Lập trình song song trên GPU
HW3: Các loại bộ nhớ trong CUDA

Nên nhớ mục tiêu chính ở đây là học, học một cách chân thật. Bạn có thể thảo luận ý tưởng với bạn khác, nhưng bài làm phải là của bạn, dựa trên sự hiểu thật sự của bạn. Nếu vi phạm thì sẽ bị 0 điểm cho toàn bộ môn học.

Trong môn học, để thống nhất, tất cả các bạn (cho dù máy bạn có GPU) đều phải dùng Google Colab để biên dịch và chạy code (khi chấm Thầy cũng sẽ dùng Colab để chấm). Với mỗi bài tập, bạn thường sẽ phải nộp:

1) **File code** (file .cu)
2) **File báo cáo** là file notebook (file .ipynb) của Colab (nếu bạn nào biết Jupyter Notebook thì bạn thấy Jupter Notebook và Colab khá tương tự nhau, nhưng 2 cái này hiện chưa tương thích 100% với nhau: file .ipynb viết bằng Jupyter Notebook có thể sẽ bị mất một số cell khi mở bằng Colab và ngược lại). File này sẽ chứa các kết quả chạy. Ngoài ra, một số bài tập có phần viết (ví dụ, yêu cầu bạn nhận xét về kết quả chạy), và bạn sẽ viết trong file notebook của Colab luôn. Colab có 2 loại cell: code cell và text cell. Ở code cell, bạn có thể chạy các câu lệnh giống như trên terminal của Linux bằng cách thêm dấu ! ở đầu. Ở text cell, bạn có thể soạn thảo văn bản theo cú pháp của Markdown (rất dễ học, bạn có thể xem ở đây); như vậy, bạn sẽ dùng text cell để làm phần viết trong các bài tập. Bạn có thể xem về cách thêm code/text cell và các thao tác cơ bản ở đây, mục "Cells" (đừng đi qua mục "Working with Python"). Một phím tắt ưa thích của mình khi làm với Colab là ctrl+shift+p để có thể search các câu lệnh của Colab (nếu câu lệnh có phím tắt thì bên cạnh kết quả search sẽ có phím tắt). File notebook trên Colab sẽ được lưu vào Google Drive của bạn; bạn cũng có thể download trực tiếp xuống bằng cách ấn ctrl+shift+p, rồi gõ "download .ipynb".

Đề bài

Câu 1 (7đ)

Áp dụng hiểu biết về các loại bộ nhớ trong CUDA để tối ưu hóa chương trình làm mờ ảnh RGB (cách làm mờ giống như ở HW1).

Các bạn có thể tham khảo file Demo/05-MemoryDemo.cu trong Drive để xem cách xử lý với Convolution1D, từ đó áp dụng lên Convolution2D trong bài này.

Code (5 đ)

Mình có đính kèm file ảnh đầu vào in.pnm (trong Windows, bạn có thể xem file *.pnm bằng chương trình IrfanView). Mình cũng đã viết sẵn khung chương trình trong file HW3_P1.cu đính kèm. Bạn sẽ cần phải viết code ở những chỗ mình để // TODO:

- Định nghĩa hàm kernel blurImgKernel1 – hàm kernel làm mờ ảnh ở HW1. Bạn nào đã làm HW1 rồi thì chỉ cần copy-paste (code lại một lần nữa cũng tốt); bạn nào chưa làm HW1 thì đầu tiên bạn phải hoàn

thành hàm kernel này. Đây là hàm kernel cơ bản nhất; nếu bạn chưa cài đặt được hàm kernel này thì không thể qua hàm kernel 2 và 3 (sẽ được trình bày ở dưới) được.

- Gọi hàm blurImgKernel1.

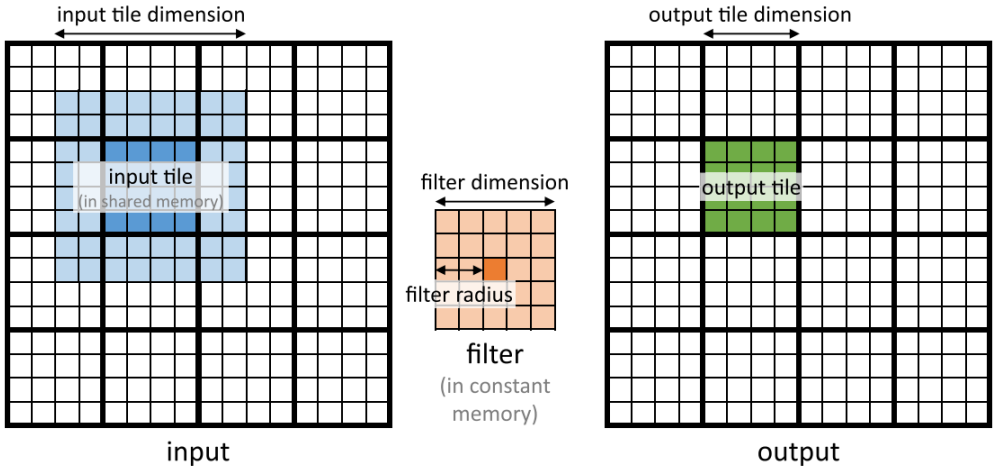
- Định nghĩa hàm kernel blurImgKernel2 – hàm kernel làm mờ ảnh **có sử dụng SMEM**. Mỗi block sẽ đọc phần dữ liệu của mình từ inPixels ở GMEM vào SMEM, sau đó phần dữ liệu ở SMEM này sẽ được **dùng lại nhiều lần** cho các thread trong block. Bạn sẽ **cấp phát động** một mảng ở SMEM của mỗi block để cho phép kích thước của mảng này có thể thay đổi theo filterWidth và blockSize:
  - Trong hàm kernel blurImgKernel2, bạn khai báo mảng s_inPixels ở SMEM như sau:
    `extern __shared__ uchar3 s_inPixels[];`
  - Khi gọi hàm kernel blurImgKernel2, trong cặp <<<...>>>, ngoài tham số gridSize và blockSize, bạn sẽ truyền vào **tham số thứ ba** cho biết kích thước (byte) của mảng s_inPixels trong SMEM của mỗi block (kích thước này được tính theo biến filterWidth và biến blockSize, ở đây ta cũng lưu mảng 2 chiều dưới dạng mảng 1 chiều).
- Gọi hàm blurImgKernel2.

Lưu ý:

Phần dữ liệu cần copy từ ***inPixels*** sang ***SMEM*** sẽ lớn hơn kích thước block, do phải copy thêm các phần tử xung quanh input để có thể thực hiện convolution tại các vùng gần biên.

Hình bên dưới thể hiện ý tưởng:

- Block output màu **xanh lá**
- Block Input màu xanh **dương đậm**.
- Cần phải copy thêm phần xanh dương nhạt xung quanh block input.



- Định nghĩa hàm kernel blurImgKernel3 – hàm kernel làm mờ ảnh có sử dụng SMEM cho inPixels và **sử dụng CMEM cho filter**. Ở đầu file code, mình đã khai báo cho bạn mảng dc_filter ở CMEM. Do dc_filter ở tầm vực toàn cục nên trong hàm kernel blurImgKernel3 bạn có thể sử dụng được dc_filter (do đó, không cần tham số đầu vào ứng với filter nữa). Nếu bạn đã viết xong hàm kernel blurImgKernel2 thì bạn chỉ cần copy-paste và sửa filter thành dc_filter.
- Copy dữ liệu từ filter ở host sang dc_filter ở CMEM (dùng hàm cudaMemcpyToSymbol).
- Gọi hàm blurImgKernel3.

Hướng dẫn về các câu lệnh:

(Các câu lệnh dưới đây là chạy trên terminal của Linux, khi chạy ở code cell của Colab thì bạn thêm dấu ! ở đầu)

- Biên dịch file HW3.cu: nvcc HW3.cu -o HW3
- Chạy file HW3 với file ảnh đầu vào là in.pnm, file ảnh đầu ra là out.pnm:
  ./HW3 in.pnm out.pnm
  Lúc này, chương trình sẽ: đọc file ảnh in.pnm; làm mờ ảnh bằng host (để có kết quả đúng làm chuẩn), và làm mờ ảnh bằng device với 3 phiên bản của hàm kernel: blurImgKernel1, blurImgKernel2, và blurImgKernel3; các kết quả sẽ được ghi xuống 4 file: out_host.pnm, out_device1.pnm, out_device2.pnm, và out_device3.pnm. Với mỗi hàm kernel, chương trình sẽ in ra màn hình thời gian chạy và sự sai biệt so với kết quả của host (mình chạy thì thấy kết quả của device có sự sai biệt **nhỏ** so với kết quả của host, khoảng 0.000x; đó là do GPU tính toán số thực có thể hơi khác so với CPU, chứ không phải là do cài đặt sai).

  Mặc định thì chương trình sẽ dùng block có kích thước 32×32; nếu bạn muốn chỉ định kích thước block thì truyền thêm vào câu lệnh 2 con số lần lượt ứng với kích thước theo chiều x và theo chiều y của block (ví dụ, ./HW3 in.pnm out.pnm 32 16).

Báo cáo (2đ)

Trong file "HW3.ipynb" mà mình đính kèm:

- Bạn biên dịch và chạy file "HW3.cu"
- Giải thích tại sao kết quả lại như vậy (tại sao dùng SMEM lại chạy nhanh/chậm hơn so với không dùng, tại sao dùng CMEM lại chạy nhanh/chậm hơn so với không dùng). Nếu cần thì bạn có thể thực hiện thêm các thí nghiệm để kiểm chứng cho lý giải của bạn. Chỗ nào mà bạn không biết tại sao thì cứ nói là không biết tại sao.

Câu 2 (3đ)

Áp dụng luồng CUDA để tối ưu hóa chương trình thực hiện cộng 2 véc-tơ.

Cụ thể, với nStreams luồng thì bạn sẽ chia véc-tơ output ra làm nStreams phần (phần cuối sẽ có thể có ít số lượng phần tử hơn các phần còn lại); mỗi véc-tơ input cũng sẽ được chia làm nStreams phần tương ứng. Việc tính toán (chép dữ liệu từ host sang device, các thread ở device thực thi hàm kernel, chép dữ liệu từ device về host) các phần khác nhau trong véc-tơ output sẽ được đưa vào các stream khác nhau → các stream có thể overlap với nhau → tận dụng hiệu quả các tài nguyên phần cứng hơn.

Code (2đ)

Mình đã viết sẵn cho bạn khung chương trình trong file "HW3_P2.cu" đính kèm; bạn chỉ viết code ở những chỗ có từ "// TODO" trong nhánh else (dùng device) của hàm addVec. Hàm addVec này có tham số khá tương tự như các hàm ở HW1 (bạn nhìn qua là có thể hiểu được ngay), nhưng có thêm tham số nStreams cho biết số lượng stream được sử dụng.

Hướng dẫn về các câu lệnh (các câu lệnh dưới đây là chạy trên terminal của Linux, khi chạy ở code cell của Colab thì bạn thêm dấu ! ở đầu):

- Biên dịch file "HW2_P2.cu": nvcc HW3_P2.cu -o HW3_P2

- Chạy file “HW3_P2”: ./HW3_P2

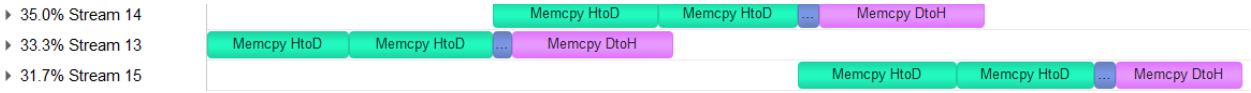
Mặc định thì sẽ dùng block có kích thước 512 và số lượng stream là 1; nếu bạn muốn dùng block có kích thước khác, chẳng hạn 256, và số lượng stream khác, chẳng hạn 3, thì bạn truyền thêm hai tham số dòng lệnh: ./HW3_P2 **256 3**

Báo cáo (1đ)

Ở mục “Câu 2” trong file “HW3.ipynb” mà mình đính kèm, bạn biên dịch file “HW3_P2.cu” và chạy với số lượng stream bằng 1 và bằng 3 (để kích thước block là 512). Bạn cũng cần chụp lại kết quả của NVIDIA Nsight System để cho thấy dùng 3 stream có sự overlap giữa các công việc, còn dùng 1 stream thì không có sự overlap. Ví dụ:

Dùng 1 stream:

Dùng 3 stream:



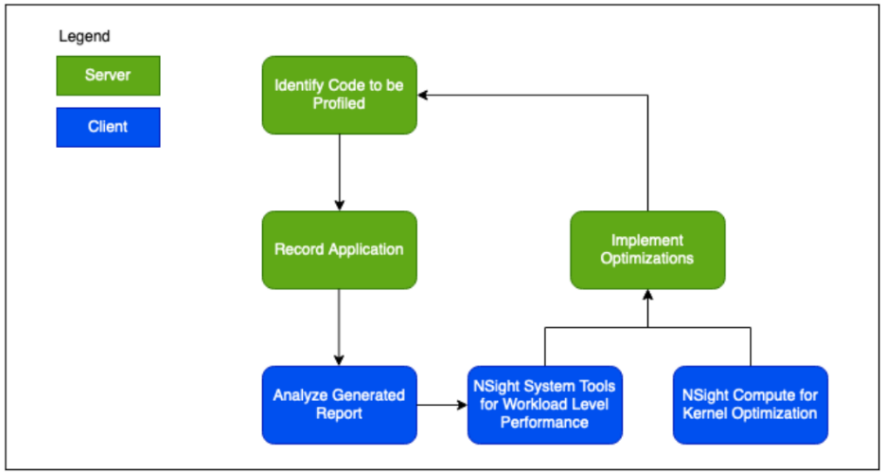
Ở Windows, bạn có thể chụp lại một phần màn hình bằng cách ấn ⊞+Shift+s rồi kéo chuột để chọn vùng cần chụp. Sau khi đã lưu ảnh xuống file, bạn có thể cho ảnh này hiển thị ở text cell của file “HW3.ipynb” bằng chọn “insert image” (nút có biểu tượng ảnh ở text cell của Colab) rồi chọn file ảnh ở máy của bạn.

Hướng dẫn dùng NVIDIA Nsight System:

- Ở Colab, sau khi đã biên dịch ra file chạy, chẳng hạn “a.out”, thì bạn có thể chạy và xem các thông tin (ví dụ, thời gian chép dữ liệu từ host sang device, thời gian chạy hàm kernel, …) bằng câu lệnh nvprof (ở code cell của Colab nhớ thêm dấu ! ở đầu): nvprof ./a.out
- Chúng ta đã thử dùng nvprof trong bài HW2. Tuy nhiên nvprof là nền tảng cũ, đã outdate. Để xem được nhiều thông tin hơn, thông tin trực quan hơn, NVIDIA cung cấp cho chúng ta các công cụ ***Nsight System***, ***Nsight Compute***.

Để biết thêm thông tin về các công cụ trên, các bạn có thể tham khảo thêm ở đây:

https://www2.cisl.ucar.edu/events/gpu-series-hands-session-nsight-systems-and-compute



Hình trên mô tả quy trình profiling (xem chi tiết thông tin thực thi để optimize code)

Gồm 2 phần:

- **Server**: nơi chạy CUDA code và tạo ra report. VD: Google Colab.
- **Client**: nơi xem report. VD: máy tính cá nhân của bạn

Cả hai đều cần phải cài **Nsight System**.

- Cài Nsight System trên Google Colab
  - Trong file HW3.ipynb đã ghi sẵn cách cài. Các bạn chỉ cần chạy cell tương ứng.
  - Sau khi cài xong, tạo ra report bằng cách gõ câu lệnh sau: nsys profile ./a.out (a.out là file thực thi, đã biên dịch bằng nvcc trước đó, có thể thêm tham số dòng lệnh sau ***a.out*** nếu cần.
  - File ***report*** sẽ được tạo ra sau câu lệnh trên. Download về máy.
- Cài Nsight System trên máy tính cá nhân:
  - Cài đặt thông qua trang chủ của NVIDIA
  - Sau khi cài xong, cách đơn giản nhất để mở file ***report*** ở Windows là double click vào file này
  - Để *Nsight system* ở máy cá nhân có thể mở được file ***report*** được tạo ra bằng *Nsight system* ở Colab thì phiên bản *nsight system* ở máy cá nhân phải bằng hoặc cao hơn phiên bản *nsight system* ở Colab. Bạn có thể check phiên bản của *nsight system* ở Colab bằng câu lệnh nsys --version
  - Khi file ***report*** được mở ra ở *nsight system*, có thể bạn sẽ cần zoom lớn một xíu mới thấy được cái cần thấy (CTRL + Scroll, Select section -> Zoom into selection).

Nộp bài

Bạn tổ chức thư mục bài nộp như sau:

Thư mục <MSSV> (vd, nếu bạn có MSSV là 1234567 thì bạn đặt tên thư mục là 1234567)

- File code “HW3_P1.cu”
- File code “HW3_P2.cu”
- File báo cáo “HW3.ipynb”

Sau đó, bạn nén thư mục <MSSV> này lại và nộp ở link trên moodle.