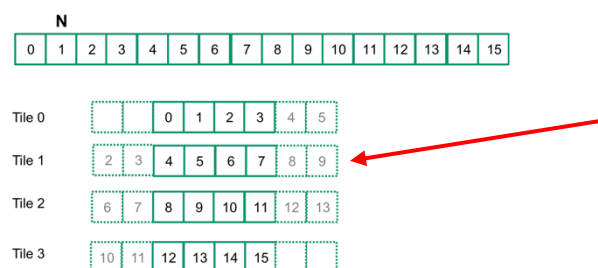
    __syncthreads();

    //calculating p ...
    float Pvalue = 0;
    for(int j = 0; j < Mask_Width; j++) {
        Pvalue += N_ds[threadIdx.x + j]*M[j];
    }

    P[i] = Pvalue;
}
```

- توضیح دهید چرا این روش تعداد دسترسی به حافظه را کمتر خواهد کرد؟
- در این کد از یک حافظه shared memory با نام N_ds برای هر block استفاده شده است. توجیه کنید چرا اندازه این حافظه $\text{TILE_SIZE} + \text{MAX_MASK_WIDTH} - 1$ می باشد؟

- نحوه پر شدن حافظه N_ds در این قطعه کد مشخص است. می توانید برای مثال ۱۶ تایی بالا و به ازای Tile1 که در شکل زیر مجددا مشخص شده است بگویید هر کدام از خانه های Tile1 توسط نخ شماره چند پر می شوند؟



- به کمک قطعه کد داده شده مجددا اندازه گیری زمان را برای اندازه های گفته شده در تست های قبل تکرار کنید.