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

Cập nhật lần cuối: 23/11/2021

3 ninja = level 3 😉

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

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).

Code (7đ)

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.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. Nếu cần thì bạn có thể tham khảo code cài đặt tuần tự ở hàm blurImg (ở if (useDevice == false)).

- 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.

- 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 (3đ)

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.

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.cu”
- File báo cáo “HW3.ipynb”

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