Lập trình song song trên GPU

Hoàng Minh Thanh (18424062)

BT5 : Stream 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/1tSSxxtJMB9HVTExi8xER_ptkBpg3s-oZ

(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 của toàn bộ các phương pháp :

```
############ GPU Properties ############
Device 0: Tesla P100-PCIE-16GB
Kích thước mảng : 134217728
Kích thước : <<<Grid (512, 1), Block (262144, 1)>>>
ID| Kernel                    |Time             | Sum result
1 | sumOnHost                       | 0.727061 sec
2 | sumOnDeviceWithoutStream            | 0.496212 sec    | 1
3 | sumOnDevice2Stream                  | 0.000177 sec    | 1
4 | sumOnDevice3Stream                  | 0.000144 sec    | 1
5 | sumOnDevice3StreamUseEvent          | 0.000113 sec    | 1
```

Block và Grid được cấu hình theo **1D grid** và **1D block**

1) Hàm tính trung bình 2 vector trên host

```
void sumOnHost(int *in1, int *in2, int *out, int size){
    for (int i = 0; i < size; i++){
        out[i] = (in1[i] + in2[i])/2;
    }
}
```

```
// ############ 1. sumOnHost #############
iStart = seconds();
sumOnHost(A, B, hostRef, size);
iElaps = seconds() - iStart;
printf("1 | sumOnHost \t\t\t\t| %f sec\t\n", iElaps);
```

2) Hàm tính trung bình 2 vector trên device không Stream :

```
// Hàm thực hiện reduce trên CPU
__global__ void sumOnDevice(int *in1, int *in2, int *out, int size){
    unsigned int idx = threadIdx.x + blockDim.x * blockIdx.x;

    if (idx < size){
        out[idx] = (in1[idx] + in2[idx])/2;
    }
}
```

```
// ############# 2. sumOnDeviceWithoutStream #############
// Cấp phát bộ nhớ trên device (GPU)
int *d_A = NULL, *d_B = NULL, *d_C = NULL;
CHECK(cudaMalloc((int**)&d_A, nBytes));
CHECK(cudaMalloc((int**)&d_B, nBytes));
CHECK(cudaMalloc((int**)&d_C, nBytes));

iStart = seconds();

// Copy inputs to device
cudaMemcpy(d_A, A, nBytes, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, B, nBytes, cudaMemcpyHostToDevice);

sumOnDevice<<<gridSize, blockSize, 0, 0>>>(d_A, d_B, d_C, size);
cudaMemcpy(gpuRef, d_C, nBytes, cudaMemcpyDeviceToHost);
iElaps = seconds() - iStart;
int isTrue = checkResult(hostRef, gpuRef, size);
printf("2 | sumOnDeviceWithoutStream \t\t| %f sec\t| %d\t\n", iElaps, isTrue);
```

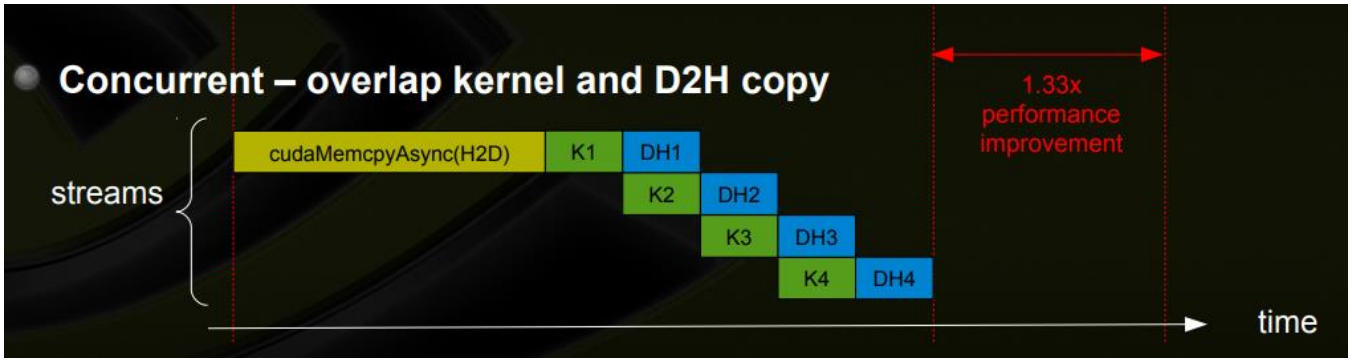
3) Hàm tính trung bình 2 vector trên Device sử dụng 2 Stream:

```
cudaStream_t stream1[nStream];
for (int i = 0; i < nStream; i++)
    cudaStreamCreate(&stream1[i]);

int iSize = size/nStream;
int iBytes = iSize * sizeof(int);

iStart = seconds();

for (int i = 0; i < nStream; ++i)
{
    int ioffset = i * iSize;
    CHECK(cudaMemcpyAsync(&d_A[ioffset], &h_A[ioffset], iBytes, cudaMemcpyHostToDevice, stream1[i]));
    CHECK(cudaMemcpyAsync(&d_B[ioffset], &h_B[ioffset], iBytes, cudaMemcpyHostToDevice, stream1[i]));
    sumOnDevice<<<gridSize, blockSize, 0, stream1[i]>>>(&d_A[ioffset], &d_B[ioffset], &d_C[ioffset], iSize);
    CHECK(cudaMemcpyAsync(&h_gpuRef[ioffset], &d_C[ioffset], iBytes, cudaMemcpyDeviceToHost, stream1[i]));
}

iElaps = seconds() - iStart;
isTrue = checkResult(hostRef, gpuRef, size);
printf("3 | sumOnDevice2Stream \t\t\t| %f sec\t| %d\t\n", iElaps, isTrue);
for (int i = 0; i <nStream; i++)
    cudaStreamDestroy(stream1[i]);
```

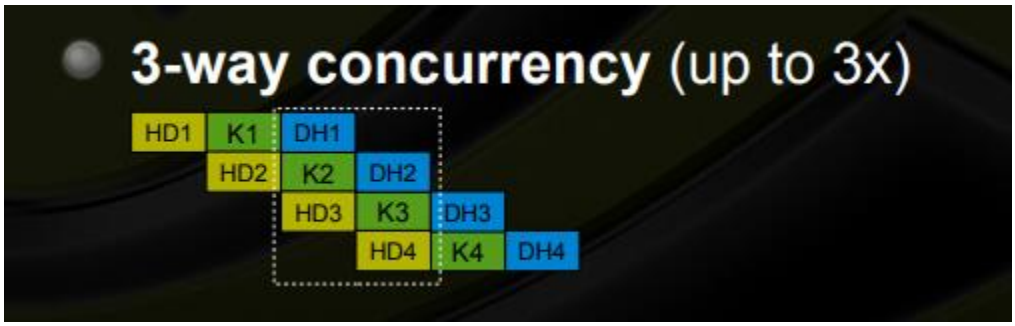
Ta thực hiện tính đồng bộ cả 3 quá trình Copy H2D, thực thi kernel và Copy D2H

Có thể trình bày khác bằng cách Copy H2D trước

Rồi sau đó mới thực thi Kernel và Copy D2H giống như hình sau



nhưng ở đây với 2 Stream thì em sử dụng phiên bản tối ưu nhất



4) Hàm tính trung bình 2 vector sử dụng 3 Stream :

```
// Tạo stream
cudaStream_t stream2[nStream];
for (int i = 0; i < nStream; i++)
    cudaStreamCreate(&stream2[i]);

iSize = size/nStream;
iBytes = iSize * sizeof(int);

iStart = seconds();

for (int i = 0; i < nStream; ++i){
    int ioffset = i * iSize;
    CHECK(cudaMemcpyAsync(&d_A[ioffset], &h_A[ioffset], iBytes, cudaMemcpyHostToDevice, stream2[i]));
    CHECK(cudaMemcpyAsync(&d_B[ioffset], &h_B[ioffset], iBytes, cudaMemcpyHostToDevice, stream2[i]));
    sumOnDevice<<<gridSize, blockSize, 0, stream2[i]>>>(&d_A[ioffset], &d_B[ioffset], &d_C[ioffset], iSize);
    CHECK(cudaMemcpyAsync(&h_gpuRef[ioffset], &d_C[ioffset], iBytes, cudaMemcpyDeviceToHost, stream2[i]));
}

iElaps = seconds() - iStart;
isTrue = checkResult(hostRef, gpuRef, size);
printf("4 | sumOnDevice3Stream \t\t\t| %f sec\t| %d\t\n", iElaps, isTrue);
for (int i = 0; i <nStream; i++)
    cudaStreamDestroy(stream2[i]);
```

Ta thực hiện tương tự với cả 3 tác vụ như trên nhưng thay bằng 3 stream

5) Hàm tính trung bình 2 vector sử dụng Stream với Issuse Order:

```
// Tạo stream
cudaStream_t stream3[nStream];
for (int i = 0; i < nStream; i++)
    cudaStreamCreate(&stream3[i]);

cudaEvent_t *kernelEvent;
kernelEvent = (cudaEvent_t *) malloc(nStream * sizeof(cudaEvent_t));

for (int i = 0; i < nStream; i++){
    CHECK(cudaEventCreateWithFlags(&(kernelEvent[i]), cudaEventDisableTiming));
}

iSize = size/nStream;
iBytes = iSize * sizeof(int);

iStart = seconds();

for (int i = 0; i < nStream; ++i){
    int ioffset = i * iSize;
    CHECK(cudaMemcpyAsync(&d_A[ioffset], &h_A[ioffset], iBytes, cudaMemcpyHostToDevice, stream3[i]));
    CHECK(cudaMemcpyAsync(&d_B[ioffset], &h_B[ioffset], iBytes, cudaMemcpyHostToDevice, stream3[i]));
    sumOnDevice<<<gridSize, blockSize, 0, stream3[i]>>>(&d_A[ioffset], &d_B[ioffset], &d_C[ioffset], iSize);
    CHECK(cudaMemcpyAsync(&h_gpuRef[ioffset], &d_C[ioffset], iBytes, cudaMemcpyDeviceToHost, stream3[i]));

    CHECK(cudaEventRecord(kernelEvent[i], stream3[i]));
    CHECK(cudaStreamWaitEvent(stream3[nStream - 1], kernelEvent[i], 0));
}

iElaps = seconds() - iStart;
isTrue = checkResult(hostRef, gpuRef, size);
printf("5 | sumOnDevice3StreamUseEvent \t\t| %f sec\t| %d\t\n", iElaps, isTrue);
```

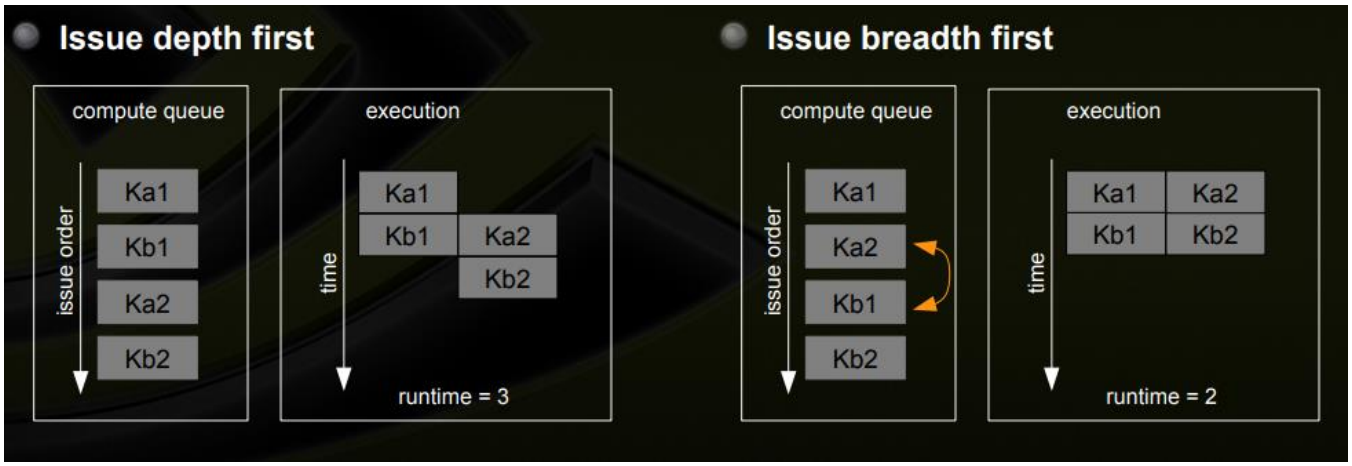
Tương tự như phần trên tuy nhiên mỗi lần thực thi kernel ta sẽ có 1 event đồng bộ để quá trình thực thi nhanh hơn

II. Báo cáo và rút ra nhận xét, kết luận

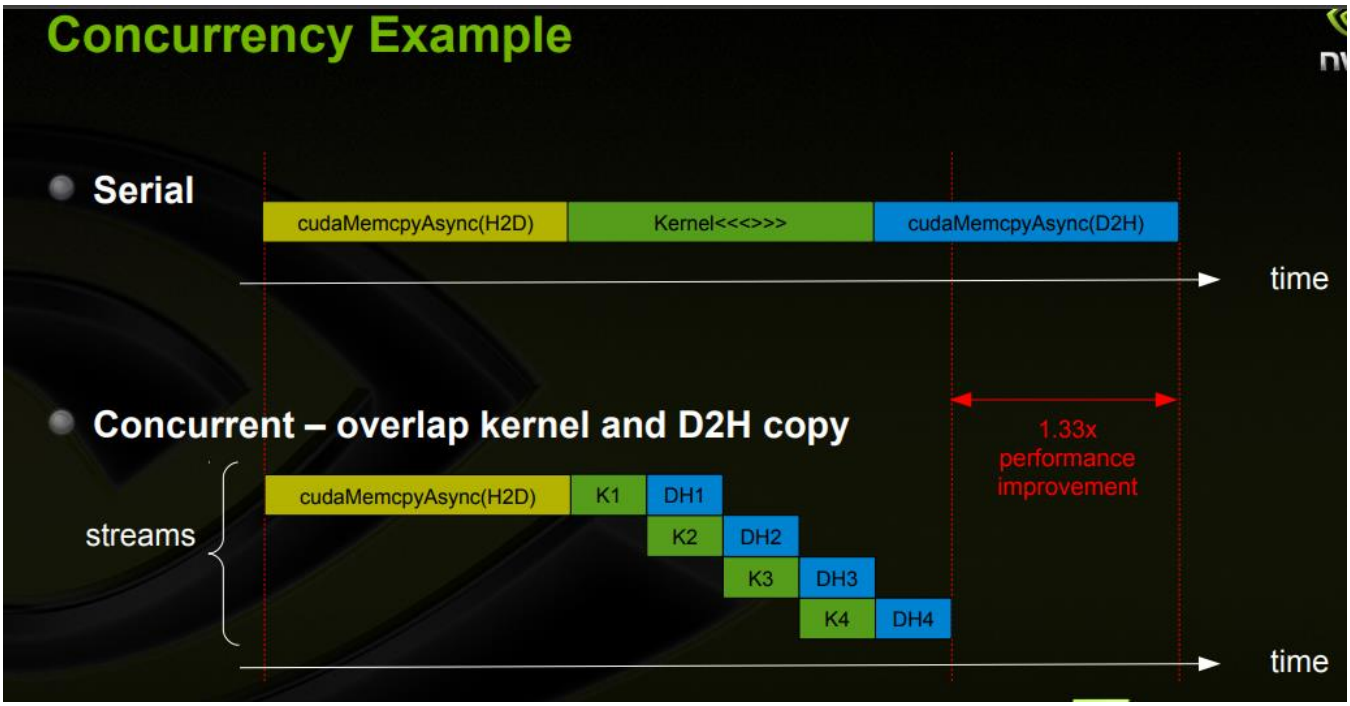
Có tham khảo mã nguồn tại : https://github.com/hmthanh/ProfessionalCUDACProgramming

(Repo của em)

- Muốn đồng bộ phải sử dụng bộ nhớ Pinned
- Để tăng tốc quá trình tính toán cần sử dụng các cudaEvent để thay đổi thứ tự Issuse Order giúp đảm bảo quá trình thực thi được sắp xếp theo các kích thước tối ưu nhất



- Sử dụng cudaEvent_t, và cudaStream_t tương tư phải khởi tạo, cấp phát, cấu hình và hủy
- Chia các khối lượng công việc lớn vào các Stream để các Stream xử lý đồng thời từng nhóm quá trình Copy H2D, thực thi kernel, và Copy D2H.



- Ta có thể tối ưu hơn nữa bằng cách chia vector với các kích thước có sẵn trước rồi thực thi các kernel đồng thời với nhau.