Lập trình song song trên GPU

Hoàng Minh Thanh (18424062)

TH3 : Bộ nhớ trong CUDA

Bộ môn Công nghệ phần mềm

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

Contents

Vì máy tính cả nhân của em không có GPU nên bắt buộc em phải sử dụng Google Colab :

https://colab.research.google.com/drive/1c1hbh_t62g-YkdI1uLWnlj7pi7qNU6hF#scrollTo=cu3t0-tTgI0E

(Thầy có thể vào link Online để xem luồng chạy dễ hơn báo cáo)

Quá trình cài đặt

Chi tiết cài đặt trong file .cu và file Google Colab

Kết quả toàn bộ chương trình :

```
**********GPU info**********
Name: Tesla P100-PCIE-16GB
Compute capability: 6.0
Num SMs: 56
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 17071734784 bytes
CMEM: 65536 bytes
L2 cache: 4194304 bytes
SMEM / one SM: 65536 bytes
******************************

Image size (width x height): 512 x 512

Kernel 1, block size 32dxC32, grid size 16dxC16
Kernel time: 0.201568 ms
Error: 0.000703

Kernel 2, block size 32dxC32, grid size 16dxC16
Kernel time: 0.117152 ms
Error: 0.000703

Kernel 3, block size 32dxC32, grid size 16dxC16
Kernel time: 0.108640 ms
Error: 0.000703
```

1) Hàm tính blurImgKernel1

Kết quả cài đặt

Chương trình cài đặt

```cpp
// TODO
int idxR = blockIdx.y * blockDim.y + threadIdx.y;
int idxC = blockIdx.x * blockDim.x + threadIdx.x;
int idx = idxR * width + idxC;
int filterPadding = filterWidth / 2;

if (idxR < height && idxC < width)
{
    float3 outPixel = make_float3(0, 0, 0);
    for (int fR = 0; fR < filterWidth; fR++)
    {
        for (int fC = 0; fC < filterWidth; fC++)
        {
            float filterVal = filter[fR * filterWidth + fC];

            int inR = (idxR - filterPadding) + fR;
            int inC = (idxC - filterPadding) + fC;
            inR = min(height - 1, max(0, inR));
            inC = min(width - 1, max(0, inC));
            uchar3 inPixel = inPixels[inR * width + inC];

            outPixel.x += filterVal * inPixel.x;
            outPixel.y += filterVal * inPixel.y;
            outPixel.z += filterVal * inPixel.z;
        }
    }

    outPixels[idx] = make_uchar3(outPixel.x, outPixel.y, outPixel.z);
}
```

2) Hàm blurImgKernel2 sử dụng SMEM

```
int idxC = threadIdx.x + blockIdx.x * blockDim.x;
int idxR = threadIdx.y + blockIdx.y * blockDim.y;

int filterPadding = filterWidth / 2;
int shareBlockWidth = blockDim.x + filterWidth;

int inRow = idxR - filterPadding;
int inCol = idxC - filterPadding;
inRow = max(0, inRow);
inCol = max(0, inCol);
// SMEM copying : [-filterPadding, -filterPadding] -> [0, 0]
s_inPixels[threadIdx.y * shareBlockWidth + threadIdx.x] = inPixels[inRow * width + inCol];

// SMEM copying : [blockDim.x - filterPadding, -filterPadding] -> [0 + blockDim.x, 0]
if (floorf(threadIdx.x / filterWidth) == 0){
    inRow = idxR - filterPadding;
    inRow = max(0, inRow);
    inCol = (idxC - filterPadding) + blockDim.x;
    inCol = min(width - 1, max(0, inCol));
    s_inPixels[threadIdx.y * shareBlockWidth + threadIdx.x + blockDim.x] = inPixels[inRow * width + inCol];
}

// SMEM copying : [-filterPadding, blockDim.y - filterPadding] -> [0, 0 + blockDim.y]
if (floorf(threadIdx.y / filterWidth) == 0){
    inRow = (idxR - filterPadding) + blockDim.y;
    inRow = min(height - 1, max(0, inRow));
    inCol = idxC - filterPadding;
    inCol = max(0, inCol);

    s_inPixels[(threadIdx.y + blockDim.y) * shareBlockWidth + threadIdx.x] = inPixels[inRow * width + inCol];
}
```

```
// SMEM copying : [-filterPadding, blockDim.y - filterPadding] -> [0, 0 + blockDim.y]
if (floorf(threadIdx.y / filterWidth) == 0){
    inRow = (idxR - filterPadding) + blockDim.y;
    inRow = min(height - 1, max(0, inRow));
    inCol = idxC - filterPadding;
    inCol = max(0, inCol);

    s_inPixels[(threadIdx.y + blockDim.y) * shareBlockWidth + threadIdx.x] = inPixels[inRow * width + inCol];
}

// SMEM copying : [blockDim.x - filterPadding, blockDim.y - filterPadding] -> [0 + blockDim.x, 0 + blockDim.y]
if (floorf(threadIdx.y / filterWidth) == 0 && floorf(threadIdx.x / filterWidth) == 0){
    inRow = (idxR - filterPadding) + blockDim.y;
    inRow = min(height - 1, max(0, inRow));
    inCol = (idxC - filterPadding) + blockDim.x;
    inCol = min(width - 1, max(0, inCol));

    s_inPixels[(threadIdx.y + blockDim.y) * shareBlockWidth + blockDim.x + threadIdx.x] =
        inPixels[inRow * width + inCol];
}
__syncthreads();
```

```
// Convolution : [0, 0] -> [+filterPaddimg, +filterPadding]
if (idxC < width && idxR < height)
{
    float3 outPixel = make_float3(0, 0, 0);
    for (int filterR = 0; filterR < filterWidth; filterR++)
    {
        for (int filterC = 0; filterC < filterWidth; filterC++)
        {
            float filterVal = filter[filterR * filterWidth + filterC];
            int inPixelR = threadIdx.y + filterR;
            int inPixelC = threadIdx.x + filterC;
            uchar3 inPixel = s_inPixels[inPixelR * shareBlockWidth + inPixelC];

            outPixel.x += (filterVal * inPixel.x);
            outPixel.y += (filterVal * inPixel.y);
            outPixel.z += (filterVal * inPixel.z);
        }
    }

    outPixels[idxR * width + idxC] = make_uchar3(outPixel.x, outPixel.y, outPixel.z);
}
```

```
// TODO: call blurImgKernel2
size_t smem = (blockSize.x + (filterWidth >> 1) << 1) * (blockSize.y + (filterWidth >> 1) << 1) * sizeof(uchar3);
blurImgKernel2<<<gridSize, blockSize, smem>>>(d_inPixels, width, height, d_filter, filterWidth, d_outPixels);
```

3) Hàm blurImgKernel3 sử dụng CMEM

Hàm copy CMEM

```
// Convolution : [0, 0] -> [+filterPaddimg, +filterPadding]
if (idxC < width && idxR < height)
{
    float3 outPixel = make_float3(0, 0, 0);
    for (int filterR = 0; filterR < filterWidth; filterR++)
    {
        for (int filterC = 0; filterC < filterWidth; filterC++)
        {
            float filterVal = dc_filter[filterR * filterWidth + filterC];
            int inPixelR = threadIdx.y + filterR;
            int inPixelC = threadIdx.x + filterC;
            uchar3 inPixel = s_inPixels[inPixelR * shareBlockWidth + inPixelC];

            outPixel.x += (filterVal * inPixel.x);
            outPixel.y += (filterVal * inPixel.y);
            outPixel.z += (filterVal * inPixel.z);
        }
    }

    outPixels[idxR * width + idxC] = make_uchar3(outPixel.x, outPixel.y, outPixel.z);
}
```

Hàm gọi blurImgKernel3

```
// TODO: call blurImgKernel3      You, 10 days ago • add bt3
size_t smem = (blockSize.x + (filterWidth >> 1) << 1) * (blockSize.y + (filterWidth >> 1) << 1) * sizeof(uchar3);
blurImgKernel3<<<gridSize, blockSize, smem>>>(d_inPixels, width, height, filterWidth, d_outPixels);
```

4)Chạy các kích thước khác nhau:

32 x 32

```
**********GPU info**********
Name: Tesla P100-PCIE-16GB
Compute capability: 6.0
Num SMs: 56
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 17071734784 bytes
CMEM: 65536 bytes
L2 cache: 4194304 bytes
SMEM / one SM: 65536 bytes
*****************************

Image size (width x height): 512 x 512

Kernel 1, block size 32x32, grid size 16x16
Kernel time: 0.213760 ms
Error: 0.000703

Kernel 2, block size 32x32, grid size 16x16
Kernel time: 0.132000 ms
Error: 0.000703

Kernel 3, block size 32x32, grid size 16x16
Kernel time: 0.125984 ms
Error: 0.000703
```

32 x 16

```
**********GPU info**********
Name: Tesla P100-PCIE-16GB
Compute capability: 6.0
Num SMs: 56
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 17071734784 bytes
CMEM: 65536 bytes
L2 cache: 4194304 bytes
SMEM / one SM: 65536 bytes
*****************************

Image size (width x height): 512 x 512

Kernel 1, block size 32x16, grid size 16x32
Kernel time: 0.206208 ms
Error: 0.000703

Kernel 2, block size 32x16, grid size 16x32
Kernel time: 0.121088 ms
Error: 0.000703

Kernel 3, block size 32x16, grid size 16x32
Kernel time: 0.113376 ms
Error: 0.000703
```

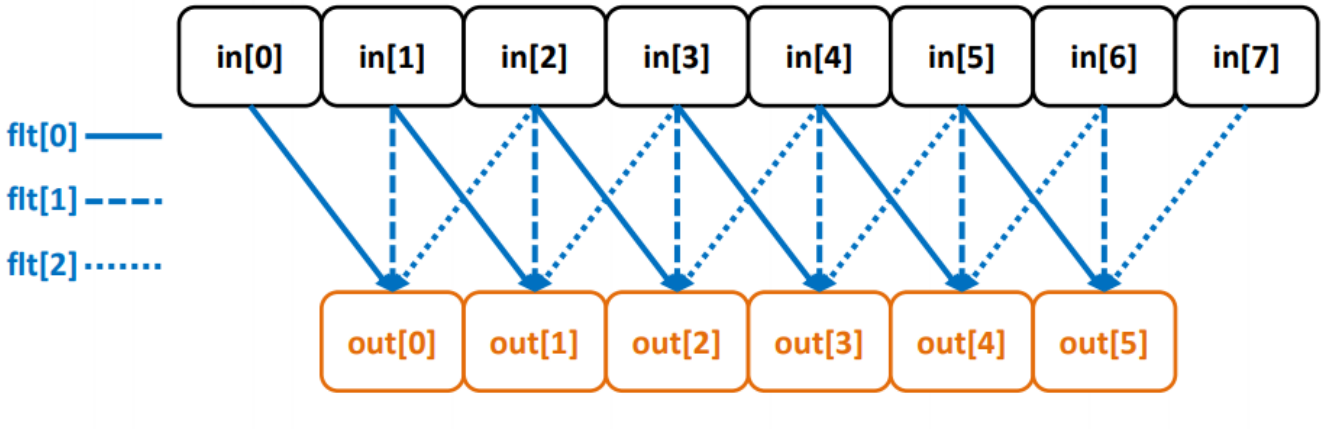
8x8

```
**********GPU info**********
Name: Tesla P100-PCIE-16GB
Compute capability: 6.0
Num SMs: 56
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 17071734784 bytes
CMEM: 65536 bytes
L2 cache: 4194304 bytes
SMEM / one SM: 65536 bytes
**************************

Image size (width x height): 512 x 512

Kernel 1, block size 8x8, grid size 64x64
Kernel time: 0.315136 ms
Error: 0.000703

Kernel 2, block size 8x8, grid size 64x64
Kernel time: 0.114176 ms
Error: 0.000703

Kernel 3, block size 8x8, grid size 64x64
Kernel time: 0.107424 ms
Error: 0.000703
```

5) Giải thích kết quả :

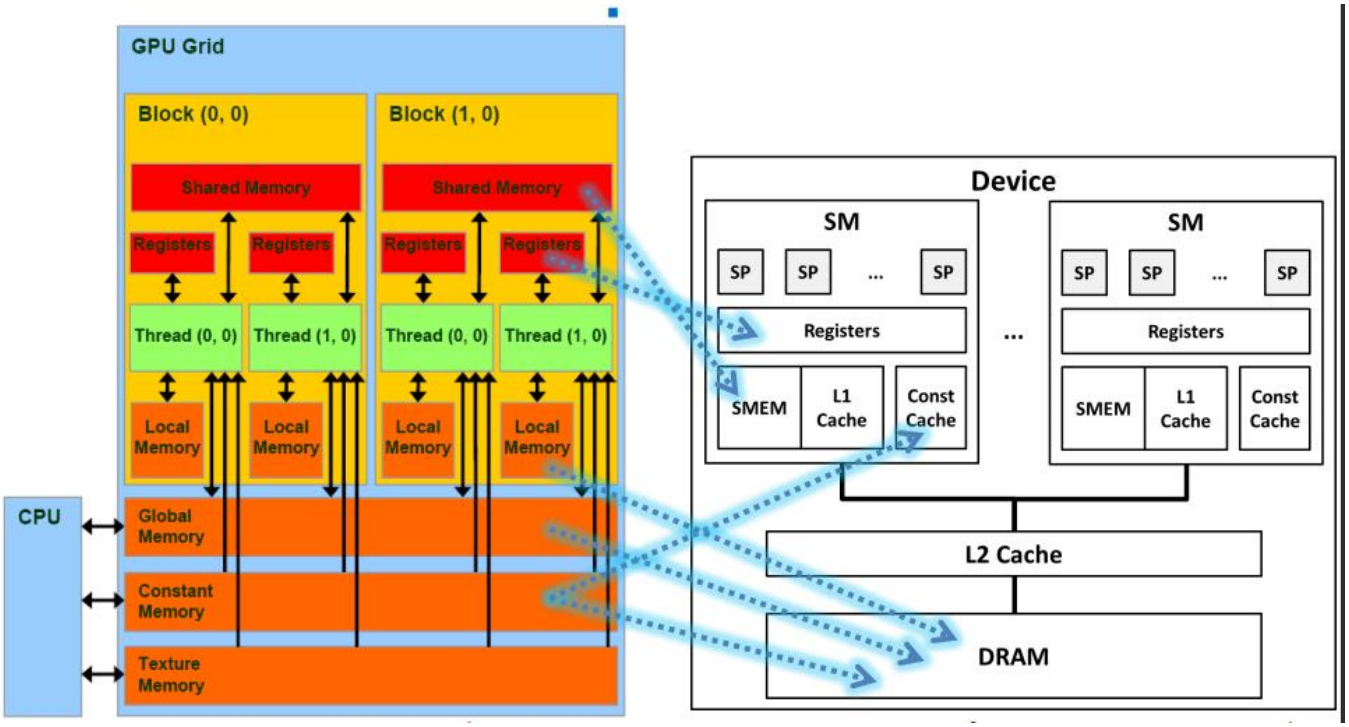
Giải thích tại sao kết quả lại như vậy (tại sao dùng SMEM lại chạy nhanh hơn so với không dùng, tại sao dùng).

1. Sử dụng SMEM sẽ nhanh hơn so với không dùng:



Ví dụ với convolution 1 chiều

Vì khi copy dữ liệu từ GMEM sang SMEM thì với mỗi lần tính convolution thì thay vì các thread phải lên GMEM (nơi có tốc độ truy xuất chậm) thì các thread sẽ chạy lên SMEM là bộ nhớ chia sẻ trong mỗi block thì sẽ có kết quả truy xuất nhanh hơn và không phải chạy đến GMEM để lấy dữ liệu nhiều lần.



2. CMEM lại chạy nhanh hơn so với không dùng

Vì CMEM là bộ nhớ hằng, ở trong bộ nhớ hằng sẽ không thay đổi giá trị khi đã gán là bộ nhớ được tối ưu để lưu các giá trị không thay đổi, và sẽ được lưu dưới một địa chỉ trong quá trình tính toán. Vì vậy nếu các thread trong cùng một wrap sẽ gọi đến cùng một địa chỉ, và chỉ tốn 1 lần đọc duy nhất. Sau đó

địa chỉ đó sẽ được broadcast cho các thread trong một wrap. Nên từ đó tối ưu tốc độ tính toán. Nên có tốc độ chạy nhanh hơn so với không dùng.