# CSED490C Lab Assignment 4

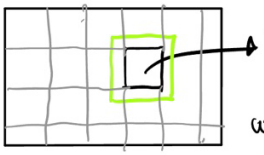
20232241 강두경

**1. How many floating operations are being performed by your convolution kernel?**

- 스레드당 `Mask_width * Mask_width * channels` = 75회
- 바운더리 내의 모든 스레드가 아웃풋 계산에 참여한다. 총 스레드는 Width * Height 개
- 따라서, `width * height * 75`회의 floating operations가 일어난다.

**2. How many global memory reads are being performed by your convolution kernel?**

- constant memory 사용으로 간주하는 M에 대한 접근을 제외하고, 인풋 이미지인 I에 대한 접근만 계산한다.
- 스레드 블럭당 `(T + Mask_width − 1) * (T + Mask_width − 1)` 사이즈의 shared memory에 I의 내용을 복사한다.
- 그러나, 바운더리의 경우 hallow cell로 0으로 남겨두기 때문에 이 경우 global read하지 않는다.
- $\left(W+4\left\lfloor\frac{W-1}{T}\right\rfloor\right)\left(H+4\left\lfloor\frac{H-1}{T}\right\rfloor\right)$ 만큼의 Global read가 발생한다.



$T_{ij}$ tile에 대해

$w_{ij}, h_{ij}$ : Shared memory의 크기

이 때 W는 $j$에만 의존하여, H는 $i$에만 의존한다. $\Rightarrow W_j H_i$

$\therefore \sum \text{Global Read} = \sum_i \sum_j w_j h_i = \sum w_j \sum h_i$

i) $w_j = \begin{cases} T+2; & j=1 \text{ (첫 열)} \\ W-\left(\left\lceil\frac{W}{T}\right\rceil-1\right)T+2; & j=\left\lceil\frac{W}{T}\right\rceil \text{ (마지막 열)} \\ T+4; & \text{else} \end{cases}$

$\therefore \sum w_j = W-\left\lceil\frac{W}{T}\right\rceil T+2T+4+(T+4)\left(\left\lceil\frac{W}{T}\right\rceil-2\right)$

$= W-\left\lceil\frac{W}{T}\right\rceil T+2T+4+T\left\lceil\frac{W}{T}\right\rceil-2T+4\left\lceil\frac{W}{T}\right\rceil-8$

$= W+4\left(\left\lceil\frac{W}{T}\right\rceil-1\right) = W+4\left\lfloor\frac{W-1}{T}\right\rfloor$

WLOG. $\sum h_i = H+4\left\lfloor\frac{H-1}{T}\right\rfloor$, $\therefore \sum HG.read = \left(W+4\left\lfloor\frac{W-1}{T}\right\rfloor\right)\left(H+4\left\lfloor\frac{H-1}{T}\right\rfloor\right)$

**3. How many global memory writes are being performed by your convolution kernel?**

- 바운더리 내의 모든 스레드가 하나의 output image에 쓴다.
- 따라서 global write는 WH 만큼 발생한다.

**4. How much time is spent as an overhead cost for using the GPU for computation? Consider all code executed within your host function with the exception of the kernel itself, as overhead. How does the overhead scale with the size of the input?**

- 오버헤드는 W, H 의 크기가 커질수록 점점 WH에 비례하는 양상으로 커진다.
- Problem 7에서 overhead를 계산한 결과, W, H이 커질수록 점점 WH에 비례하게 커진다는 것을 알 수 있다.

**5. What do you think happens as you increase the mask size (say to 1024) while you set the block dimensions to 16x16? What do you end up spending most of your time doing? Does that put other constraints on the way you'd write your algorithm?**

- 최대 1024 x 1024 사이즈의 constant memory를 확보해야 하며, constant memory로 복사하는 시간이 지연된다.
- 커널에서 한 스레드당 평균 global read는 $\frac{(T+M-1)^2}{T^2}$ 만큼 증가하며, 이 경우 최대 4096개의 shared memory에 접근해야 한다.
- 커널에서 컨볼루션을 계산할 때 $M^2$만큼의 iteration을 돌게 된다.
- 대부분은 데이터 이동에 시간을 차지하게 된다.
- M이 T보다 어느 정도 이상 커질 경우 스레드 병렬성을 증가시키기 위해 input tiling을 사용하여 지연 시간을 단축할 수 있다. 대신 더 많은 스레드를 이용한다.

**6. Your version of template.cu.**

```
// 중략
__global__ void convolution(float* I, const float* __restrict__ M, float* P, int channels, int width, int height)
{
  int row = blockIdx.y * blockDim.y + threadIdx.y;
  int col = blockIdx.x * blockDim.x + threadIdx.x;
  int ti = threadIdx.y * TILE_WIDTH + threadIdx.x;

  __shared__ float Ns[w][w][3];

  while (ti < w * w) {
    int Si = ti / w, Sj = ti % w;
    int Ii = blockIdx.y * blockDim.y + Si - Mask_radius;
    int Ij = blockIdx.x * blockDim.x + Sj - Mask_radius;
    if (Ii >= 0 && Ii < height && Ij >= 0 && Ij < width) {
      for (int m = 0; m < channels; m++) {
        Ns[Si][Sj][m] = I[(Ii * width + Ij) * channels + m];
      }
    } else {
      for (int m = 0; m < channels; m++) {
        Ns[Si][Sj][m] = 0.0f;
      }
    }
    ti += TILE_WIDTH * TILE_WIDTH;
  }
  __syncthreads();

  float out[3] = { 0.0f, 0.0f, 0.0f };

  if (row < height && col < width) {
    for (int i = 0; i < Mask_width; i++) {
      for (int j = 0; j < Mask_width; j++) {
        for (int m = 0; m < channels; m++) {
          out[m] += Ns[threadIdx.y + i][threadIdx.x + j][m] * M[i * Mask_width + j];
        }
      }
    }
  }

  if (row < height && col < width) {
    for (int m = 0; m < channels; m++) {
      P[(row * width + col) * channels + m] = out[m];
    }
  }
}

int main(int argc, char *argv[]) {
  // 중략

  gpuTKTime_start(GPU, "Doing GPU memory allocation");
  //@@ INSERT CODE HERE
  size_t imageSize = sizeof(float) * imageWidth * imageHeight * imageChannels;
  cudaMalloc((void**)&deviceInputImageData, imageSize);
  cudaMalloc((void**)&deviceOutputImageData, imageSize);
  cudaMalloc((void**)&deviceMaskData, Mask_width * Mask_width * sizeof(float));

  gpuTKTime_stop(GPU, "Doing GPU memory allocation");

  gpuTKTime_start(Copy, "Copying data to the GPU");
  //@@ INSERT CODE HERE
  cudaMemcpy(deviceInputImageData, hostInputImageData, imageSize, cudaMemcpyHostToDevice);
  cudaMemcpy(deviceMaskData, hostMaskData, Mask_width * Mask_width * sizeof(float), cudaMemcpyHostToDevice);
  gpuTKTime_stop(Copy, "Copying data to the GPU");

  gpuTKTime_start(Compute, "Doing the computation on the GPU");
  //@@ INSERT CODE HERE
  dim3 dimGrid((imageWidth + TILE_WIDTH - 1) / TILE_WIDTH, (imageHeight  + TILE_WIDTH - 1) / TILE_WIDTH);
  dim3 dimBlock(TILE_WIDTH, TILE_WIDTH);
  convolution<<<dimGrid, dimBlock>>>(deviceInputImageData, deviceMaskData,
                                     deviceOutputImageData, imageChannels,
                                     imageWidth, imageHeight);
  gpuTKTime_stop(Compute, "Doing the computation on the GPU");

  gpuTKTime_start(Copy, "Copying data from the GPU");
  //@@ INSERT CODE HERE
```

```
  cudaMemcpy(hostOutputImageData, deviceOutputImageData,
             imageWidth * imageHeight * imageChannels * sizeof(float),
             cudaMemcpyDeviceToHost);
  gpuTKTime_stop(Copy, "Copying data from the GPU");

  gpuTKTime_stop(GPU, "Doing GPU Computation (memory + compute)");

  gpuTKSolution(arg, outputImage);

  //@@ Insert code here
  cudaFree(deviceInputImageData);
  cudaFree(deviceOutputImageData);
  cudaFree(deviceMaskData);

  free(hostMaskData);
  gpuTKImage_delete(outputImage);
  gpuTKImage_delete(inputImage);

  return 0;
}
```

**7. The result as a table/graph of kernel execution times for different input data, with the system information where you performed your evaluation. Run your implementation with the input generated by the provided dataset generator. For time measurement, use gpuTKTime start and gpuTKTime stop functions (You can find details in libgputk/README.md).**

| No. | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **W** | 64 | 64 | 256 | 256 | 1024 | 4096 | 8192 |
| **H** | 64 | 128 | 256 | 512 | 1024 | 2048 | 8192 |
| **Alloc** | 0.154188 | 0.155781 | 0.161432 | 0.252895 | 0.386872 | 0.613609 | 1.64015 |
| **H->D** | 0.048802 | 0.049959 | 0.134238 | 0.196816 | 1.04515 | 9.50605 | 70.0164 |
| **Kernel** | 0.02556 | 0.027052 | 0.029998 | 0.038273 | 0.134115 | 0.85141 | 6.57492 |
| **D->H** | 0.045137 | 0.07555 | 0.434885 | 0.92726 | 6.32087 | 51.0272 | 407.915 |
| **Free** | 0.35473 | 0.390252 | 0.848416 | 1.51 | 7.9915 | 62.1418 | 486.292 |
| **Overhead** ($\sum T - T_{kernel}$) | 0.602857 | 0.671542 | 1.578971 | 2.886971 | 15.744392 | 123.288659 | 965.86355 |