Lập trình song song trên GPU
BT02: Cách thực thi song song trong CUDA

Cập nhật 18/11/2020

2 ninja = level 2 😉

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 (6đ)

Áp dụng những hiểu biết về cách thực thi song song trong CUDA để tối ưu hóa chương trình thực hiện "reduction" (cụ thể: ta sẽ tính tổng của một mảng số nguyên).

Code (2đ)

Mình đã viết sẵn cho bạn khung chương trình trong file "bt02_p1.cu" đính kèm; bạn chỉ viết code ở những chỗ có từ "// TODO":

- Hàm kernel 1, 2, và 3 (bạn xem nội dung cụ thể của các hàm kernel này trong slide "07-CachThucThiTrongCUDA_P3.pdf"; *ở đây, để đơn giản, ta giả định 2*kích-thước-block = 2^k* với k là một số nguyên dương nào đó). Hàm kernel sẽ tính giá trị tổng cục bộ trong từng block; sau đó, host sẽ chép các giá trị tổng cục bộ này về bộ nhớ của mình và cộng hết lại để ra giá trị tổng toàn cục.
- Tính "gridSize" (từ "blockSize" và "n") khi gọi hàm kernel.

Với mỗi hàm kernel, chương trình sẽ in ra:

- Kích thước grid và kích thước block.
- Thời gian chạy của hàm kernel ("kernel time") và thời gian host thực hiện cộng các giá trị tổng cục bộ của các block ("post-kernel time").
- "CORRECT" nếu kết quả tính được giống với kết quả đúng, "INCORRECT" nếu ngược lại.

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 "bt02_p1.cu": nvcc bt02_p1.cu -o bt02_p1
- Chạy file "bt02_p1": ./bt02_p1
  Mặc định thì sẽ dùng block có kích thước 512; nếu bạn muốn dùng block có kích thước khác, chẳng hạn 256, thì bạn truyền thêm tham số dòng lệnh: ./bt02_p1 **256**

Báo cáo (4đ)

Bạn làm ở các mục "Câu 1A" và "Câu 1B" trong file "bt02.ipynb" mà mình đính kèm.

Câu 1A (2đ)

Biên dịch và chạy file "bt02_p1.cu" (để kích thước block mặc định).

Từ kết quả chạy, bạn so sánh "kernel time" của 3 hàm kernel với nhau và thử giải thích xem tại sao lại như vậy (chỗ nào bạn không biết tại sao thì cứ nói là không biết tại sao).

Câu 1B (2đ)

Chạy file đã biên dịch ở câu 1 với các kích thước block khác nhau: 1024, 512, 256, 128.

Ở câu này ta chỉ tập trung vào kết quả của hàm kernel 1. Với mỗi kích thước block, bạn điền các kết quả của hàm kernel 1 theo mẫu bảng biểu bên dưới (trong đó, Total time = Kernel time + Post-kernel time). Với số block / SM và occupancy, ngoài việc điền kết quả vào bảng thì bạn cũng cần giải thích thêm là tại sao lại tính ra được như vậy. Khi tính số block / SM và occupancy, tạm thời ta chỉ xét 2 ràng buộc của SM là số block tối đa và số thread tối đa; bạn có thể tra cứu 2 ràng buộc này ở document của CUDA, mục "Programming Guide" ("Guide" không có s), mục "H. Compute Capabilities", bảng "Table 14. Technical Specifications per Compute Capability", 2 ràng buộc đó là "Maximum number of resident blocks per multiprocessor" và "Maximum number of resident threads per multiprocessor".

Block size	Grid size	Num blocks / SM	Occupancy (%)	Kernel time (ms)	Post-kernel time (ms)	Total time (ms)
1024						
512						
256						
128						

Từ bảng biểu đã điền, bạn thử suy nghĩ và giải thích xem tại sao khi thay đổi block size thì "kernel time" và "post-kernel time" lại thay đổi như vậy? (chỗ nào bạn không biết tại sao thì cứ nói là không biết tại sao)

Câu 2 (4đ)

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 "bt02_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 BT01 (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 "bt02_p2.cu": nvcc bt02_p2.cu -o bt02_p2
- Chạy file "bt02_p2": ./bt02_p2

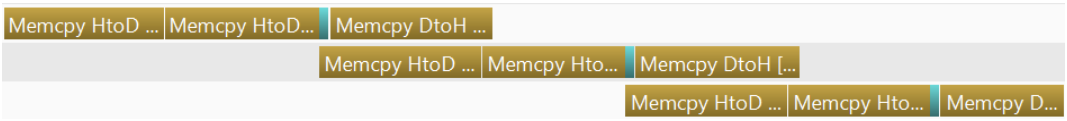
Mặc định thì sẽ dùng block có kích thước 512 và số lượng stream là 1; nếu bạn muốn dùng block có kích thước khác, chẳng hạn 256, và số lượng stream khác, chẳng hạn 3, thì bạn truyền thêm hai tham số dòng lệnh: ./bt02_p2 **256 3**

Báo cáo (2đ)

Ở mục "Câu 2" trong file "bt02.ipynb" mà mình đính kèm, bạn biên dịch file "bt02_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 Visual Profiler 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 "bt02.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 Visual Profiler:

- 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
- Mặc dù ta có thể điều chỉnh các tham số của nvprof để xem nhiều thông tin hơn, nhưng ta sẽ không thể xem được một cách trực quan sự overlap giữa các stream.
  Để xem được trực quan sự overlap giữa các stream thì:
  - Đầu tiên, bạn sẽ lưu các thông tin mà nvprof thu thập được xuống file, giả sử "a.nvvp" (nvvp là đuôi của NVIDIA Visual Profiler), bằng câu lệnh: nvprof -o a.nvvp ./a.out (để có thể overwrite file "a.nvvp" đã có thì bạn dùng: nvprof -o a.nvvp **-f** ./a.out)
  - Sau đó, bạn download file này về máy cá nhân của bạn và mở bằng NVIDIA Visual Profiler.
    Cách cài Visual Profiler:
    - Download CUDA Toolkit bản 10.0 ở đây (nếu bạn download các bản mới hơn thì bạn phải cài thêm Java Runtime Environment, sẽ khá phiền đấy, nên tốt nhất là bạn down bản 10.0), chọn bản down là bản network (vì ở đây ta chỉ muốn cài đặt phần Visual Profiler trong CUDA Toolkit thôi, nếu chọn bản local thì đầu tiên bạn sẽ down toàn bộ CUDA Toolkit về, sẽ khá tốn thời gian).
    - Khi chạy file cài đặt, bạn sẽ thấy có 2 option là "Express" (gần như là cài đặt toàn bộ CUDA Toolkit) và "Custom" (chọn phần mà bạn muốn cài trong CUDA Toolkit); bạn chọn "Custom", và check chọn CUDA – Development – Tools (còn lại là bỏ check hết).

    Sau khi cài Visual Profiler thì bạn có thể mở file .nvvp bằng cách double click vào file này. Khi file được mở ra, có thể bạn sẽ cần zoom lớn một xíu mới thấy được cái cần thấy (ấn vào nút có hình kính lúp có dấu cộng ở thanh phía trên).

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 “bt02_p1.cu”
- File code “bt02_p2.cu”
- File báo cáo “bt02.ipynb”

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