Lập trình song song
HW1: Giới thiệu CUDA (hay CUDA ở mức cơ bản)

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 (3.5đ)

Viết chương trình chuyển ảnh RGB (ảnh màu) sang ảnh grayscale (ảnh xám) theo công thức:

$$\text{giá-trị-grayscale} = 0.299 \times \text{giá-trị-red} + 0.587 \times \text{giá-trị-green} + 0.114 \times \text{giá-trị-blue}$$

Code (3đ)

Mình có đính kèm:

- File ảnh đầu vào RGB "in.pnm" (trong Windows, bạn có thể xem file *.pnm bằng chương trình IrfanView).
- File khung chương trình "HW1_P1.cu". Chương trình sẽ:
  - Đọc file ảnh đầu vào RGB.
  - Chuyển ảnh đầu vào RGB sang ảnh grayscale vào bằng host (làm tuần tự).
  - Chuyển ảnh đầu vào RGB sang ảnh grayscale bằng device (làm song song). *Bạn sẽ phải code phần này, cụ thể là ở những chỗ mình để "// TODO". Lưu ý: bạn cần kiểm lỗi khi gọi các hàm*

CUDA API (dùng macro CHECK mà mình đã viết sẵn cho bạn) và khi gọi hàm kernel (bạn xem cách kiểm lỗi hàm kernel ở file slide 01 – trang 34).

- So sánh ảnh kết quả của device với host để giúp bạn biết là đã cài đặt đúng hay chưa.
- Ghi các ảnh kết quả xuống file.

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 "HW1_P1.cu": nvcc HW1_P1.cu -o HW1_P1
  Câu lệnh này sẽ biên dịch file "HW1_P1.cu" bằng trình biên dịch nvcc của NVIDIA, và xuất ra file chạy "HW1_P1" (nếu bạn muốn xuất ra file chạy có tên khác thì sau -o bạn thay HW1_P1 bằng tên mà bạn muốn; nếu bạn chỉ gõ là nvcc HW1_P1.cu thì sẽ xuất ra file chạy có tên mặc định là "a.out").
- Chạy file "HW1_P1" với file ảnh đầu vào là "in.pnm" và xuất ảnh kết quả ra file "out.pnm" (kết quả của host sẽ xuất ra file "out_host.pnm", còn device thì là "out_device.pnm"): ./HW1_P1 in.pnm out.pnm
  Chương trình sẽ in ra thời gian thực thi của host và của device, và giá trị khác biệt trung bình giữa ảnh kết quả của host và của device (được tính bằng cách: lấy các pixel tương ứng của 2 ảnh trừ cho nhau, lấy trị tuyệt đối, và cuối cùng tính trung bình hết lại). Như vậy, nếu giá trị khác biệt trung bình bằng 0 thì nghĩa là 2 ảnh giống hệt nhau. Nếu giá trị khác biệt trung bình có giá trị nhỏ (ví dụ, 0.xxx) thì chưa chắc là sai, vì khi tính toán số thực thì giữa CPU và GPU có thể có sai biệt nhỏ; nhưng nếu giá trị khác biệt trung bình lớn hơn 0.xxx thì chắc là sai rồi.
  Mặc định thì 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, ./HW1_P1 in.pnm out.pnm 32 16).

File báo cáo (0.5đ)

Bạn làm trong file "HW1.ipynb" mà mình đính kèm, mục "Câu 1": biên dịch file code và chạy với các kích thước block khác nhau: 16x16, 32x32, 64x64 (nếu kiểm lỗi đúng thì sẽ bắt được lỗi khi chạy với block 64x64).

Câu 2 (6.5đ)

Viết chương trình làm mờ ảnh RGB. Để làm mờ ảnh RGB, ta sẽ thực hiện phép tích chập (convolution) giữa một filter (bộ lọc) với từng kênh màu của ảnh. Để rõ hơn về phép tích chập, bạn có thể xem và thử nghiệm ở link này:

- Trong link, từ "kernel" = từ "filter" của mình (mình không dùng từ "kernel" để tránh nhập nhằng với hàm kernel trong CUDA).
- Filter thường là mảng 2 chiều vuông, và kích thước của mỗi chiều là số lẻ (như vậy thì filter sẽ có một phần tử chính giữa); ví dụ, trong link, filter có kích thước 3×3.
- Với các filter khác nhau thì khi áp dụng lên ảnh sẽ có các hiệu ứng khác nhau. Ví dụ, ở link, bạn có thể chọn filter có hiệu ứng làm mờ ảnh (blur) hoặc làm sắc nét ảnh (sharpen).
- Trong link, ảnh input là ảnh xám (một kênh màu). Với ảnh RGB thì cách làm là làm trên từng kênh màu, và với mỗi kênh màu thì làm giống ảnh xám.

- Như bạn có thể thấy trong link, để tính giá trị của một phần tử ở chỉ số dòng r và chỉ số cột c trong ảnh output thì ta sẽ đặt filter lên ảnh input sao cho phần tử chính giữa của filter trùng với phần tử ở chỉ số dòng r và chỉ số cột c trong ảnh input; rồi lấy các phần tử tương ứng của filter và ảnh input nhân với nhau; cuối cùng là cộng hết tất cả các tích này lại.
- Khi đặt filter ở vùng biên của ảnh input thì một số phần tử của filter có thể sẽ không có phần tử tương ứng của ảnh input để nhân. Các cách xử lý:
    - Trong link: bỏ đi các trường hợp này → ảnh output sẽ có kích thước nhỏ hơn ảnh input.
    - Trong bài tập này: giữ kích thước ảnh output như ảnh input bằng cách thêm các phần tử giả vào ảnh input và tính bình thường. Trong bài tập này, giá trị của phần tử giả (có chỉ số dòng không hợp lệ, hoặc chỉ số cột không hợp lệ, hoặc cả hai) sẽ được lấy là giá trị của phần tử thật (có các chỉ số hợp lệ) gần nhất:
        - Nếu phần tử giả có chỉ số dòng < 0 thì lấy phần tử thật có chỉ số dòng = 0, còn nếu phần tử giả có chỉ số dòng > số dòng – 1 thì lấy phần tử thật có chỉ số dòng = số dòng – 1. Nếu phần tử giả có chỉ số dòng hợp lệ thì giữ nguyên chỉ số dòng (trong trường hợp này, chỉ số cột sẽ không hợp lệ).
        - Chỉ số cột của phần tử giả: tương tự như chỉ số dòng ở trên.

Code (6đ)

Mình có đính kèm:

- File ảnh đầu vào RGB "in.pnm" (cùng file với câu 1).
- File ảnh đầu ra RGB đúng "out2_target.pnm". Trong câu này, mình đính kém file ảnh đầu ra đúng vì bạn sẽ phải code cả phần làm tuần tự lẫn phần làm song song (cụ thể ở dưới); ở câu 1 thì không cần file này vì phần code tuần tự đã được cung cấp, và dựa vào đây là bạn biết đã code đúng hay chưa.
- File khung chương trình "HW1_P2.cu". Chương trình sẽ:
    - Đọc file ảnh đầu vào RGB (mảng filter được tạo trực tiếp trong code nên không phải đọc từ file), và file ảnh đầu ra đúng.
    - Làm mờ ảnh đầu vào RGB bằng host (làm tuần tự). *Bạn sẽ phải code phần này (nhắc lại: qui định chung là bạn sẽ code ở những chỗ mà mình để "// TODO").*
    - So sánh ảnh kết quả của host với ảnh kết quả đúng để giúp bạn biết là đã cài đặt đúng hay chưa.
    - Làm mờ ảnh đầu vào RGB bằng device (làm song song). *Bạn sẽ phải code phần này. Lưu ý: bạn cần kiểm lỗi khi gọi các hàm CUDA API (dùng macro CHECK mà mình đã viết sẵn cho bạn) và khi gọi hàm kernel (bạn xem cách kiểm lỗi hàm kernel ở file slide 01 – trang 34).*
    - So sánh ảnh kết quả của device với ảnh kết quả đúng để giúp bạn biết là đã cài đặt đúng hay chưa.
    - Ghi các ảnh kết quả xuống file.

  File khung chương trình này khá giống với file ở câu 1, có khác ở vụ có thêm ảnh đầu ra đúng. Ngoài ra, còn khác ở chỗ: ở câu 1, mình lưu ảnh dưới dạng mảng một chiều, trong đó mỗi phần tử có kiểu dữ liệu là uint8_t (với ảnh RGB thì 3 phần tử liên tiếp nhau tạo thành 1 pixel, với ảnh grayscale thì 1 phần tử là 1 pixel); ở câu này, vì mình chỉ làm với ảnh RGB nên để cho tiện khi truy xuất thì mình để kiểu dữ liệu của mỗi phần tử của mảng là uchar3 luôn (uchar3 là struct được định nghĩa sẵn trong CUDA, struct này gồm 3 thành phần số nguyên 8 bit không dấu là x, y, và z) và 1 phần tử ứng với 1 pixel của ảnh RGB.

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 "HW1_P2.cu": nvcc HW1_P2.cu -o HW1_P2
  Câu lệnh này sẽ biên dịch file "HW1_P2.cu" bằng trình biên dịch nvcc của NVIDIA, và xuất ra file chạy "HW1_P2".
- Chạy file "HW1_P2" với file ảnh đầu vào là "in.pnm", xuất ảnh kết quả ra file "out2.pnm" (kết quả của host sẽ xuất ra file "out2_host.pnm", còn device thì là "out2_device.pnm"), và dùng file ảnh kết quả đúng "out2_target.pnm" để kiểm tra đúng/sai: ./HW1_P2 in.pnm out2.pnm out2_target.pnm
  Chương trình sẽ in ra thời gian thực thi của host, giá trị khác biệt trung bình giữa ảnh kết quả của host và ảnh kết quả đúng (cách tính như ở câu 1), thời gian thực thi của device, giá trị khác biệt trung bình giữa ảnh kết quả của device và ảnh kết quả đúng.
  Mặc định thì 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, ./HW1_P2 in.pnm out2.pnm out2_target.pnm 32 16).

File báo cáo (0.5đ)

Bạn làm trong file "HW1.ipynb" mà mình đính kèm, mục "Câu 2": biên dịch file code và chạy với các kích thước block khác nhau: 16x16, 32x32, 64x64 (nếu kiểm lỗi đúng thì sẽ bắt được lỗi khi chạy với block 64x64).

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 "HW1_P1.cu"
  - File code "HW1_P2.cu"
  - File báo cáo "HW1.ipynb"

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