TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN ĐIỆN TỬ - VIỄN THÔNG

BÁO CÁO
KIẾN TRÚC MÁY TÍNH

Đề tài: Đọc và trình bày về kiến trúc bộ xử lý đồ họa GPU. Minh họa việc lập trình bộ xử lý đồ họa GPU để triển khai chương trình tính ma trận và bộ lọc trung vị.

Sinh viên thực hiện:	Phạm Minh Đức
Mã số sinh viên:	20172476
Mã lớp học:	129277
Giảng viên hướng dẫn:	TS. Tạ Thị Kim Huệ

Hà Nội, 01/2022

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN ĐIỆN TỬ - VIỄN THÔNG

BÁO CÁO
KIẾN TRÚC MÁY TÍNH

Đề tài: Đọc và trình bày về kiến trúc bộ xử lý đồ họa GPU. Minh họa việc lập trình bộ xử lý đồ họa GPU để triển khai chương trình tính ma trận và bộ lọc trung vị.

Sinh viên thực hiện:	Phạm Minh Đức
Mã số sinh viên:	20172476
Mã lớp học:	129277
Giảng viên hướng dẫn:	TS. Tạ Thị Kim Huệ

Hà Nội, 01/2022

LỜI NÓI ĐẦU

Với sự phát triển của công nghệ và khoa học kỹ thuật ngày nay, các sản phẩm kỹ thuật tiên tiến, công nghệ cao đang ngày càng đóng một vai trò quan trọng và cấp thiết trong cuộc sống của mỗi con người chúng ta. Các sản phẩm này có thể được tìm thấy tại khắp nơi xung quanh môi trường mà chính ta sinh sống, đến với mọi hình dáng, chức năng khác nhau, và ngày càng được tích hợp thêm nhiều tính năng để hỗ trợ chúng ta trong quá trình vận hành.

Một trong những kênh để con người chúng ta vận động, tiếp nhận thông tin một cách nhanh chóng, đơn giản, đó chính là việc sử dụng thị giác. Các hình ảnh, các sự vật, sự việc trực quan có khả năng đem lại lượng thông tin rất lớn, và nếu ta có khả năng khai thác, sử dụng được kênh thông tin này, năng suất làm việc, lao động của ta có thể tăng lên đáng kể. Đó là lí do mà ngành công nghiệp phát triển đồ họa máy tính được ra đời, đi liền cùng lịch sử phát triển của ngành công nghiệp bán dẫn và máy vi tính.

Trong khuôn khổ của môn học "Kiến trúc máy tính", em xin lựa chọn đề tài ***"Đọc và trình bày về kiến trúc bộ xử lý đồ họa GPU. Minh học việc lập trình cho bộ xử lý đồ họa GPU để triển khai bộ lọc trung vị cho ảnh và chương trình tính ma trận"*** làm đề tài cho bài tập lớn.

Em xin gửi lời cảm ơn chân thành nhất tới giảng viên hướng dẫn, TS. Tạ Thị Kim Huệ đã nhiệt tình giảng dạy, hướng dẫn trong suốt quá trình em theo học lớp Kiến trúc máy tính. Thông qua môn học, em đã nắm được các kiến thức nền tảng của bộ xử lý RISC-V cùng các kiến thức liên quan, và đã có thể sử dụng những điều học được để áp dụng vào nghiên cứu kiến trúc của bộ xử lý đồ họa GPU đã đưa ra tại đề tài.

Trong quá trình thực hiện và hoàn thành, đề tài của em có thể vẫn còn nhiều những hạn chế và thiếu sót. Vì vậy, em rất mong nhận được sự góp ý, bổ sung của cô để đề tài của em có thể được hoàn thiện hơn nữa.

TÓM TẮT ĐỀ TÀI

Bài làm xin trình bày về đề tài ***"Đọc và trình bày về kiến trúc bộ xử lý đồ họa GPU. Minh học việc lập trình cho bộ xử lý đồ họa GPU để triển khai chương trình tính ma trận và bộ lọc trung vị".***

Bài làm được chia thành 07 mục chính tương ứng với các nội dung được trình bày xuyên suốt báo cáo:

1. Giới thiệu chung về GPU (Graphics Processing Unit)
2. Kiến trúc hệ thống GPU
3. Lập trình GPU
4. Kiến trúc xử lý đa luồng
5. Hệ thống bộ nhớ song song
6. Minh họa lập trình GPU – Triển khai bộ lọc trung vị cho ảnh & Chương trình tính ma trận
7. Kết luận

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

DANH MỤC THUẬT NGỮ VIẾT TẮT

Thuật ngữ viết tắt	Tên đầy đủ
GPU	Graphics Processing Unit
CPU	Central Processing Unit
HPC	High Performance Computing
API	Application Programming Interface
SP	Streaming Processor
SMs	Multithreaded Streaming Multiprocessors
SFU	Special Function Units
ROP	Raster Operation Processors

MỤC LỤC

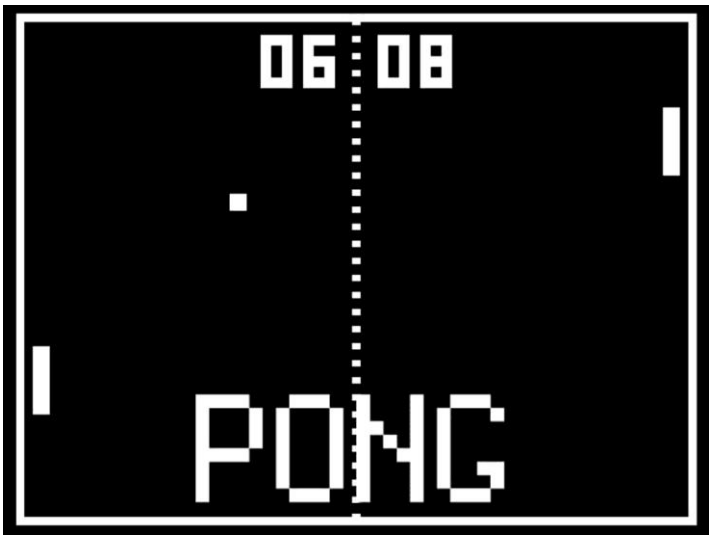
MỞ ĐẦU

Thường thức – Sự phát triển của đồ họa trong ngành công nghiệp trò chơi điện tử

Từ những ngày đầu tiên trong sự phát triển của đồ họa máy tính, các nhà phát triển phải dựa vào những kiến thức, hiểu biết của họ về phần cứng mà họ đang làm việc cùng một cách chặt chẽ. Những kiến thức này bắt nguồn từ những kiến thức nền tảng về điện – điện tử, đồng thời trải qua rất nhiều các thử nghiệm, nghiên cứu trên các thiết bị đó.

Một trong những điểm trình bày rõ ràng nhất về các bước tiến trong ngành công nghệ đồ họa chính là những hình ảnh trực quan được thể hiện trong các trò chơi điện tử (video game). Từ những khởi điểm còn rất sơ khai, cho tới sự bùng nổ của các hệ thống giải trí (gaming arcade) và các thiết bị chơi game tại gia (home console), rồi sự xuất hiện và hoàn thiện của những trò chơi với không gian ba chiều (3D), đồ họa máy tính đã đi được một chặng đường rất dài, đồng hành với sự phát triển của các trò chơi điện tử.

Để phát triển những trò chơi đầu tiên, như Pong (bóng bàn) hay Asteroids (phá thiên thạch), các nhà phát triển phải có một kiến thức rất chắc chắn về những phần cứng mà họ đang sử dụng. Tất cả các yếu tố liên quan, như lượng bộ nhớ có thể sử dụng, hay những màu sắc, âm thanh cần đưa vào bàn cân, để cân đo đong đếm khả năng tận dụng của từng byte của bộ nhớ, hay khả năng xử lý của các vi mạch xử lý.

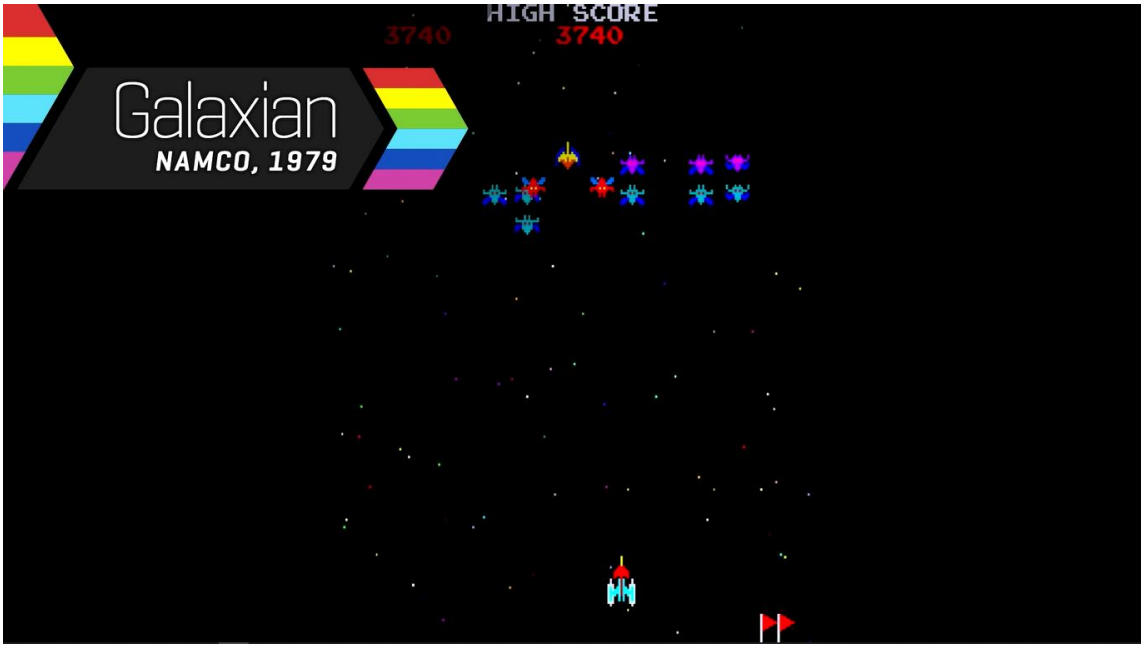


Hình 0. 1: Hình ảnh trò chơi Pong (Atari, 1972), sử dụng raster graphics

Hình 0. 2: Hình ảnh trò chơi Asteroids (Atari, 1979), sử dụng vector graphics

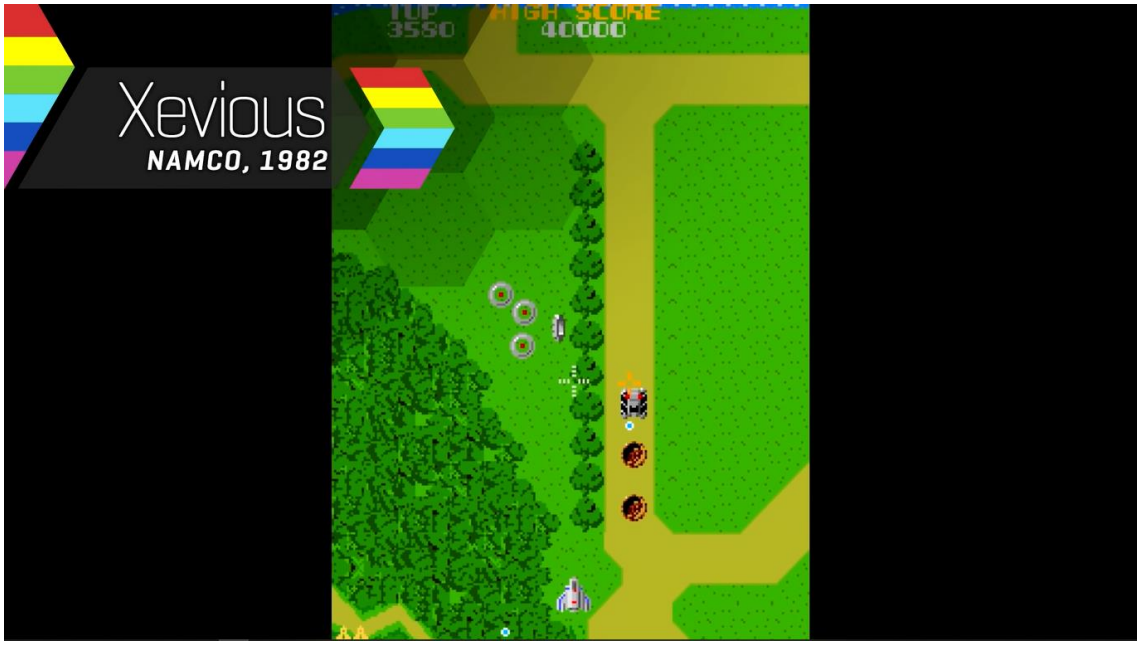
Những khám phá sơ khai tuy rất thú vị, xong sự phát triển của các trò chơi điện tử, hay ngành đồ họa nói chung không thể dừng bước tại các phát minh sơ khai như thế này. Nhu cầu phát triển thêm những phần cứng tiên tiến hơn, có khả năng cung cấp những hình ảnh đồ họa tốt hơn rõ ràng trở thành một yêu cầu cấp bách của thời đại.

Một trong những bước tiến đầu tiên của công nghệ đồ họa: yêu cầu màu sắc trọn vẹn, và đây cũng từng là một trong những ngưỡng cửa mà các trò chơi điện tử cần đạt được. Mặc dù TV màu đã xuất hiện từ những năm trong Thế chiến II, xong các trò chơi điện tử hiện vẫn chỉ đang dừng lại trên màn hình đơn sắc (chỉ có hai màu đen và trắng). Và phải đến năm 1979, trò chơi điện tử đầu tiên đạt thành công lớn trong việc đưa các màu sắc vào trong đồ họa chính là Galaxian (Namco, 1979).



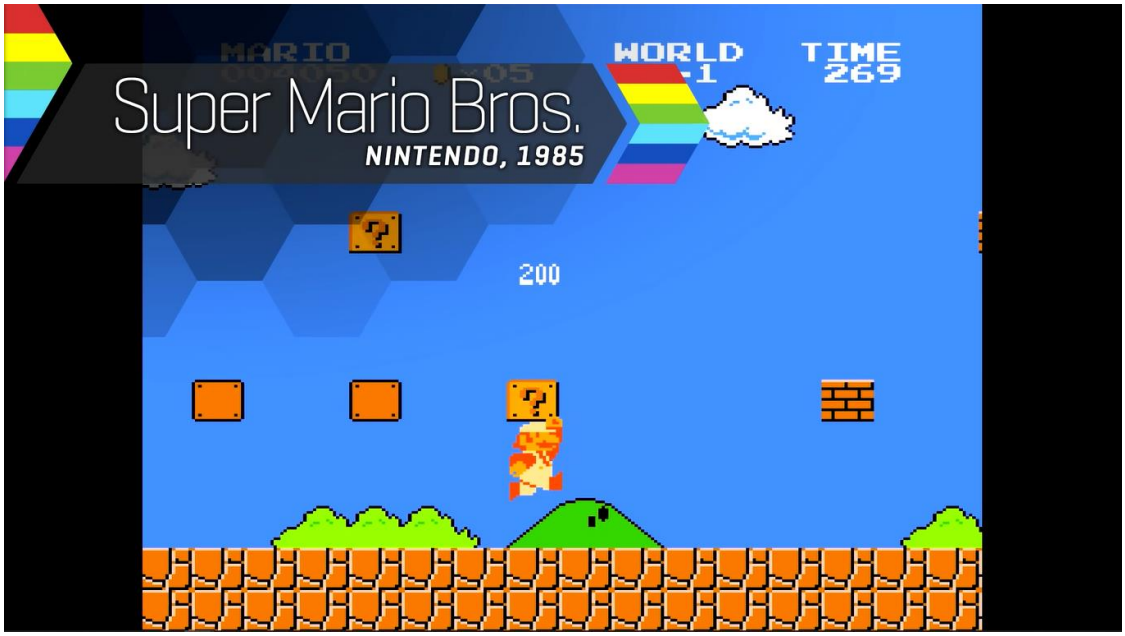
Hình 0. 3: Trò chơi Galaxian (Namco, 1979)

Các trò chơi sơ khai thường chỉ có vùng chơi giới hạn, và việc chuyển giao giữa các vùng bản đồ khác nhau yêu cầu một phần cứng rất lớn để thực hiện. Từ đó những khái niệm như Smooth Scrolling (cuộn bản đồ chơi mượt mà), Sprite Scaling (thu phóng các vật thể theo vị trí), và Parallax Scrolling (cuộn nền khu vực chơi) được phát triển, giúp cho các trò chơi trông ngày càng phức tạp và hấp dẫn hơn trước, trong khi không yêu cầu thêm quá nhiều về tài nguyên phần cứng.



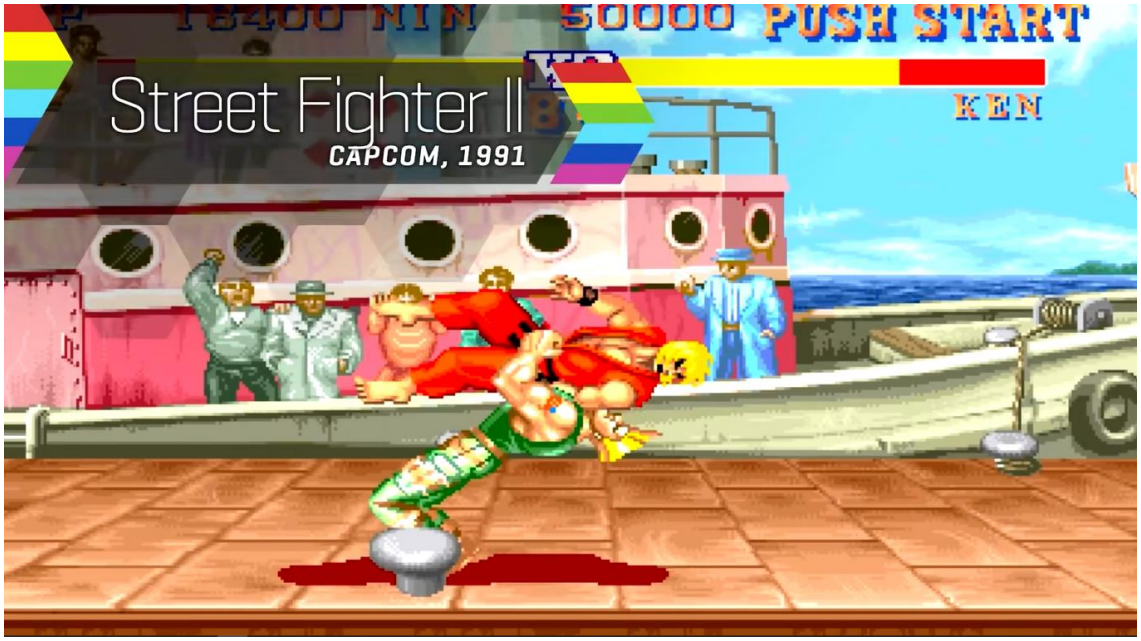
Hình 0. 4: Trò chơi Xevious (Namco, 1982)

Tới những năm 1980, với sự ra đời của đồ họa 16-bit, cùng với sự xuất hiện của các thiết bị chơi game tại gia đã phần nào thay đổi cục diện của ngành phát triển trò chơi điện tử lẫn ngành phát triển đồ họa máy tính. Các thiết bị chơi game tại gia dần phát triển hơn về mặt doanh số, đồng thời cũng có khả năng đem lại các trò chơi với màu sắc và các cách thức xử lý đồ họa thông minh, không thua kém gì các máy điện tử giải trí.



Hình 0. 5: Trò chơi Super Mario Bros (Nintendo, 1985)

Với đồ họa 16-bit, một trong những khía cạnh quan trọng ta phải nhắc tới đó chính là Sprite. Sprite là những hình ảnh hai chiều, được thiết kế để di chuyển quanh bản đồ của game. Đó có thể là nhân vật chính chúng ta đang điều khiển, kẻ thù, hoặc các yếu tố chuyển động khác của trò chơi. Các sprite được phát triển đầu tiên thường có kích thước nhỏ và giới hạn trong màu sắc. Tuy nhiên, với sự phát triển của công nghệ & đặc biệt là các vi mạch được thiết kế đặc biệt để xử lý các sprite, các hình ảnh này ngày càng trở nên lớn hơn, chi tiết hơn, và rực rỡ hơn bao giờ hết. Ta có thể lấy hình ảnh của Super Mario Bros (1985) phía trên, so sánh hình ảnh của tựa game đối kháng nổi tiếng – Street Fighter II (1991) để có thể thấy được bước tiến của công nghệ trong một thời gian ngắn.



Hình 0. 6: Trò chơi Street Fighter II (Capcom, 1991)

Những trò chơi trong thế giới hai chiều (2D) đã xây dựng được một chỗ đứng rất vững chắc, xong, dù với giới hạn về công nghệ và phương pháp xử lý, các nhà phát triển vẫn hướng tới việc vượt qua giới hạn của 2D, làm tiền đề cho sự phát triển của đồ họa ba chiều. Một số phương pháp, hay thủ thuật đã được đưa ra, nhằm giới thiệu thêm một chiều không gian thứ ba vào trong các trò chơi 2D truyền thống. Trong đó phương pháp nổi tiếng nhất là Flat shading (đổ bóng phẳng).

Hình 0. 7: Trò chơi Super Mario Kart (Nintendo, 1992)

Sự kết hợp giữa việc xây dựng hình ảnh đa giác 3D, Sprite Scaling và các yếu tố 2D khác đã đem lại thành công cho trò chơi Star Fox (1993). Đồng thời, trò chơi này cũng đã cho thấy *việc chuyển quá trình xử lý đồ họa ra một bộ xử lý rời là một phương pháp phù hợp và rất có ích cho quá trình phát triển sau này*.

Hình 0. 8: Trò chơi Star Fox (Nintendo, 1993)

Vào những năm 1990, các máy tính IBM PC có một lợi thế rất lớn trong phát triển đồ họa máy tính – việc thiết kế hệ thống dựa trên các mô đun. Nhờ vậy, các nhà phát triển có thể tiếp tục đẩy giới hạn của đồ họa đi xa hơn nữa. Khả năng sử dụng VGA (hay bộ xử lý đồ họa riêng) là một bước tiến rất lớn, với khả năng cung cấp 256 màu sắc, khả năng thể hiện sắc thái rộng lớn hơn hẳn, và là một điểm khác biệt lớn so với các hình ảnh đồ họa trước đó, vốn chỉ bao gồm các màu sắc rực rỡ, tươi sáng.

Hình 0. 9: Trò chơi Wing Commander (Origin Systems, 1990)

Tiếp nối cho sự xuất hiện của các trò chơi với đô họa 3D, sự xuất hiên của các thiết bị tăng tốc phần cứng 3D dần được phát triển và ra mắt, và một lần nữa làm thay đổi cục diện của ngành đồ họa máy tính. Khi các bộ xử lý đồ họa 3D xuất hiện trên thị trường, thiết bị này đã trở thành một phần không thể thiếu trong các thiết bị máy tính cá nhân, hay các thiết bị giải trí điện tử khác, mỗi khi chúng ta đề cập đến các trò chơi điện tử.

Hình 0. 10: Trò chơi Quake (id Software, 1996)

Thể loại trò chơi điện tử nhận được nhiều sự quan tâm nhất trong quá trình phát triển của các bộ tăng tốc phần cứng 3D thuộc về các trò chơi bắn súng với góc nhìn thứ nhất (First person shooter – FPS). Đây cũng là thể loại trò chơi nhận được nhiều lợi ích nhất dựa trên sự phát triển của phần cứng trong xử lý đồ họa. Và trong suốt quá trình phát triển, bắt đầu từ khi chúng ta xây dựng được những bộ xử lý đồ họa, cho tới thời điểm hiện tại, các trò chơi điện tử, hay về ngành xử lý đồ họa nói chung, đã đạt được tới những cảnh giới rất cao, cho phép chúng ta tạo ra những hình ảnh rất sát, rất thật so với cuộc sống thực tế.

Hình 0. 11: Trò chơi Bioshock Infinite (Irrational Games, 2013)

Thông qua lịch sử phát triển của đồ họa máy tính trong các trò chơi điện tử, từ những ngày đầu tiên sơ khai, cho đến những thành tựu mà chúng ta có thể nhìn được, cảm nhận được như thế giới thật tại thời điểm hiện tại năm 2021, có thể nói rằng, công nghệ xử lý hình ảnh, đồ họa của chúng ta đã có những bước tiến hết sức ngoạn mục.

***Tham khảo**: Chuỗi video về đồ họa máy tính trong các trò chơi điện tử - YouTube: Ahoy – A Brief History of Graphics [5].*

1. GIỚI THIỆU CHUNG VỀ GPU (GRAPHICS PROCESSING UNIT)

Bộ xử lý đồ họa – Graphics Processing Unit, hay còn viết tắt là GPU, nay đã trở thành một phần không thể thiếu trong các thiết bị điện tử thông minh. Vậy GPU là gì? Điểm mạnh của GPU được thể hiện qua những mặt nào? Ta sẽ cùng đi vào tìm hiểu tại phần đầu tiên, giới thiệu chung về GPU.

1.1. Khái niệm

GPU – Graphics Processing Unit, ngày nay, là một trong những thiết bị quan trọng trong công nghệ tính toán, đối với cả nhu cầu sử dụng cá nhân lẫn nhu cầu sử dụng trong kinh tế/công nghiệp. Được thiết kế để thực hiện các tác vụ tính toán song song, bộ GPU được sử dụng trong rất nhiều các ứng dụng, mà đặc biệt nhất là trong ứng dụng đồ họa và xử lý băng hình (graphics & video rendering).

GPU khởi điểm được thiết kế với mục đích nhằm tăng tốc khả năng xử lý các hình ảnh đồ họa 3D (3D graphics accelaration). Qua thời gian, các bộ GPU ngày càng trở nên mềm dẻo và linh hoạt hơn, khả năng lập trình được nâng cao, dẫn tới các tiềm năng sử dụng cũng ngày được mở rộng. Bên cạnh các ứng dụng thông thường về đồ họa, nay khả năng xử lý của GPU còn vươn tới các ứng dụng như các tính toán với hiêu năng lớn (high performance computing – HPC), học sâu (deep learning), … và nhiều ứng dụng khác [3].

1.2. Lịch sử phát triển của GPU

15 năm trước (kể từ thời gian phát triển tài liệu tham khảo [1], năm 2007 - 2008), GPU là một khái niệm chưa tồn tại. Đồ họa trên các thiết bị PC vẫn còn được xử lý dựa trên bộ điều khiển VGA (Video graphic array). Bộ điều khiển VGA chỉ đơn thuần là một bộ điều phối bộ nhớ & sinh hình ảnh, được kết nối tới vài DRAM. Trong những năm thập niên 90, ngành công nghiệp bán dẫn đạt được những thành quả lớn, giúp cho việc bổ sung thêm các tính năng vào bộ điều khiển VGA trở nên khả thi.

Năm 1997, bộ điều khiển VGA bắt đầu được bổ sung các tính năng tăng tốc trong xử lý đồ họa 3D, bao gồm tăng tốc phần cứng cho việc “triangle setup & rasterization” – chia các đồ họa hình tam giác thành các điểm ảnh, cùng với việc

"texture mapping & shading" – tương tự như việc áp dụng các đề can hoặc họa tiết lên các điểm ảnh và trộn màu sắc.

Năm 2000, các bộ xử lý hình ảnh được tích hợp hầu hết các cấu trúc pipeline của một bộ xử lý hình ảnh đồ họa cao cấp, sử dụng trong các máy trạm. Vì vậy, bộ xử lý này xứng đáng với một cái tên mới, vượt lên trên bộ điều khiển VGA. Khái niệm GPU ra đời, đánh dấu bước tiến rằng thiết bị xử lý đồ họa nay đã trở thành một bộ xử lý.

Qua thời gian, khả năng lập trình cho GPU ngày càng dễ dàng, cùng với các bộ xử lý có thể lập trình được đang dần thay thế các thiết bị "cứng" – không có khả năng điều chỉnh lại logic, trong khi vẫn giữ được các cấu trúc pipeline cơ bản trong quá trình xử lý đồ họa 3D. Bổ sung vào đó, các tính toán trên máy tính ngày càng trở nên chính xác hơn, cải tiến từ việc chỉ xử lý các phép toán cơ bản, từ số nguyên & số thực dấu chấm tĩnh, tới việc xử lý trên số thực dấu chấm động, và gần đây nhất là số thực dấu chấm động với độ chính xác gấp đôi. Các bộ GPU đã trở thành các bộ xử lý song song, có thể lập trình được với quy mô rất lớn, bao gồm hàng trăm lõi xử lý (core) và hàng ngàn luồng (thread).

Gần đây, các tập lệnh cho bộ xử lý & bộ nhớ phần cứng đã được tích hợp vào để hỗ trợ các ngôn ngữ lập trình đa chức năng, và được sử dụng trong một môi trường lập trình cho phép GPU được lập trình trên các ngôn ngữ quen thuộc (bao gồm C/C++). Bước cải tiến này thực sự khiến cho GPU trở thành một bộ xử lý đa chức năng, có thể lập trình được, với nhiều lõi & luồng xử lý một cách hoàn chỉnh. Điều này cho thấy GPU có những điểm lợi rõ rệt, xong, đi kèm với đó là những hạn chế vẫn còn tồn đọng.

1.3. CPU & GPU: Các điểm giống và khác

Bộ GPU được tiến hóa lên từ một thiết bị tương tự, đó chính là bộ CPU (Central Processing Unit). Ngày nay, cả CPU và GPU đều đang được sử dụng và phát triển cho nhiều chức năng sử dụng khác nhau (GP – General Purpose), và các cải tiến trong ngành sản xuất bán dẫn đều đem lại cho cả hai thiết bị này những nâng cấp về kiến trúc xử lý, tốc độ đồng hồ nhanh hơn, nhiều lõi/luồng xử lý hơn, …

Xong, CPU và GPU vẫn có những khác biệt rõ ràng trong cách sử dụng các thiết bị này:

- Kiến trúc của CPU được thiết kế nhằm xử lý một lượng lớn các thao tác một cách nhanh chóng (được đo bởi tốc độ đồng hồ của CPU), tuy nhiên bị giới hạn bởi số lượng tác vụ có thể được thực hiện trong cùng một thời điểm [4].
- GPU được thiết kế với rất nhiều lõi xử lý có thể chạy đồng thời, đem lại khả năng thực hiện nhiều các tác vụ song song trên nhiều tập dữ liệu khác nhau, có thể thao tác trên những tập dữ liệu rất lớn. Mỗi lõi xử lý của GPU thay vì thực hiện tính toán nhanh nhất có thể, lại tập trung vào khả năng chạy song song và thực hiện các tính toán một cách hiệu quả nhất. [4].
- Cuối cùng, GPU vẫn là một thiết bị bổ sung, được phát triển với mục đích để bù vào các ứng dụng cho CPU, chứ không hoàn toàn thay thế vị trí của CPU. Mục đích cơ bản nhất và phân biệt giữa CPU và GPU nằm ở việc GPU được phát triển nhằm tăng tốc các quá trình xử lý đồ họa trong hệ thống.

1.4. Phân loại GPU

GPU nay được các nhà phát triển và sản xuất cung cấp theo hai loại: GPU tích hợp và GPU rời.

- GPU tích hợp: Chiếm đại đa số các bộ GPU trên thị trường. Đây là bộ GPU được tích hợp sẵn trên bo mạch chủ của các thiết bị vi tính. Điều này cho phép các thiết bị cuối cùng chứa bo mạch chủ có kiểu dáng thon gọn, nhẹ, giảm bớt được năng lượng tiêu thụ và giảm thiểu được chi phí trong sản xuất.
- GPU rời: Đối với các ứng dụng yêu cầu năng lực xử lý đồ họa cao, việc sử dụng các thiết bị GPU riêng là một lựa chọn phù hợp hơn. Tuy phải đánh đổi bằng việc tiêu thụ năng lượng lớn hơn và lượng nhiệt sinh ra trong quá trình vận hành nhiều hơn, hiệu quả làm việc của GPU rời lớn hơn hẳn so với các bộ GPU tích hợp.

1.5. Xu hướng phát triển của GPU

GPU và hệ thống các trình điều khiển (driver) được tích hợp các mô hình của xử lý đồ họa là OpenGL và DirectX. OpenGL là một chuẩn lập trình đồ họa 3D, có thể thực hiện trên hầu hết các máy tính. DirectX là một chuỗi các giao thức lập trình đa phương tiện của Microsoft, bao gồm cả Direct3D cho đồ họa 3D. Nhờ

việc cung cấp các giao diện lập trình (Application Programming Interface – API), với các chuẩn đã được định nghĩa, khả năng lập trình một bộ tăng tốc phần cứng cho việc xử lý đồ họa là khả thi. Đây là một trong nhiều lí do mà những bộ GPU mới được phát triển trong vòng 12 tới 18 tháng có thể đem lại hiệu năng tới gấp đôi thế hệ cũ trên cùng các ứng dụng đang hiện hành.

Việc tăng khả năng xử lý của GPU thường xuyên đồng thời cũng cho phép thực hiện các ứng dụng mới mà trước đây chưa khả thi. Điểm giao giữa xử lý đồ họa & tính toán song song tạo một tiền đề mới cho đồ họa, hay còn được biết tới là tính toán thị giác. Một bộ phận lớn của hệ thống pipeline truyền thống với các phần tử có thể được lập trình dành cho chương trình xử lý hình học, đỉnh & điểm ảnh. Tính toán thị giác trong bộ GPU hiện đại là sự kết hợp giữa việc xử lý đồ họa & tính toán song song, theo các cách cho phép các thuật toán mới về đồ họa có thể được tích hợp, mở ra cánh cửa tới những ứng dụng xử lý song song trên các bộ GPU cấu hình cao.

1.6. GPU xử lý đồ họa trực quan

Xử lý đồ họa trực quan bao gồm các loại ứng dụng đồ họa truyền thống, cộng với nhiều ứng dụng đồ họa mới. Mục đích ban đầu của việc sử dụng GPU là để tính toán "bất kì nội dung gì với điểm ảnh", xong, bài toán xử lý của GPU nay còn bao gồm nhiều vấn đề không bao hàm xử lý điểm ảnh mà là các tính toán thông thường với những cấu trúc dữ liệu phức tạp. Bộ GPU có hiệu năng cao với các đồ họa 2D và 3D, bởi đây là mục đích mà GPU được sinh ra, và việc không đảm bảo được ứng dụng này đồng nghĩa với việc GPU có vấn đề nghiêm trọng. Đồ họa 2D và 3D sử dụng GPU trong "chế độ đồ họa" (graphics mode), là chế độ truy cập vào các khả năng xử lý thông qua các API đồ họa, OpenGL và DirectX. Nhiều trò chơi điện tử được xây dựng dựa trên khả năng xử lý đồ họa 3D.

Ngoài xử lý đồ họa 2D và 3D, xử lý hình ảnh và xử lý video cũng là những ứng dụng quan trọng của các bộ GPU. Các ứng dụng này có thể được xử lý bằng các API đồ họa hoặc qua các chương trình tính toán, sử dụng CUDA để lập trình cho GPU hoạt động trong "chế độ tính toán" (computing mode). Qua việc sử dụng CUDA, các ứng dụng xử lý hình ảnh cũng tương tự với các chương trình xử lý các mảng dữ liệu song song khác. Trong thực tế, việc xử lý dữ liệu ảnh là một ứng dụng rất phù hợp với GPU. Việc xử lý băng hình (video processing), đặc biệt

là quá trình mã hóa (encode) và giải mã (decode) cũng được thực hiện tương đối hiệu quả.

Cơ hội lớn nhất trong ngành xử lý đồ họa trực quan dựa trên GPU lại nằm ở việc "phá vỡ cơ chế pipeline đồ họa" (break the graphics pipeline). Các bộ GPU đầu tiên chỉ dừng lại ở việc tích hợp một số API đồ họa nhất định – với hiệu suất xử lý rất cao, xong lại chỉ giới hạn trong một số các ứng dụng nhất định. Nếu ứng dụng được lập trình không được hỗ trợ bởi API, bộ GPU sẽ không thể tăng tốc quá trình xử lý cho ứng dụng, bởi các chức năng đó của các bộ GPU đầu tiên là bất biến. Tại thời điểm hiện tại, với sự nâng cấp của khả năng xử lý của GPU và sự ra đời của CUDA, các bộ GPU có thể được lập trình để triển khai các pipeline ảo, thông qua việc viết một chương trình CUDA nhằm mô tả luồng tính toán & dữ liệu theo mong muốn. Nhờ vậy, nay mọi ứng dụng đều có thể được triển khai trên GPU, đồng thời tạo bước đệm cho chúng ta tạo ra thêm càng nhiều các cách tiếp cận mới trong xử lý đồ họa trực quan.

1.7. Sử dụng CUDA & GPU trong tính toán

Chuỗi bộ xử lý thống nhất và có thể mở rộng này cần mô hình lập trình mới cho GPU. Với mức độ song song lớn và phạm vi khả năng mở rộng của mảng bộ xử lý cho các ứng dụng đồ họa, mô hình lập trình cho tính toán tổng quát hơn phải thể hiện trực tiếp tính song song lớn, nhưng cho phép thực thi có thể mở rộng.

Xử lý GPU là khái niệm sử dụng GPU cho việc tính toán dựa trên một ngôn ngữ lập trình với khả năng chạy song song & API, nhưng không sử dụng đến các API xử lý đồ họa truyền thống và mô hình pipeline truyền thống. Điều này lại tương phản so với cách tiếp cận Tính toán đa chức năng trên GPU (General Purpose computation on GPU – GPGPU), bao gồm lập trình cho GPU sử dụng các API về đồ họa và hệ thống pipeline để thực hiện các việc tính toán không bao gồm đồ họa.

Compute Unifed Device Architecture (CUDA) là một mô hình lập trình song song có thể mở rộng và nền tảng phần mềm cho GPU và các bộ xử lý song song khác cho phép lập trình viên bỏ qua API đồ họa, giao diện đồ họa của GPU và chỉ cần lập trình bằng C hoặc C ++. Mô hình lập trình CUDA có kiểu phần mềm SPMD (single-program multiple data, hay đơn chương trình với đa dữ liệu), cho

phép người lập trình thực hiện trên một luồng được khởi tạo & xử lý bởi nhiều luồng khác chạy song song trên nhiều bộ xử lý khác nhau của GPU. Trên thực tế, CUDA cũng cung cấp một cơ sở để lập trình nhiều lõi CPU, vì vậy CUDA là một môi trường để viết các chương trình song song cho toàn bộ hệ thống máy tính hỗn hợp.

1.8. GPU đồng nhất việc xử lý đồ họa & xử lý điện toán

Với sự bổ sung của CUDA & tính toán của GPU, giờ ta có thể sử dụng GPU như bộ xử lý đồ họa, lẫn một bộ xử lý tính toán, và có thể kết hợp cả hai tính năng này vào trong việc xử lý đồ họa. Kiến trúc bộ xử lý của GPU có thể được phân tích theo hai đường:

- Thứ nhất, khi triển khai các API đồ họa có thể lập trình được
- Hoặc thứ hai, như một chuỗi các bộ xử lý song song có thể lập trình được bằng ngôn ngữ C/C++ với CUDA

Mặc dù các bộ xử lý của GPU được đồng nhất, các luồng chương trình SPMD không nhất thiết phải tương đồng nhau. GPU có khả năng chạy các chương trình đổ bóng đổ họa bằng khía cạnh xử lý đồ họa của GPU, cũng như việc chạy các chương trình xử lý hình học,… hay chạy các luồng chương trình với CUDA.

Kiến trúc của GPU thật sự là một kiến trúc mềm dẻo, với khả năng đa xử lý linh hoạt, hỗ trợ nhiều các vụ khác nhau. Các bộ GPU hoàn thành xuất sắc trong vai trò xử lý và tính toán đồ họa bởi chúng được thiết kế dành cho ứng dụng này. Các bộ GPU cũng nổi bật trong nhiều ứng dụng thông lượng đa năng có liên hệ chặt chẽ tới đồ họa, nằm ở việc thực hiện nhiều công việc song song, cũng như có nhiều khó khăn khi thực hiện với cấu trúc chương trình thông thường. Tổng quan, GPU là một bộ xử lý phù hợp cho các bài toán yêu cầu xử lý dữ liệu song song, đặc biệt trong các bài toán lớn, và ít hơn trong các bài toán nhỏ, thường xuyên hơn.

2. KIẾN TRÚC HỆ THỐNG GPU

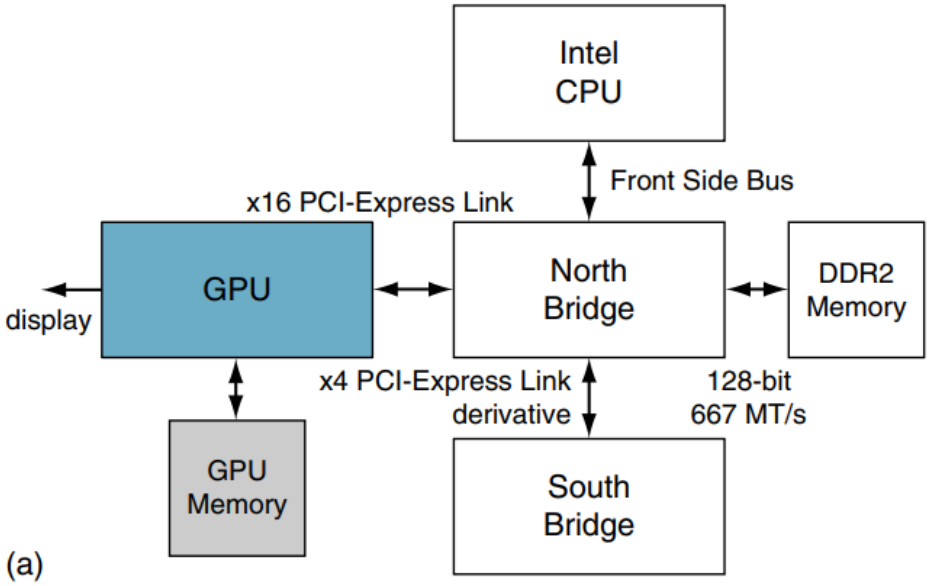
Phần này xin trình bày vè các kiến trúc hệ thống GPU hiện đang được sử dụng. Nội dung bao gồm các cấu hình hệ thống, chức năng và dịch vụ của GPU, giao diện lập trình chuẩn & một kiến trúc nội tại của một bộ GPU cơ bản.

2.1. Kiến trúc hệ thống CPU – GPU hỗn hợp

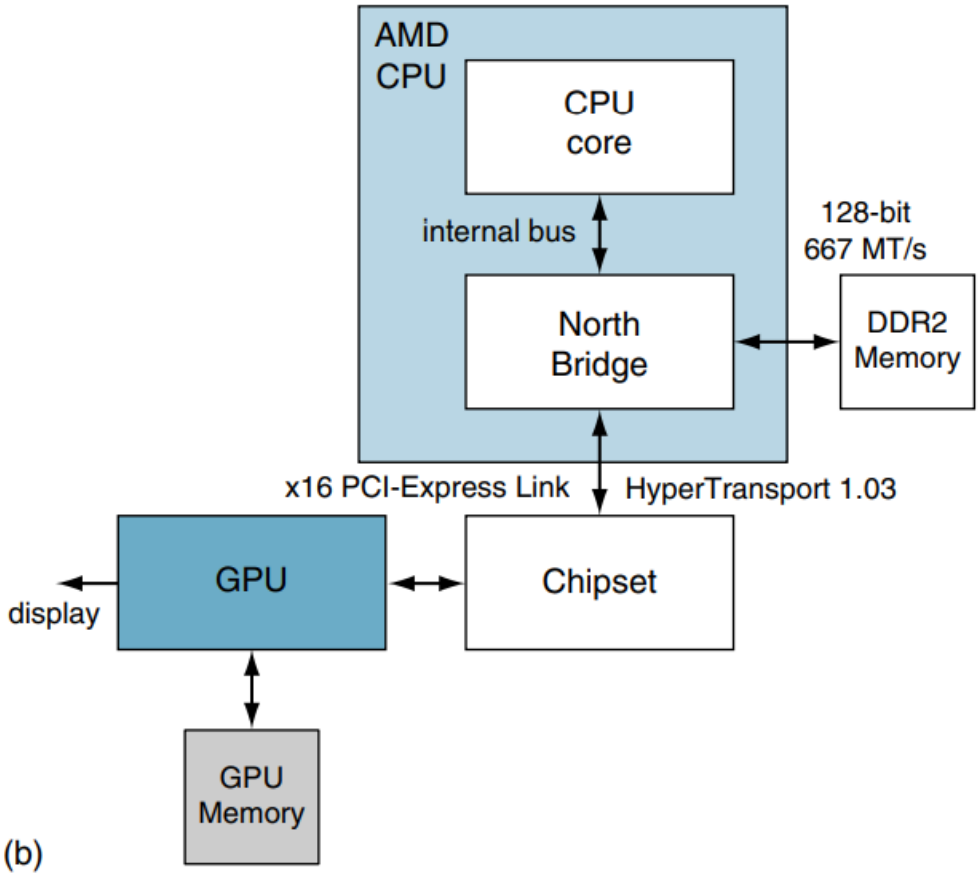
Kiến trúc của một hệ thống máy tính hỗn hợp, sử dụng CPU và GPU có thể được mô tả tại cấp độ cao với hai đặc điểm chính:

- Thứ nhất, hệ thống được thiết kế gồm bao nhiêu hệ thống con, hoặc gồm bao nhiêu chip chức năng được sử dụng cùng sự liên kết giữa chúng
- Và thứ hai, hệ thống bộ nhớ con nào đang sẵn sàng cho các hệ thống chức năng con đã đề cập
-

Hai hệ thống máy tính được sử dụng phổ biến hiện nay:



Hình 2. 1: Sơ đồ khối kiến trúc máy tính với Intel CPU



Hình 2. 2: Sơ đồ khối kiến trúc máy tính với AMD CPU

Phía trên là hình ảnh của hai cấu hình hệ thống máy tính được sử dụng phổ biến hiện nay (Hình 2. 1: Sơ đồ khối kiến trúc máy tính với Intel CPU, Hình 2. 2: Sơ đồ khối kiến trúc máy tính với AMD CPU). Các cấu hình này được đặc trưng bởi việc sử dụng một khối GPU riêng biệt (GPU rời rạc), cùng với CPU và các bộ nhớ liên kết với các hệ thống con tương ứng. Với hình 2. 1, sử dụng CPU Intel, khối GPU được liên kết sử dụng một đường PCI-Express 2.0 với 16 lane, cung cấp tốc độ truyền tải dữ liệu tối đa lên tới 16GB/s (hay 8GB/s theo mỗi hướng – gửi đi và nhận về). Tương tự đối với hình 2. 2, sử dụng CPU AMD, khối GPU được liên kết với Chipset, cũng thông qua đường PCI-Express với băng thông tương tự với ví dụ trên của Intel.

Trong cả hai trường hợp, bộ GPU và CPU có thể truy cập bộ nhớ của nhau, sử dụng ít băng thông đường truyền hơn. Đặc biệt hơn, trong trường hợp của hệ thống AMD, bộ điều khiển bộ nhớ được tích hợp sẵn vào trong bộ CPU.

Với các bộ CPU và GPU, việc quản lý cần sử dụng rất nhiều không gian địa chỉ, nên bộ nhớ đệm được sử dụng để duy trì tính nhất quán trong không gian

địa chỉ sử dụng chung. Các GPU có thể truy cập bộ nhớ nội bộ vật lý của chúng, cũng như bộ nhớ vật lý của CPU thông qua việc sử dụng các địa chỉ ảo, được tính toán bởi bộ MMU nằm trong GPU. Nhân của hệ điều hành (Operating System's kernel) thực hiện việc quản lý các bảng trang của GPU (GPU's page table). Ngược lại, bộ CPU cũng có thể truy cập tới bộ nhớ nội bộ của GPU thông qua dải địa chỉ nằm trong không gian địa chỉ của đường PCI-Express.

Các thiết bị chơi game (Game Consoles)

Các thiết bị chơi game như Sony Playstation 3 hay Microsoft Xbox 360 cũng sử dụng các kiến trúc hệ thống như phía trên đã đề cập. Đặc biệt, các hệ máy chơi game này được thiết kế và sản xuất để đảm bảo chức năng và hiệu năng trên các máy khi xuất xưởng là tương đương nhau, trong vòng đời có thể kéo dài từ 5 năm hoặc hơn. Trong thời gian này, hệ thống có thể được triển khai lại nhiều lần để tối đa việc tận dụng công nghệ trong sản xuất bán dẫn, từ đó có được khả năng cung cấp đều đặn với giá thành thấp hơn. Các hệ thống chơi game này không cần khả năng nâng cấp cho các hệ thống con nằm phía trong như các thiết bị PC, vì vậy các đường liên kết bên trong của các thiết bị này thường có xu hướng được tối ưu hóa cho các thiết bị, thay vì chuẩn hóa.

2.2. Giao diện & trình điều khiển GPU (Interfaces & Drivers)

Trong các thiết bị PC ngày nay, GPU thường được liên kết với CPU thông qua đường PCI-Express. Các chương trình đồ họa tiến hành gọi API của OpenGL hoặc Direct 3D sử dụng GPU làm bộ đồng xử lý. Các API gửi các câu lệnh, chương trình, và dữ liệu tới GPU thông qua các trình điều khiển đồ họa (graphics device driver) được tối ưu hóa dành cho GPU cụ thể.

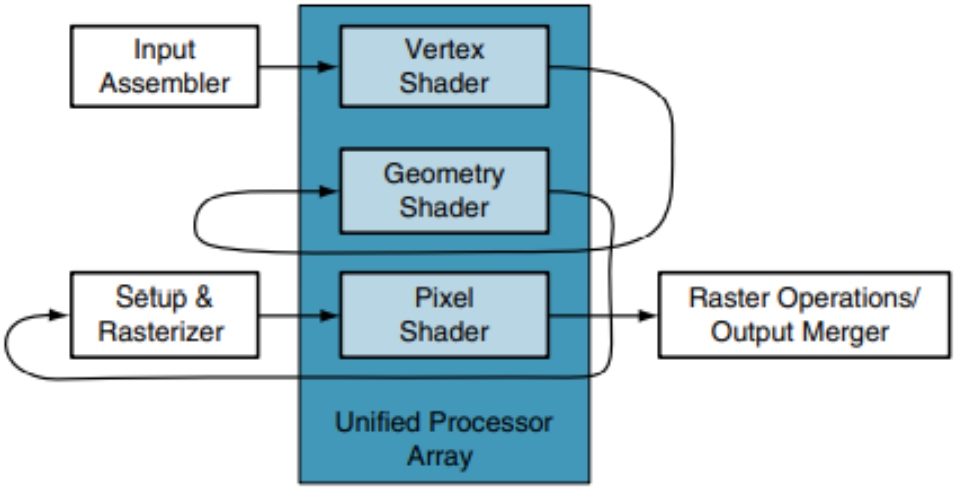
2.3. Pipeline cho logic đồ họa

Tiến trình pipeline cho logic đồ họa được thể hiện tại hình dưới, minh họa các quá trình xử lý chính, và đồng thời đánh dấu các bước quan trọng có thể được lập trình (các giai đoạn đổ bóng tại đỉnh, hình học, và điểm ảnh)

Hình 2. 3: Tiến trình của pipeline logic đồ họa (graphics logic pipeline)

2.4. Ánh xạ pipeline đồ họa tới Bộ xử lý GPU đồng nhất (Unified GPU Processor)

Hình ảnh dưới đây thể hiện pipeline logic, bao gồm các giai đoạn lập trình độc lập được ánh xạ tới mảng của các bộ xử lý, tới các bộ xử lý riêng biệt.



Hình 2. 4: Ánh xạ các pipeline logic tới các bộ xử lý nằm trong mảng của GPU

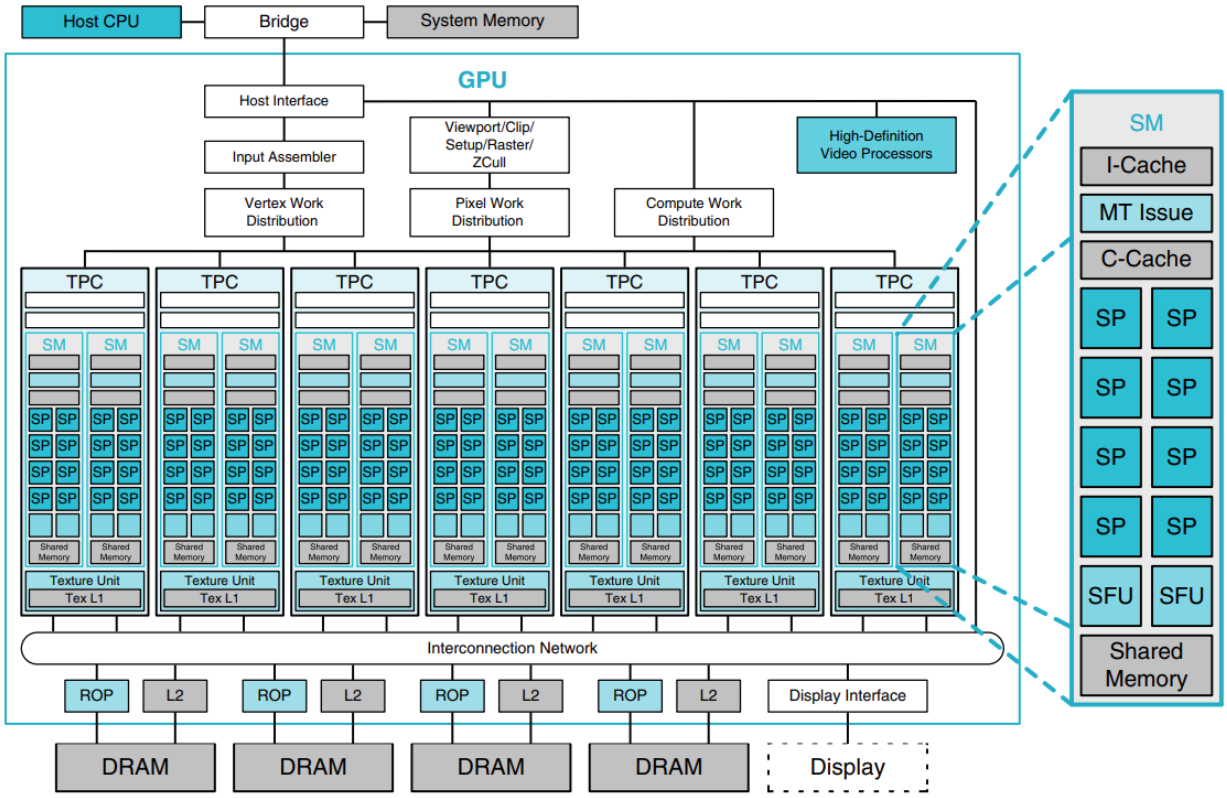
Kiến trúc bộ GPU đồng nhất cơ bản

Kiến trúc bộ GPU đồng nhất dựa trên các mảng song song gồm nhiều bộ xử lý có thể lập trình. Các bộ xử lý tính toán về đỉnh, hình học và điểm ảnh được thống nhất và đặt xử lý song song trên cùng một bộ xử lý, không giống như các bộ GPU đời trước chia riêng rẽ thành các bộ xử lý đặc thù tương ứng với mỗi loại. Mảng các bộ xử lý này được tích hợp chặt chẽ với các bộ xử lý cố định liên quan tới lọc kết cấu, điểm ảnh hóa, khử răng cưa, nén & giải nén, … Mặc dù các bộ xử lý cố định trên có hiệu năng tốt hơn rất nhiều so với các bộ xử lý có thể lập trình đa chức năng về các mặt diện tích sử dụng, chi phí, hoặc công suất sử dụng, xong chúng ta vẫn sẽ tập truing vào xây dựng các bộ xử lý có thể lập trình.

Bộ xử lý mảng

Một bộ xử lý mảng GPU đồng nhất chứa nhiều lõi xử lý, thường được tổ chức thành các bộ đa xử lý đa luồng. HÌnh dưới cho thấy môt bộ GPU với mảng gồm 112 lõi streaming processor (SP), được chia thành 14 đa luồng streaming multiprocessor (SMs). Mỗi lõi SP có khả năng đa luồng rất cao, điều khiển 96 luồng đồng thời cùng với trạng thái của chúng trong phần cứng. Các bộ xử lý kết

nối với 4 phân vùng DRAM rộng 64-bit thông qua mạng kết nối nội tại. Mỗi bộ SM bao gồm 8 lõi SP, cùng với hai bộ thực hiện chức năng đặc biệt (SFU), lệnh và bộ nhớ đệm không đổi, một đơn vị lện đa luồng và một bộ nhớ dùng chung. Kiến trúc này mang tên Kiến trúc Tesla, là kiến trúc được triển khai thực tế trong card Nvidia GeForce 8800. Bộ xử lý này có một kiến trúc đồng nhất, bao gồm các chương trình xử lý truyền thống cho các đỉnh, hình học, và điểm ảnh chạy trên các bộ SMs đồng nhất và các lõi SP, cùng với các chương trình tính toán chạy cùng trên một bộ xử lý.



Hình 2. 5: Ví dụ về cấu trúc của một bộ GPU đồng nhất

Kiến trúc bộ xử lý mảng có thể mở rộng cho các cấu hình GPU nhỏ hơn và lớn hơn, thông qua cách mở rộng số lượng bộ xử lý và số lượng phân vùng bộ nhớ. Hình trên cho thấy bảy cụm gồm hai SM chia sẻ một đơn vị kết cấu và một bộ đệm L1. Đơn vị kết cấu cung cấp kết quả đã qua lọc đến SM được cung cấp một tập hợp các tọa độ vào một bản đồ kết cấu. Vì các vùng hỗ trợ của bộ lọc thường chồng chéo lên nhau với các yêu cầu liên tiếp, một bộ đệm kết cấu L1 trực tuyến được sử dụng để giảm số lượng yêu cầu đến hệ thống bộ nhớ. Mảng bộ xử lý kết nối với bộ xử lý hoạt động raster (ROP), bộ nhớ đệm L2, bộ nhớ

DRAM bên ngoài và bộ nhớ hệ thống thông qua mạng kết nối toàn GPU. Số lượng bộ xử lý và số lượng bộ nhớ có thể mở rộng để thiết kế hệ thống GPU cân bằng cho các phân khúc thị trường và hiệu suất khác nhau.

3. LẬP TRÌNH GPU

Phần này mô tả ngắn gọn việc lập trình GPU cho các ứng dụng đồ họa thời gian thực bằng cách sử dụng API đồ họa và ngôn ngữ lập trình. Sau đó, tiếp tục mô tả việc lập trình GPU cho tính toán trực quan và các ứng dụng tính toán song song chung sử dụng ngôn ngữ C và mô hình lập trình CUDA.

3.1. Giới thiệu về lập trình GPU

Lập trình GPU đa xử lý khác là một quá trình khác biệt so với lập trình các bộ xử lý đa nhân như CPU đa lõi. GPU cung cấp độ lớn luồng và dữ liệu song song lớn hơn từ hai đến ba lần so với CPU, có thể mở rộng đến hàng trăm lõi bộ xử lý và hàng chục nghìn luồng đồng thời. Khả năng song song của GPU liên tục được cải thiện, tăng gấp đôi sau mỗi khoảng 12 cho tới 18 tháng (theo định luật Moore [1965] về việc tăng mật độ mạch tích hợp và bằng cách cải thiện hiệu quả kiến trúc). Để mở rộng phạm vi giá cả và hiệu suất của các phân khúc thị trường khác nhau, các sản phẩm GPU khác nhau triển khai số lượng bộ xử lý và luồng khác nhau. Tuy nhiên, phía người sử dụng luôn mong đợi các ứng dụng trò chơi, đồ họa, hình ảnh và máy tính hoạt động trên bất kỳ GPU nào, bất kể nó thực thi bao nhiêu luồng song song hoặc có bao nhiêu lõi xử lý song song, và các GPU đắt tiền hơn (với nhiều luồng và lõi hơn) luôn đi cùng với mong muốn ứng dụng được xử lý nhanh hơn. Do đó, các mô hình lập trình GPU và các chương trình ứng dụng được thiết kế để mở rộng quy mô theo cách minh bạch nhất, trên khoảng rộng khả năng xử lý song song.

Động lực đằng sau số lượng lớn các luồng và lõi song song trong GPU là hiệu suất đồ họa thời gian thực - nhu cầu hiển thị các cảnh 3D phức tạp với độ phân giải cao, tại tốc độ khung hình tương tác với người sử dụng ít nhất 60 khung hình/giây. Tương ứng, các mô hình lập trình có thể mở rộng của các ngôn ngữ đổ bóng đồ họa như Cg (C cho đồ họa) và HLSL (ngôn ngữ đổ bóng cấp cao) được thiết kế để khai thác mức độ song song lớn thông qua nhiều luồng song song độc lập, mở rộng đến bất kỳ số lượng lõi bộ xử lý nào. Mô hình lập trình song song có thể mở rộng CUDA cho phép các ứng dụng tính toán song song, tận dụng số lượng lớn các luồng song song và mở rộng quy mô đến bất kỳ số lượng lõi xử lý song song nào, minh bạch tới các ứng dụng chạy trên.

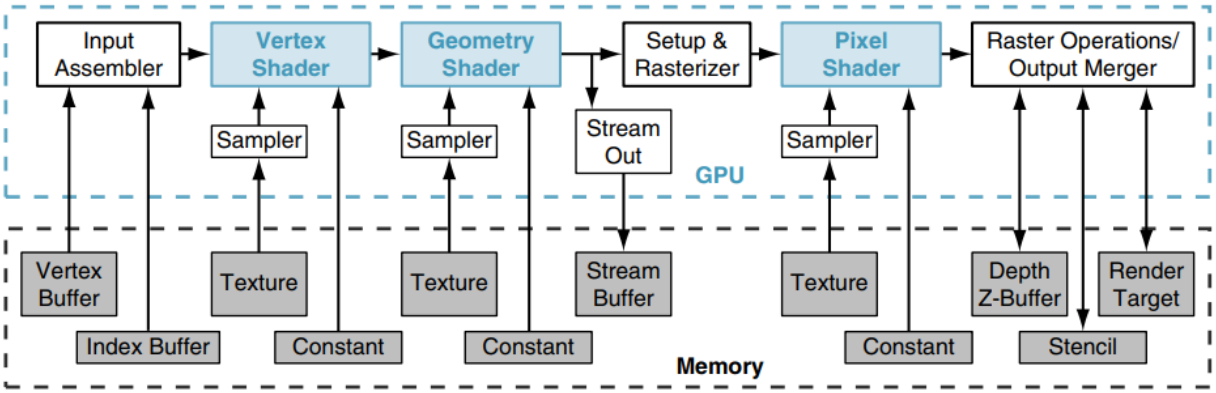
Trong các mô hình lập trình có thể mở rộng này, lập trình viên chỉ cnaf viết mã cho một luồng duy nhất, và GPU có thể chạy song song một số lượng lớn luồng. Do đó, các chương trình mở rộng quy mô một cách minh bạch trên một loạt các phần cứng song song. Mô hình đơn giản này hình thành từ các API đồ họa và ngôn ngữ đổ bóng mô tả cách đổ bóng cho một đỉnh hoặc một điểm ảnh. Đây vẫn là một mô hình hiệu quả, nhờ vào sự phát triển nhanh chóng của GPU trong khả năng song song và hiệu suất kể từ cuối những năm 1990.

Mục tiếp theo của phần 3 tiến hành mô tả ngắn gọn việc lập trình GPU cho các ứng dụng đồ họa thời gian thực bằng cách sử dụng API đồ họa và ngôn ngữ lập trình. Sau đó, tiếp tục mô tả việc lập trình GPU cho tính toán trực quan và các ứng dụng tính toán song song chung sử dụng ngôn ngữ C và mô hình lập trình CUDA.

3.2. Lập trình đồ họa thời gian thực

API đã đóng một vai trò quan trọng trong sự phát triển nhanh chóng và thành công của GPU và bộ xử lý. Hai API đồ họa tiêu chuẩn chính bao gồm OpenGL và Direct3D, là một trong những giao diện lập trình đa phương tiện Microsoft DirectX. OpenGL, một tiêu chuẩn mở, ban đầu được đề xuất và định nghĩa bởi Silicon Graphics Incorporated. Sự phát triển và mở rộng liên tục của tiêu chuẩn OpenGL [Segal và Akeley, 2006; Kessenich, 2006] được quản lý bởi Khronos, một tập đoàn công nghiệp. Direct3D [Blythe, 2006], được xác định và phát triển bởi Microsoft và các đối tác. OpenGL và Direct3D có cấu trúc tương tự nhau và tiếp tục phát triển nhanh chóng với những tiến bộ về phần cứng GPU. Họ xác định một pipeline xử lý logic đồ họa được ánh xạ vào phần cứng và bộ xử lý GPU, cùng với các mô hình và ngôn ngữ lập trình cho các giai đoạn đường ống có thể lập trình được.

3.3. Logical Graphics Pipeline



Hình 3. 1: Direct 3D 10 Graphics Pipeline

Hình trên minh họa Logical Graphics Pipeline Direct3D (cụ thể hơn là mô hình được sử dụng trong Direct3D 10). OpenGL có cấu trúc pipeline đồ họa tương tự. API và pipeline logic cung cấp cơ sở hạ tầng luồng dữ liệu trực tuyến và hệ thống plumbing cho các giai đoạn đổ bóng có thể lập trình, được hiển thị bằng màu xanh lam. Ứng dụng 3D sẽ gửi cho GPU một chuỗi các đỉnh được nhóm thành các nguyên thủy hình học – điểm, đường thẳng, hình tam giác và đa giác. Trình hợp ngữ đầu vào tiến hành thu thập các đỉnh và nguyên thủy.

Chương trình đổ bóng đỉnh thực hiện xử lý trên mỗi đỉnh, bao gồm ánh xạ vị trí 3D của đỉnh thành vị trí trên màn hình và chiếu sáng đỉnh để xác định màu của nó. Chương trình đổ bóng hình học thực hiện xử lý theo từng nguyên thủy, đồng thời có thể thêm hoặc bớt các nguyên thủy này. Chương trình pixel shader thực hiện xử lý từng mảnh, bao gồm nội suy các tham số trên mỗi mảnh, tạo họa tiết và tô màu. Pixel shader sử dụng rộng rãi các tra cứu được lấy mẫu và lọc thành các mảng 1D, 2D hoặc 3D lớn được gọi là kết cấu, sử dụng các tọa độ được nội suy.

API đồ họa và pipeline đồ họa cung cấp đầu vào, đầu ra, các đối tượng bộ nhớ và cơ sở hạ tầng cho các chương trình đổ bóng xử lý từng đỉnh, nguyên thủy và phân đoạn điểm ảnh.

3.4. Chương trình đồ họa đổ bóng:

Chúng ta có các khái niệm về Shader và Shading language:

- ***Shader:*** *Một chương trình hoạt động trên dữ liệu đồ họa như một đỉnh hoặc một đoạn các điểm ảnh.*
- ***Shading language:*** *Một ngôn ngữ kết xuất đồ họa, thường có một mô hình lập trình luồng dữ liệu hoặc luồng dữ liệu.*

Các ứng dụng đồ họa thời gian thực sử dụng nhiều chương trình đổ bóng khác nhau, mô hình hóa cách ánh sáng tương tác với các vật liệu khác nhau, hiển thị ánh sáng và bóng tối phức tạp. Các ngôn ngữ đổ bóng dựa trên các mô hình lập trình luồng dữ liệu hoặc luồng dữ liệu tương ứng với pipeline logic đồ họa. Các chương trình đổ bóng đỉnh ánh xạ vị trí của các đỉnh tam giác lên màn hình, thay đổi vị trí, màu sắc hoặc hướng của chúng. Thông thường, một luồng đổ bóng đỉnh nhập vào vị trí đỉnh (x, y, z, w) và tính toán vị trí màn hình (x, y, z). Các chương trình đổ bóng hình học hoạt động dựa trên các gốc hình học (chẳng hạn như đường thẳng và hình tam giác) được xác định bởi nhiều đỉnh, thay đổi chúng hoặc tạo ra các gốc bổ sung. Tính năng đổ bóng phân mảnh điểm ảnh - mỗi "bóng" một điểm ảnh, tính toán đóng góp màu đỏ, lục, lam, alpha (RGBA) vào hình ảnh được hiển thị tại vị trí hình ảnh mẫu điểm ảnh (x, y) của nó. Trình tạo bóng (và GPU) sử dụng số học dấu phẩy động cho tất cả các phép tính màu điểm ảnh để loại bỏ các yếu tố tạo tác có thể nhìn thấy được, trong khi tính toán với phạm vi cực lớn của các giá trị đóng góp điểm ảnh gặp phải, đặc biệt khi hiển thị các cảnh có ánh sáng, bóng tối phức tạp và phạm vi dải động của ảnh sáng cao. Đối với cả ba loại trình đổ bóng đồ họa, nhiều phiên bản chương trình có thể chạy song song, dưới dạng các luồng song song độc lập. Vì mỗi luồng hoạt động trên dữ liệu độc lập, các chương trình này tạo ra các kết quả độc lập và không có tác động qua lại lẫn nhau. Các đỉnh, gốc và điểm ảnh độc lập tiếp tục cho phép cùng một chương trình đồ họa chạy trên các GPU có kích thước khác nhau để xử lý song song số lượng đỉnh, gốc và điểm ảnh khác nhau. Do đó, các chương trình đồ họa mở rộng quy mô một cách rõ ràng đến GPU với số lượng song song và hiệu suất khác nhau.

Người dùng tiến hành lập trình cả ba luồng logic đồ họa bằng một ngôn ngữ cấp cao, hướng tới một mục tiêu chung. HLSL (high-level shading language - ngôn ngữ đổ bóng cấp cao) và Cg (C cho đồ họa) thường được sử dụng. Ngôn ngữ này có cú pháp câu lệnh giống với câu lệnh của C và một tập các thư viện phong phú, cho các phép toán ma trận, lượng giác, nội suy và truy cập và lọc kết cấu, nhưng khác xa với các ngôn ngữ máy tính thông thường. Các câu lệnh hiện thiếu quyền truy cập bộ nhớ chung, con trỏ, tệp I / O và đệ quy. HLSL và Cg giả

định rằng các chương trình chỉ hoạt động trong một pipeline logic đồ họa, và do đó I/O là ngầm định.

Phần cứng GPU tạo ra một luồng độc lập mới để thực thi một chương trình đổ bóng đỉnh, hình học hoặc điểm ảnh cho mọi đỉnh, mọi nguyên thủy và mọi phân đoạn điểm ảnh. Trong trò chơi điện tử, phần lớn các luồng thực thi là của các chương trình tạo bóng điểm ảnh, vì thường có nhiều đoạn điểm ảnh gấp 10 đến 20 lần so với các đỉnh và ánh sáng và bóng phức tạp đòi hỏi tỷ lệ điểm ảnh trên các luồng đổ bóng đỉnh thậm chí còn lớn hơn. Mô hình lập trình đổ bóng đồ họa thúc đẩy kiến trúc GPU thực thi hiệu quả hàng nghìn luồng phân tích nhỏ độc lập, thực hiện trên nhiều lõi xử lý song song.

Ví dụ về Pixel Shader

Hình 3. 2 cho thấy da được hiển thị bằng bộ đổ bóng điểm ảnh phân mảnh. Da thật xuất hiện khá khác so với sơn màu da thịt vì ánh sáng phản xạ xung quanh rất nhiều trước khi tái xuất hiện. Trong bộ đổ bóng phức tạp này, ba lớp da riêng biệt, mỗi lớp có phương án tính toán tán xạ dưới bề mặt riêng biệt, được mô hình hóa để tạo độ sâu và độ trong mờ cho da. Sự tán xạ có thể được mô hình hóa bằng một tổ hợp làm mờ trong không gian "kết cấu" được nâng cao, với màu đỏ bị làm mờ nhiều hơn màu xanh lá cây và màu xanh lam bị làm mờ ít hơn. Trình đổ bóng Cg đã biên dịch thực hiện 1400 hướng dẫn để tính toán màu cho mỗi một điểm ảnh trên da.

Hình 3. 2: Hình ảnh được sinh ra bởi GPU, sử dụng để mô tả khuôn mặt người

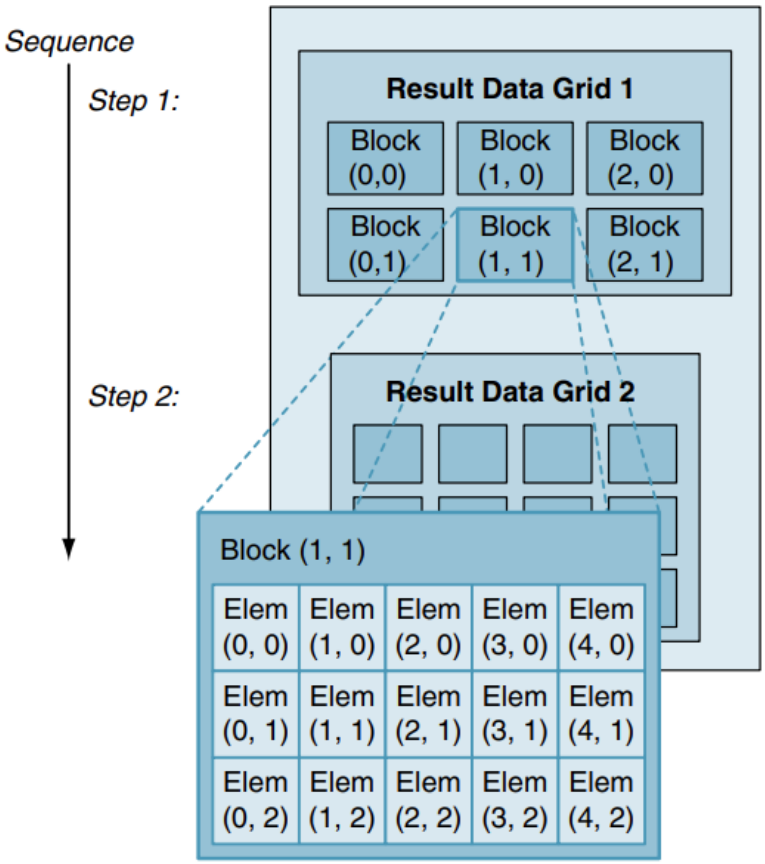
Khi GPU đã phát triển hiệu suất tính toán với dấu phẩy động vượt trội, cùng với băng thông bộ nhớ phát trực tuyến rất cao cho đồ họa thời gian thực, GPU đã thu hút được thêm nhiều ứng dụng yêu cầu song song cao bên cạnh đồ họa truyền thống. Ban đầu, việc truy cập vào sức mạnh này chỉ có thực hiện thông qua việc đặt một ứng dụng làm thuật toán kết xuất đồ họa, nhưng cách tiếp cận thường gặp nhiều khó khăn và hạn chế. Gần đây hơn, mô hình lập trình CUDA đã cung cấp một cách dễ dàng hơn nhiều để khai thác băng thông bộ nhớ và dấu chấm động hiệu suất cao có thể mở rộng của GPU với ngôn ngữ lập trình C.

3.5. Lập trình ứng dụng tính toán song song

CUDA, Brook và CAL là các giao diện lập trình cho GPU tập trung vào tính toán song song dữ liệu hơn là xử lý đồ họa. CAL (Lớp trừu tượng tính toán) là một giao diện ngôn ngữ trình hợp dịch cấp thấp cho GPU AMD. Brook là một

ngôn ngữ phát trực tuyến được điều chỉnh cho GPU. CUDA được phát triển bởi là một phần mở rộng cho ngôn ngữ C và C ++, phục vụ mục đích lập trình song song có thể mở rộng cho GPU nhiều lõi và CPU đa lõi. Với các mô hình mới, GPU sở hữu khả năng vượt trội trong tính toán thông lượng và song song dữ liệu, thực thi các ứng dụng tính toán hiệu suất cao cũng như các ứng dụng đồ họa.

Phân tích vấn đề song song dữ liệu:



Hình 3. 3: Phân tích khối dữ liệu thành các phần tử nhỏ hơn để tính toán song song

Để ánh xạ các vấn đề tính toán lớn một cách hiệu quả tới một kiến trúc xử lý song song cao, lập trình viên hoặc trình biên dịch tiến hành phân tách vấn đề thành nhiều vấn đề nhỏ có thể giải quyết song song.

Ví dụ: *lập trình viên phân vùng dữ liệu kết quả lớn thành các khối và tiếp tục phân vùng mỗi khối thành các phần tử, sao cho các khối kết quả có thể được tính toán song song một cách độc lập, và các phần tử trong mỗi khối cũng được tính*

song song. Hình trên cho thấy sự phân rã của một mảng dữ liệu kết quả thành một lưới các khối 3 x 2, trong đó mỗi khối tiếp tục được phân rã thành một mảng 5 x 3 các phần tử. Việc phân tách song song hai cấp ánh xạ tới kiến trúc GPU: các bộ xử lý đa xử lý song song tính toán các khối và các luồng song song tính toán các phần tử. Chúng có thể được tính song song một cách độc lập.

3.6. Mô hình CUDA

CUDA là một phần mở rộng tối thiểu của ngôn ngữ lập trình C và C ++. Người lập trình viết một chương trình nối tiếp gọi các nhân song song, có thể là các hàm đơn giản hoặc các chương trình đầy đủ. Một nhân thực thi song song trên một tập hợp các luồng song song. Người lập trình tổ chức các luồng này thành một hệ thống phân cấp các khối luồng và lưới các khối luồng. Một khối luồng là một tập hợp các luồng đồng thời thực hiện cùng một chương trình luồng và có thể hợp tác để tính toán một kết quả. Lưới là một tập hợp các khối luồng mà mỗi khối có thể được thực thi độc lập và do đó có thể thực thi song song.

Khi gọi một hạt nhân, lập trình viên chỉ định số luồng trên mỗi khối và số khối bao gồm lưới. Mỗi luồng được cấp một số ID luồng duy nhất threadIdx trong khối luồng của nó, được đánh số 0, 1. 2, ..., blockDim-1 và mỗi khối luồng được cấp một số ID khối duy nhất blockIdx trong lưới của nó. CUDA hỗ trợ các khối luồng chứa tối đa 512 luồng. Để thuận tiện, các khối và lưới luồng có thể có một, hai hoặc ba thứ nguyên, được truy cập thông qua các trường chỉ mục .x, .y và .z.

Hình 3. 4 mô tả mã tuần tự (trên cùng) trong C so với mã song song (dưới cùng) trong CUDA cho SAXPY. Các luồng song song CUDA thay thế vòng lặp nối tiếp C – mỗi luồng tính kết quả giống như một lần lặp vòng lặp. Mã song song tính toán n kết quả với n luồng được tổ chức thành các khối 256 luồng.

Computing y = ax + y with a serial loop:

```
void saxpy_serial(int n, float alpha, float *x, float *y)
{
     for(int i = 0; i<n; ++i)
          y[i] = alpha*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

Computing y = ax + y in parallel using CUDA:

```
__global__
void saxpy_parallel(int n, float alpha, float *x, float *y)
{
     int i = blockIdx.x*blockDim.x + threadIdx.x;

     if( i<n ) y[i] = alpha*x[i] + y[i];
}

// Invoke parallel SAXPY kernel (256 threads per block)
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Hình 3. 4: So sánh tính toán tuần tự & tính toán song song với CUDA

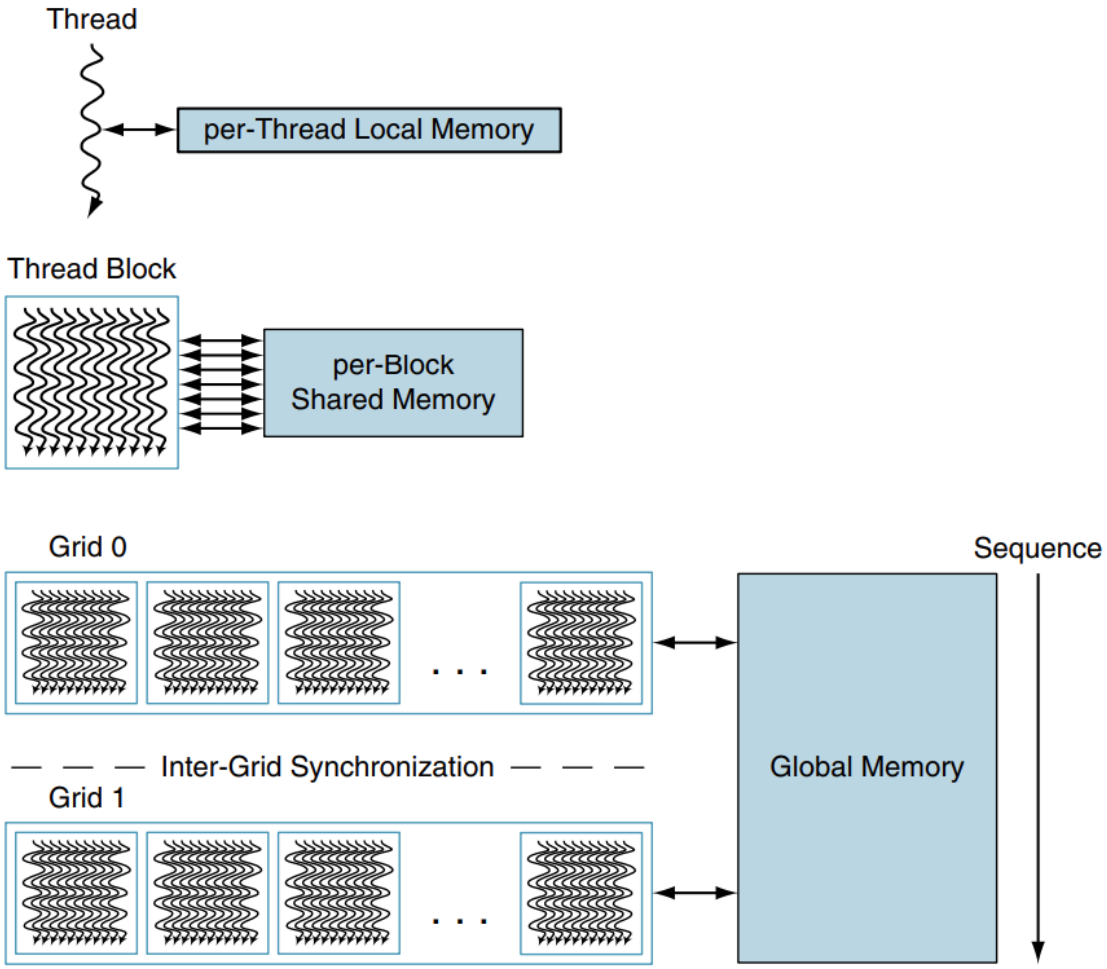
Việc thực hiện song song và quản lý luồng là tự động. Tất cả việc tạo luồng, lập lịch và kết thúc được xử lý cho lập trình viên bởi hệ thống cơ bản. Các luồng của một khối thực thi đồng thời và có thể đồng bộ hóa tại một rào cản đồng bộ hóa bằng cách gọi nội tại __syncthreads(). Do đó, các luồng trong một khối có thể giao tiếp với nhau bằng cách ghi và đọc bộ nhớ dùng chung trên mỗi khối tại một rào cản đồng bộ hóa.

Vì các luồng trong một khối có thể chia sẻ bộ nhớ và đồng bộ hóa qua các rào cản, chúng sẽ nằm cùng nhau trên cùng một bộ xử lý vật lý hoặc đa xử lý. Tuy nhiên, số lượng khối luồng có thể vượt quá số bộ xử lý. Mô hình lập trình luồng CUDA ảo hóa các bộ xử lý và cung cấp cho lập trình viên sự linh hoạt để song song hóa ở bất kỳ mức độ chi tiết nào là thuận tiện nhất. Ảo hóa thành các luồng và khối cho phép phân rã vấn đề một cách trực quan. Nó cũng cho phép cùng một chương trình CUDA mở rộng quy mô đến số lượng lõi xử lý khác nhau.

Để quản lý ảo hóa phần tử xử lý này và cung cấp khả năng mở rộng, CUDA yêu cầu các khối luồng có thể thực thi độc lập. Yêu cầu độc lập này cho phép các khối luồng được lập lịch theo bất kỳ thứ tự nào trên bất kỳ số lõi nào, làm cho mô hình CUDA có thể mở rộng trên một số lõi tùy ý cũng như trên nhiều kiến trúc song song. Nó cũng giúp tránh khả năng bị bế tắc. Một ứng dụng có thể thực thi nhiều lưới độc lập hoặc phụ thuộc. Các lưới độc lập có thể thực thi đồng thời nếu có đủ tài nguyên phần cứng. Các lưới phụ thuộc thực thi tuần tự, với một rào cản ngầm giữa chúng, do đó đảm bảo rằng tất cả các khối của lưới thứ nhất hoàn thành trước khi bất kỳ khối nào của lưới phụ thuộc thứ hai bắt đầu.

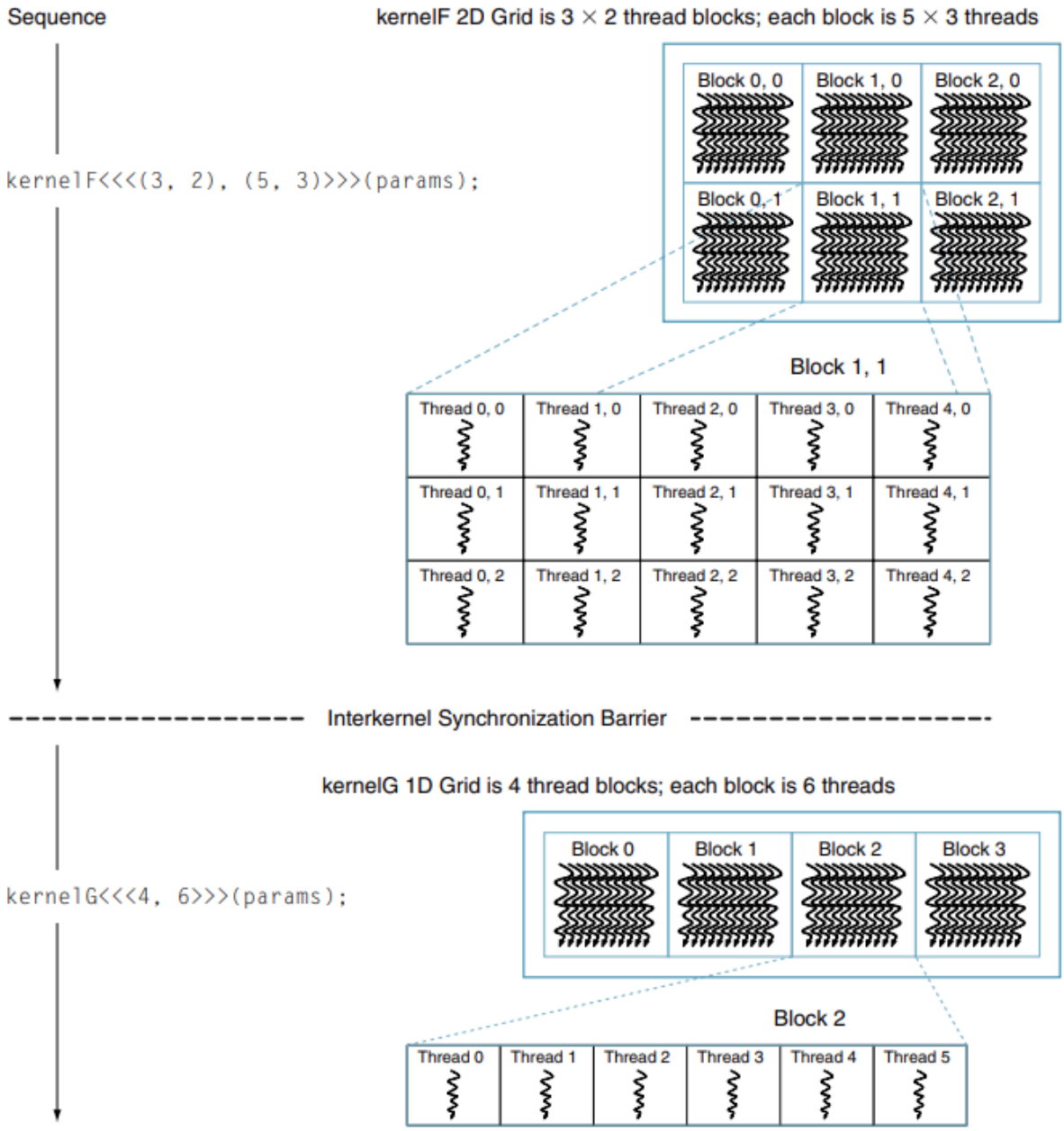
Các luồng có thể truy cập dữ liệu từ nhiều không gian bộ nhớ trong quá trình thực thi của chúng. Mỗi luồng có một bộ nhớ cục bộ riêng. CUDA sử dụng bộ nhớ cục bộ cho các biến riêng tư của luồng không phù hợp với các thanh ghi của luồng, cũng như cho các khung ngăn xếp và tràn thanh ghi. Mỗi khối luồng có một bộ nhớ dùng chung, hiển thị cho tất cả các luồng của khối, có cùng thời gian tồn tại với khối. Cuối cùng, tất cả các luồng đều có quyền truy cập vào cùng một bộ nhớ chung.

Hình 3. 5 trình bày sơ đồ về các mức lồng nhau của luồng, khối luồng và lưới của các khối luồng. Nó cho thấy thêm các mức độ chia sẻ bộ nhớ tương ứng: bộ nhớ cục bộ, bộ nhớ được chia sẻ và toàn cục để chia sẻ dữ liệu trên mỗi luồng, mỗi khối và mỗi ứng dụng.



Hình 3. 5: Mức độ lồng nhau của các luồng, các khối luồng và các lưới của khối luồng

Mô hình lập trình CUDA thể hiện tính song song một cách rõ ràng và mỗi hạt nhân thực thi trên một số luồng cố định. CUDA linh hoạt vì mỗi lệnh gọi hạt nhân sẽ tự động tạo ra một lưới mới với đúng số khối luồng và luồng cho bước ứng dụng đó. Lập trình viên có thể sử dụng mức độ song song thuận tiện cho mỗi nhân, thay vì phải thiết kế tất cả các giai đoạn của tính toán để sử dụng cùng một số luồng. **Error! Reference source not found.** cho thấy một ví dụ về chuỗi mã CUDA: F khởi tạo trên lưới 2D gồm các khối luồng 2D, một rào cản đồng bộ hóa giữa các kênh, tiếp theo là hạt nhân G trên lưới 1D gồm các khối luồng 1D.



Hình 3. 6: Ví dụ về một chuỗi mã CUDA

Lập trình song song có thể mở rộng với CUDA

Mô hình lập trình song song có thể mở rộng - CUDA mở rộng ngôn ngữ C & C ++, nhằm khai thác mức độ song song lớn cho các ứng dụng chung, trên các bộ đa xử lý có độ song song cao mà đặc biệt là GPU.

Các trải nghiệm ban đầu với CUDA cho thấy rằng nhiều chương trình phức tạp có thể được thể hiện một cách dễ dàng chỉ với một vài lớp phân tách đã được hiểu rõ ràng. Các ứng dụng bao gồm xử lý dữ liệu địa chấn, hóa học tính toán, đại số tuyến tính, .v..v trực quan đã mở rộng quy mô một cách minh bạch tới hàng trăm lõi bộ xử lý và hàng nghìn luồng đồng thời. Mô hình CUDA cũng có thể áp dụng cho các kiến trúc xử lý song song bộ nhớ dùng chung khác, bao gồm cả CPU đa lõi.

CUDA cung cấp ba phần chính, cung cấp một cấu trúc song song rõ ràng tới mã lệnh C thông thường cho mỗi luồng của hệ thống phân cấp:

- Một hệ thống phân cấp của các nhóm luồng
- Bộ nhớ được chia sẻ
- Đồng bộ hóa rào cản

Những hạn chế với mô hình CUDA

Để việc triển khai được hiệu quả và đơn giản hơn, mô hình lập trình CUDA có một số hạn chế. Các luồng và khối luồng chỉ có thể được tạo ra bằng cách gọi một nhân song song, không phải từ bên trong một nhân song song. Cùng với sự độc lập bắt buộc của các khối luồng, điều này giúp việc thực thi các chương trình CUDA với một bộ lập lịch đơn giản sử dụng chi phí thời gian chạy tối thiểu. Trên thực tế, kiến trúc GPU Tesla thực hiện quản lý *phần cứng* và lập lịch cho các luồng và khối luồng.

Tính song song của nhiệm vụ có thể được thể hiện ở mức khối luồng, nhưng khó thể hiện trong một khối luồng vì các rào cản đồng bộ hóa luồng hoạt động trên tất cả các luồng của khối. Để các chương trình CUDA có thể chạy trên bất kỳ số lượng bộ xử lý nào, các khối luồng trong cùng một lưới hạt nhân không được phép phụ thuộc vào nhau, mà các khối phải thực thi độc lập. Vì CUDA yêu cầu khối luồng phải độc lập và cho phép các khối được thực thi theo bất kỳ thứ tự nào, việc kết hợp các kết quả được tạo bởi nhiều khối phải được thực hiện bằng cách khởi chạy hạt nhân thứ hai trên một lưới các khối luồng mới (mặc dù các khối luồng có thể điều phối các hoạt động của chúng, bằng cách sử dụng các hoạt động của bộ nhớ nguyên tử trên bộ nhớ chung hiển thị cho tất cả các luồng - ví dụ: bằng cách tăng nguyên tử các con trỏ hàng đợi).

Các lệnh gọi hàm đệ quy hiện không được sử dụng trong nhân CUDA. Đệ quy không phù hợp trong một nhân song song lớn, bởi vì việc cung cấp không gian ngăn xếp cho hàng chục nghìn luồng có thể đang hoạt động sẽ yêu cầu một lượng rất lớn bộ nhớ. Các thuật toán nối tiếp thường được biểu diễn bằng cách sử dụng đệ quy, chẳng hạn như quicksort, được triển khai tốt nhất bằng cách sử dụng song song dữ liệu lồng nhau thay vì đệ quy.

Để hỗ trợ kiến trúc hệ thống hỗn hợp kết hợp CPU và GPU, với mỗi hệ thống có hệ thống bộ nhớ riêng, các chương trình CUDA phải sao chép dữ liệu và kết quả giữa bộ nhớ chủ và bộ nhớ thiết bị. Chi phí tương tác giữa CPU-GPU và truyền dữ liệu được giảm thiểu bằng cách sử dụng công cụ truyền khối DMA và kết nối nhanh. Các vấn đề máy tính đủ lớn cần tăng hiệu suất GPU được phân bổ chi phí tốt hơn các vấn đề nhỏ.

3.7. Những hàm ý khi xây dựng kiến trúc (Implications for Architecture)

Các mô hình lập trình song song cho đồ họa và tính toán đã thúc đẩy sự khác biệt giữa kiến trúc GPU so với kiến trúc CPU. Các khía cạnh chính của các chương trình GPU thúc đẩy kiến trúc bộ xử lý GPU là:

- Sử dụng rộng rãi tính song song dữ liệu
- Mô hình lập trình phân luồng cao
- Có khả năng mở rộng: Có thể tự tăng hiệu suất khi được cung cấp thêm bộ xử lý mà không cần biên dịch lại
- Tính toán dấu phẩy động (hoặc số nguyên)
- Hỗ trợ tính toán thông lượng cao

4. KIẾN TRÚC XỬ LÝ ĐA LUỒNG

GPU là bộ xử lý đa năng bao gồm nhiều bộ xử lý, mỗi bộ xử lý đa luồng đều có tính đa luồng cao. Một GPU chất lượng cơ bản nhất có từ hai đến bốn bộ vi xử lý, trong khi GPU hoặc nền tảng máy tính của người đam mê chơi game có hàng chục bộ xử lý.

Thiết kế đa xử lý đa luồng được mô tả ở đây có tám lõi xử lý vô hướng trong một kiến trúc được kết hợp chặt chẽ và thực thi tối đa 512 luồng. Để tiết kiệm diện tích và năng lượng, bộ đa xử lý chia sẻ các đơn vị phức tạp lớn trong số tám lõi bộ xử lý, bao gồm bộ nhớ đệm lệnh, đơn vị lệnh đa luồng và RAM bộ nhớ dùng chung.

4.1. Đa luồng lớn

Bộ xử lý GPU được thiết kế với khả năng đa luồng cao để đạt được một số mục tiêu:

- Che mở độ trễ khi tải bộ nhớ và tải kết cấu từ DRAM
- Hỗ trợ các mô hình lập trình đổ bóng đồ họa song song chi tiết
- Hỗ trợ các mô hình lập trình tính toán song song chi tiết
- Ảo hóa các bộ xử lý vật lý dưới dạng luồng và khối luồng để cung cấp sự trong suốt tới khả năng mở rộng
- Đơn giản hóa mô hình lập trình song song để viết một chương trình nối tiếp cho một luồng

Độ trễ trong tìm nạp bộ nhớ và kết cấu có thể yêu cầu hàng trăm xung nhịp của bộ xử lý, bởi vì GPU thường có bộ nhớ đệm phát trực tuyến nhỏ hơn là bộ nhớ đệm của CPU. Một yêu cầu tìm nạp thường yêu cầu độ trễ truy cập DRAM đầy đủ cộng với độ trễ kết nối và bộ đệm. Đa luồng giúp che phủ độ trễ bằng tính toán hữu ích - trong khi một luồng đang đợi tải hoặc tìm nạp kết cấu hoàn tất, bộ xử lý có thể thực thi một luồng khác. Các mô hình lập trình song song chi tiết nhỏ cung cấp hàng nghìn luồng độc lập có thể giữ cho nhiều bộ xử lý bận rộn, mặc dù độ trễ bộ nhớ dài khi theo góc nhìn của từng luồng độc lập.

Chương trình đồ họa đỉnh hoặc pixel shader là một chương trình cho một luồng xử lý một đỉnh hoặc một điểm ảnh. Tương tự, một chương trình CUDA là một

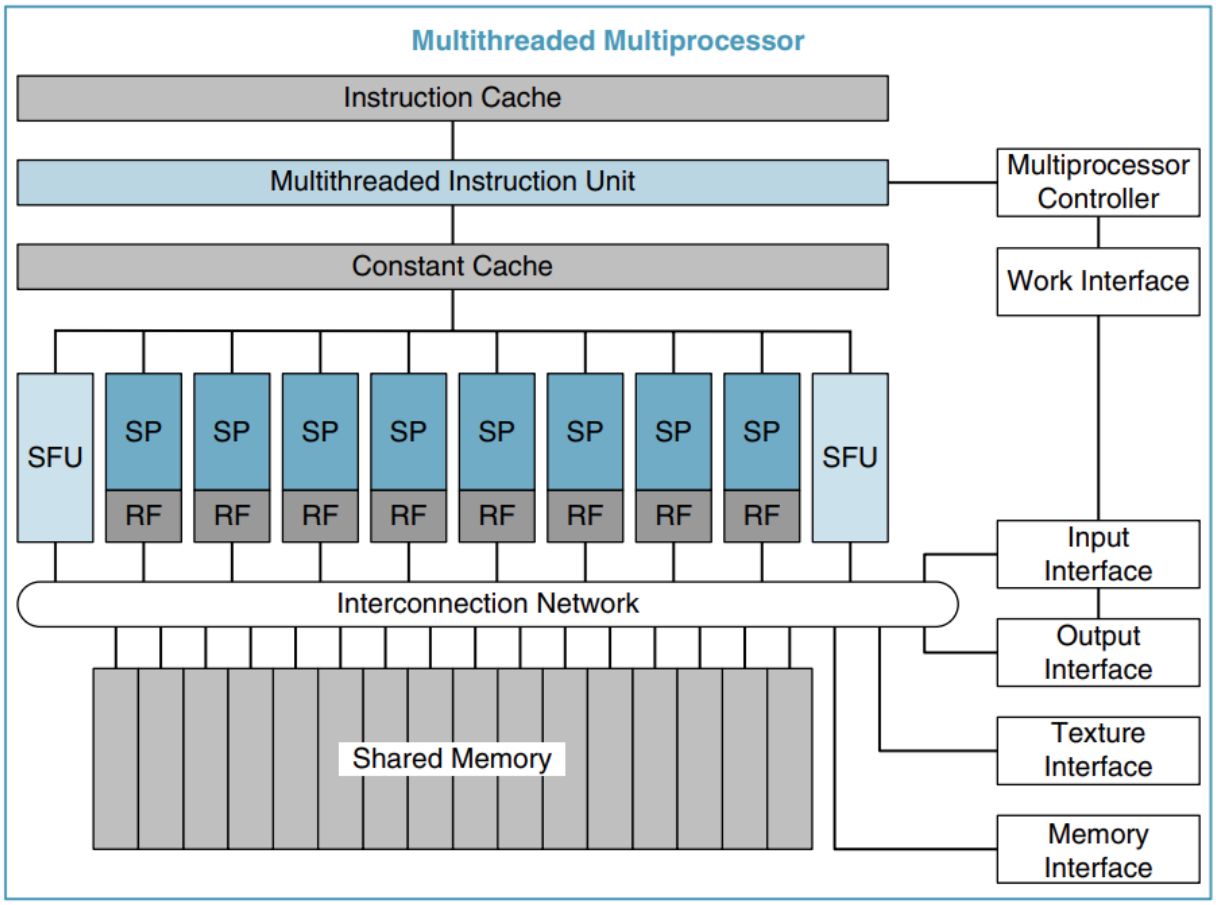
chương trình C cho một luồng đơn tính toán một kết quả. Các chương trình đồ họa và máy tính khởi tạo nhiều luồng song song để hiển thị các hình ảnh phức tạp và tính toán các mảng kết quả lớn. Để cân bằng động khối lượng công việc của luồng chuyển đổi đỉnh và trình tạo bóng điểm ảnh, mỗi bộ xử lý đa xử lý đồng thời thực thi nhiều chương trình luồng khác nhau và các loại chương trình tạo bóng khác nhau.

Để hỗ trợ mô hình lập trình đỉnh, nguyên thủy và điểm ảnh độc lập của các ngôn ngữ đổ bóng đồ họa và mô hình lập trình luồng đơn của CUDA C/C + +, mỗi luồng GPU đều có các thanh ghi riêng, bộ nhớ riêng, bộ đếm chương trình và thực thi luồng trạng thái, và có thể thực thi một đường dẫn mã độc lập. Để thực thi hiệu quả hàng trăm luồng nhẹ đồng thời, bộ đa xử lý GPU là phần cứng đa luồng - nó quản lý và thực thi hàng trăm luồng đồng thời trong phần cứng mà không cần lên lịch chi phí. Các luồng đồng thời trong các khối luồng có thể đồng bộ hóa tại một rào cản với một lệnh duy nhất. Tạo luồng nhẹ, lập lịch luồng không đầu nối và đồng bộ hóa rào cản nhanh chóng hỗ trợ hiệu quả song song rất chi tiết.

4.2. Kiến trúc đa xử lý

Một bộ xử lý đa xử lý đồ họa và tính toán đồng nhất thực thi các chương trình tạo đỉnh, hình học và phân mảnh điểm ảnh và các chương trình tính toán song song. Như Hình 4. 1 cho thấy, bộ đa xử lý mẫu bao gồm tám lõi của bộ xử lý vô hướng (SP), mỗi lõi có một tệp thanh ghi đa luồng lớn (RF), hai đơn vị chức năng

đặc biệt (SFU), một đơn vị lệnh đa luồng, một bộ đệm lệnh, một bộ đọc- chỉ bộ nhớ cache không đổi và một bộ nhớ được chia sẻ.



Hình 4. 1: Bộ đa xử lý đa luồng với 8 lõi SP

SP là phần cứng đa luồng, hỗ trợ lên đến 64 luồng. Mỗi lõi SP pipelined thực hiện một lệnh vô hướng cho mỗi luồng trên mỗi xung nhịp, dao động từ 1.2 GHz đến 1.6 GHz trong các sản phẩm GPU khác nhau. Mỗi lõi SP có một RF lớn 1024 thanh ghi 32-bit cho mục đích chung, được phân vùng giữa các luồng được chỉ định của nó. SP có thể chạy đồng thời nhiều luồng sử dụng một vài thanh ghi hoặc ít luồng sử dụng nhiều thanh ghi hơn. Trình biên dịch tối ưu hóa phân bổ thanh ghi để cân bằng giữa chi phí tràn thanh ghi so với chi phí của ít luồng hơn. Các chương trình CUDA đã biên dịch thường cần 32 thanh ghi cho mỗi luồng, giới hạn mỗi SP là 32 luồng, điều này giới hạn một chương trình nhân như vậy là 256 luồng cho mỗi khối luồng trên bộ xử lý đa năng ví dụ này, thay vì tối đa 512 luồng.

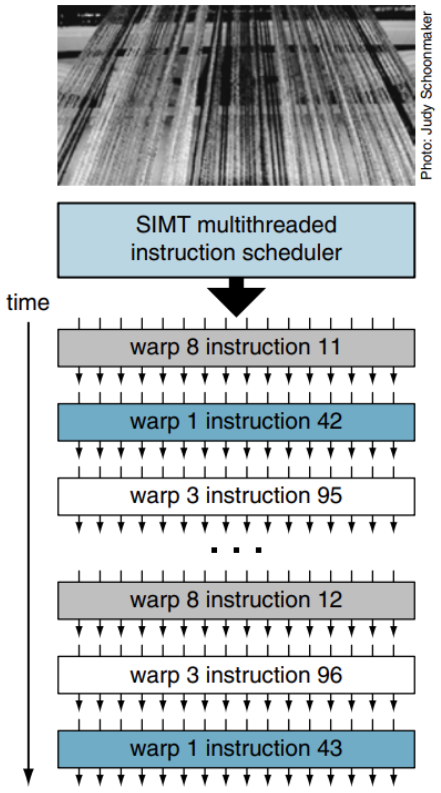
Các SFU pipelined thực thi các lệnh luồng tính toán các chức năng đặc biệt và nội suy các thuộc tính pixel từ các thuộc tính đỉnh nguyên thủy. Các lệnh này có thể thực hiện đồng thời với các lệnh trên SP.

Bộ đa xử lý thực hiện các lệnh tìm nạp kết cấu trên đơn vị kết cấu thông qua giao diện kết cấu và sử dụng giao diện bộ nhớ cho các lệnh tải, lưu trữ và truy cập nguyên tử bộ nhớ ngoài. Các lệnh này có thể thực hiện đồng thời với các lệnh trên SP. Truy cập bộ nhớ dùng chung sử dụng mạng kết nối có độ trễ thấp giữa bộ xử lý SP và ngân hàng bộ nhớ dùng chung.

4.3. Signal-Instruction Multiple-Thread (SIMT)

Signal-Instruction Multiple-Thread (SIMT) Một kiến trúc bộ xử lý áp dụng song song một lệnh cho nhiều luồng độc lập. Nó tạo, quản lý, lập lịch và thực thi các luồng đồng thời trong các nhóm luồng song song được gọi là warps.

Hình dưới thể hiện bộ lập lịch chọn một sợi dọc đã sẵn sàng và đưa ra lệnh một cách đồng bộ cho các luồng song song tạo nên sợi dọc. Bởi vì các sợi dọc là độc lập, bộ lập lịch có thể chọn một sợi dọc khác nhau mỗi lần.



Hình 4. 2: Kiến trúc bộ xử lý SIMT

Kiến trúc bộ xử lý SIMT áp dụng một lệnh cho nhiều luồng độc lập song song, không chỉ lệnh cho bộ xử lý SIMT điều khiển một luồng riêng lẻ và đơn vị lệnh SIMT đưa ra lệnh cho một sợi dọc song song độc lập để đạt hiệu quả. Bộ xử lý SIMT tìm sự song song mức dữ liệu giữa các luồng trong thời gian chạy, tương tự như cách một bộ xử lý siêu cấp tìm thấy sự song song mức chỉ lệnh giữa các lệnh trong thời gian chạy.

Bộ xử lý SIMT nhận ra hiệu quả và hiệu suất đầy đủ khi tất cả các luồng của sợi dọc có cùng một đường dẫn thực thi. Nếu các luồng của một sợi dọc phân kỳ qua một nhánh có điều kiện phụ thuộc vào dữ liệu, việc thực thi sẽ tuần tự hóa cho từng đường dẫn nhánh được thực hiện và khi tất cả các đường dẫn hoàn tất, các luồng sẽ hội tụ đến cùng một đường dẫn thực thi. Đối với các đường dẫn có độ dài bằng nhau, bộ đa xử lý sử dụng ngăn xếp đồng bộ hóa nhánh để quản lý các luồng độc lập phân kỳ và hội tụ. Các đoạn mã khác nhau thực thi độc lập với tốc độ tối đa bất kể chúng đang thực thi các đường dẫn mã chung hay riêng biệt. Kết quả là, GPU SIMT hiệu quả hơn và linh hoạt hơn về mã phân nhánh so với các GPU trước đó, vì độ cong của chúng hẹp hơn nhiều so với chiều rộng SIMD của các GPU trước.

Sự phân kì và thực thi SIMT

Phương pháp SIMT lập lịch cho các sợi cong độc lập linh hoạt hơn cách lập lịch cho các kiến trúc GPU trước đây. Một sợi dọc bao gồm các luồng song song cùng loại: đỉnh, hình học, pixel hoặc tính toán. Đơn vị cơ bản của xử lý bộ đổ bóng phân mảnh pixel là bộ tứ pixel 2 x 2 được triển khai dưới dạng bốn chủ đề bộ đổ bóng pixel. Bộ điều khiển đa xử lý đóng gói các đơn vị pixel thành một sợi dọc. Tương tự, nó nhóm các đỉnh và gốc thành các sợi dọc, và đóng gói các luồng tính toán thành một sợi dọc. Một khối chủ đề bao gồm một hoặc nhiều sợi dọc. Thiết kế SIMT chia sẻ đơn vị tìm nạp và phát hành lệnh một cách hiệu quả trên các luồng song song của một sợi dọc, nhưng yêu cầu một dải đầy đủ các luồng đang hoạt động để có được hiệu suất hoạt động đầy đủ.

Bộ xử lý hợp nhất này lên lịch và thực thi đồng thời nhiều loại sợi dọc, cho phép nó thực hiện đồng thời các đoạn dọc đỉnh và điểm ảnh. Bộ lập lịch dọc của nó hoạt động ở mức thấp hơn tốc độ xung nhịp của bộ xử lý, vì có bốn làn luồng trên mỗi lõi bộ xử lý. Trong mỗi chu kỳ lập lịch, nó chọn một sợi dọc để thực hiện lệnh dọc SIMT, như thể hiện trong Hình 4.2. Một lệnh dọc đã phát hành thực

thi dưới dạng bốn bộ tám luồng trong bốn chu kỳ thông lượng của bộ xử lý. Đường ống bộ xử lý sử dụng một số đồng hồ độ trễ để hoàn thành mỗi lệnh. Nếu con số của độ trễ hoạt động nhân với số đồng hồ trên mỗi sợi dọc vượt quá độ trễ của đường ống, thì lập trình viên có thể bỏ qua độ trễ của đường ống. Đối với bộ đa xử lý này, một lịch trình quay vòng gồm tám sợi dọc có khoảng thời gian 32 chu kỳ giữa các lệnh liên tiếp cho cùng một sợi dọc. Nếu chương trình có thể giữ cho 256 luồng hoạt động trên mỗi bộ đa xử lý, thì độ trễ lệnh lên đến 32 chu kỳ có thể bị ẩn khỏi một luồng tuần tự riêng lẻ. Tuy nhiên, với một vài sự cố hoạt động, độ sâu đường ống bộ xử lý có thể nhìn thấy và có thể khiến bộ xử lý bị đình trệ.

Một vấn đề thiết kế đầy thách thức là thực hiện lập lịch trình sợi dọc không trên đầu cho sự kết hợp năng động của các chương trình và loại chương trình sợi dọc khác nhau. Bộ lập lịch lệnh phải chọn một sợi dọc cứ sau bốn đồng hồ để đưa ra một lệnh cho mỗi đồng hồ trên mỗi luồng, tương đương với IPC là 1.0 cho mỗi lõi bộ xử lý. Bởi vì các sợi dọc là độc lập, các phần phụ thuộc duy nhất nằm trong số các lệnh tuần tự từ cùng một sợi dọc. Bộ lập lịch sử dụng bảng điểm phụ thuộc thanh ghi để xác định các warp có các luồng hoạt động sẵn sàng thực hiện một lệnh. Nó ưu tiên tất cả các đoạn cong sẵn sàng như vậy và chọn ưu tiên cao nhất cho vấn đề. Việc ưu tiên phải xem xét loại sợi dọc, loại lệnh và mong muốn công bằng cho tất cả các sợi dọc đang hoạt động.

4.4. Quản lý luồng và khối luồng

Bộ điều khiển đa xử lý và đơn vị lệnh quản lý các luồng và khối luồng. Bộ điều khiển chấp nhận các yêu cầu công việc và dữ liệu đầu vào và phân xử quyền truy cập vào các tài nguyên được chia sẻ. Đối với khối lượng công việc đồ họa có các đường dẫn đầu vào và đầu ra độc lập. Nó tích lũy và đóng gói từng loại công việc đầu vào này thành các đoạn SIMT của các luồng song song thực hiện cùng một chương trình luồng. Nó phân bổ một sợi dọc miễn phí, phân bổ các thanh ghi cho các luồng sợi dọc và bắt đầu thực hiện sợi dọc trong bộ đa xử lý. Khi tất cả các luồng của sợi dọc thoát ra, bộ điều khiển giải nén kết quả và giải phóng các thanh ghi và tài nguyên sợi dọc.

Bộ điều khiển tạo ra CTA thực hiện các khối luồng CUDA dưới dạng một hoặc nhiều sợi dọc của các luồng song song. CTA là một tập hợp các luồng đồng thời thực hiện cùng một chương trình luồng và có thể hợp tác để tính toán một

kết quả. CTA GPU thực hiện một khối luồng CUDA. Ngoài các luồng và thanh ghi, CTA yêu cầu phân bổ bộ nhớ dùng chung và các rào cản. Bộ điều khiển giám sát khi tất cả các luồng của CTA đã thoát và giải phóng tài nguyên được chia sẻ CTA và tài nguyên dọc của nó.

Chỉ thị luồng

Các bộ xử lý luồng SP thực thi các lệnh vô hướng cho các luồng riêng lẻ, không giống như các kiến trúc lệnh vectơ GPU trước đó, các kiến trúc này thực thi các lệnh vectơ bốn thành phần cho mỗi chương trình.

Để hỗ trợ nhiều GPU với các định dạng vi lệnh nhị phân khác nhau, trình biên dịch ngôn ngữ máy tính và đồ họa cấp cao tạo ra các lệnh cấp trình biên dịch trung gian sau đó được tối ưu hóa và dịch sang các lệnh vi lệnh GPU nhị phân. Bởi vì các lệnh cấp trình hợp dịch trung gian sử dụng các thanh ghi ảo, trình tối ưu hóa phân tích các phụ thuộc dữ liệu và cấp phát các thanh ghi thực. Trình tối ưu hóa loại bỏ mã chết, gấp các hướng dẫn lại với nhau khi khả thi và tối ưu hóa các điểm phân kỳ và hội tụ nhánh SIMT.

4.5. Kiến trúc bộ hướng dẫn (ISA)

Ví dụ: *Xét chỉ thị vô hướng PTX. Hình 4.3 liệt kê các chỉ thị luồng GPU PTX cơ bản. Định dạng chỉ thị sẽ là:*

```
Opcode.type d, a, b, c;
```

Trong đó:

`d`: toán hạng đích

`a, b, c`: toán hạng nguồn

`Opcode.type`: một trong các kiểu được liệt kê trong bảng

Type	.type Specifer
Untyped bits 8, 16, 32, and 64 bits	.b8, .b16, .b32, .b64
Unsigned integer 8, 16, 32, and 64 bits	.u8, .u16, .u32, .u64
Signed integer 8, 16, 32, and 64 bits	.s8, .s16, .s32, .s64
Floating-point 16, 32, and 64 bits	.f16, .f32, .f64

Bảng 4. 1: Các kiểu dữ liệu được sử dụng

Toán hạng nguồn là các giá trị vô hướng 32-bit hoặc 64-bit trong thanh ghi, một giá trị tức thời hoặc một hằng số. Đích là các thanh ghi, ngoại trừ lưu trữ vào bộ nhớ. Các lệnh PTX chỉ định hành vi của một luồng.

Basic PTX GPU Thread Instructions

Group	Instruction	Example	Meaning	Comments
Arithmetic	arithmetic *.type* = .s32, .u32, .f32, .s64, .u64, .f64			
	add.*type*	add.f32 d, a, b	d = a + b;	
	sub.*type*	sub.f32 d, a, b	d = a - b;	
	mul.*type*	mul.f32 d, a, b	d = a * b;	
	mad.*type*	mad.f32 d, a, b, c	d = a * b + c;	multiply-add
	div.*type*	div.f32 d, a, b	d = a / b;	multiple microinstructions
	rem.*type*	rem.u32 d, a, b	d = a % b;	integer remainder
	abs.*type*	abs.f32 d, a	d = |a|;	
	neg.*type*	neg.f32 d, a	d = 0 - a;	
	min.*type*	min.f32 d, a, b	d = (a < b)? a:b;	floating selects non-NaN
	max.*type*	max.f32 d, a, b	d = (a > b)? a:b;	floating selects non-NaN
	setp.*cmp.type*	setp.lt.f32 p, a, b	p = (a < b);	compare and set predicate
	numeric *.cmp* = eq, ne, lt, le, gt, ge; unordered *cmp* = equ, neu, ltu, leu, gtu, geu, num, nan			
	mov.*type*	mov.b32 d, a	d = a;	move
	selp.*type*	selp.f32 d, a, b, p	d = p? a: b;	select with predicate
	cvt.dtype.atype	cvt.f32.s32 d, a	d = convert(a);	convert atype to dtype
Special Function	special *.type* = .f32 (some .f64)			
	rcp.*type*	rcp.f32 d, a	d = 1/a;	reciprocal
	sqrt.*type*	sqrt.f32 d, a	d = sqrt(a);	square root
	rsqrt.*type*	rsqrt.f32 d, a	d = 1/sqrt(a);	reciprocal square root
	sin.*type*	sin.f32 d, a	d = sin(a);	sine
	cos.*type*	cos.f32 d, a	d = cos(a);	cosine
	lg2.*type*	lg2.f32 d, a	d = log(a)/log(2)	binary logarithm
	ex2.*type*	ex2.f32 d, a	d = 2 ** a;	binary exponential
Logical	logic. *type* = .pred, .b32, .b64			
	and.*type*	and.b32 d, a, b	d = a & b;	
	or.*type*	or.b32 d, a, b	d = a | b;	
	xor.*type*	xor.b32 d, a, b	d = a ^ b;	
	not.*type*	not.b32 d, a, b	d = ~a;	one's complement
	cnot.*type*	cnot.b32 d, a, b	d = (a==0)? 1:0;	C logical not
	shl.*type*	shl.b32 d, a, b	d = a << b;	shift left
	shr.*type*	shr.s32 d, a, b	d = a >> b;	shift right
Memory Access	memory *.space* = .global, .shared, .local, .const; *.type* = .b8, .u8, .s8, .b16, .b32, .b64			
	ld.*space.type*	ld.global.b32 d, [a+off]	d = *(a+off);	load from memory space
	st.*space.type*	st.shared.b32 [d+off], a	*(d+off) = a;	store to memory space
	tex.*nd.dtyp.btype*	tex.2d.v4.f32.f32 d, a, b	d = tex2d(a, b);	texture lookup
	atom.*spc.op.type*	atom.global.add.u32 d,[a], b atom.global.cas.b32 d,[a], b, c	atomic { d = *a; *a = op(*a, b); }	atomic read-modify-write operation
	atom *.op* = and, or, xor, add, min, max, exch, cas; *.spc* = .global; *.type* = .b32			
Control Flow	branch	@p bra target	if (p) goto target;	conditional branch
	call	call (ret), func, (params)	ret = func(params);	call function
	ret	ret	return;	return from function call
	bar.sync	bar.sync d	wait for threads	barrier synchronization
	exit	exit	exit;	terminate thread execution

Bảng 4. 2: Các câu lệnh cơ bản trong luồng của PTX GPU

4.6. Các lệnh truy cập bộ nhớ

Để hỗ trợ nhu cầu tính toán và ngôn ngữ C / C ++, Tesla PTX ISA thực hiện các lệnh tải / lưu trữ bộ nhớ. Nó sử dụng địa chỉ byte số nguyên với thanh ghi cộng với số học địa chỉ bù đắp để tạo điều kiện tối ưu hóa mã trình biên dịch thông thường.

Bộ nhớ cục bộ cho dữ liệu tạm thời có thể địa chỉ riêng theo từng luồng (được triển khai trong DRAM bên ngoài)

Bộ nhớ được chia sẻ để truy cập độ trễ thấp vào dữ liệu được chia sẻ bởi các luồng hợp tác trong cùng một khối CTA / luồng (được triển khai trong SRAM trên chip)

Bộ nhớ chung cho các tập dữ liệu lớn được chia sẻ bởi tất cả các luồng của một ứng dụng máy tính (được triển khai trong DRAM bên ngoài)

Để cải thiện băng thông bộ nhớ và giảm chi phí, các lệnh tải / lưu trữ cục bộ và toàn cục sẽ kết hợp các yêu cầu luồng song song riêng lẻ từ cùng một SIMT với nhau thành một yêu cầu khối bộ nhớ duy nhất khi các địa chỉ nằm trong cùng một khối và đáp ứng các tiêu chí liên kết. Các yêu cầu bộ nhớ kết hợp cung cấp hiệu suất tăng đáng kể so với các yêu cầu riêng biệt từ các luồng riêng lẻ. Số lượng luồng lớn của bộ đa xử lý, cùng với việc hỗ trợ nhiều yêu cầu tải vượt trội, giúp che phủ độ trễ tải để sử dụng cho bộ nhớ cục bộ và toàn cầu được triển khai trong DRAM bên ngoài.

4.7. Bộ xử lý phát trực tuyến (SP)

Lõi bộ xử lý luồng đa luồng (SP) là bộ xử lý lệnh luồng chính trong bộ đa xử lý. Tệp đăng ký (RF) của nó cung cấp 1024 thanh ghi 32-bit vô hướng cho tối đa 64 luồng. Nó thực thi tất cả các phép toán dấu phẩy động cơ bản. Lõi SP cũng thực hiện tất cả các lệnh số học, so sánh, chuyển đổi và PTX logic 32 bit và 64 bit.

4.8. Đơn vị chức năng đặc biệt (SFU)

Một số hướng dẫn luồng nhất định có thể thực thi trên SFU, đồng thời với các hướng dẫn luồng khác đang thực thi trên SP. SFU thực hiện các lệnh chức năng đặc biệt được mô tả ở Hình 4.3.

Mỗi đường ống SFU tạo ra một kết quả hàm đặc biệt dấu phẩy động 32 bit cho mỗi chu kỳ; hai SFU trên mỗi bộ đa xử lý thực hiện các lệnh chức năng đặc biệt với tốc độ một phần tư lệnh đơn giản của tám SP. Các SFU cũng thực hiện lệnh nhân mul.f32 đồng thời với tám SP, tăng tốc độ tính toán cao nhất lên đến 50% cho các luồng có hỗn hợp lệnh phù hợp.

4.9. So sánh với các bộ đa xử lý khác

So với các kiến trúc vectơ SIMD như x86 SSE, bộ đa xử lý SIMT có thể thực thi các luồng riêng lẻ một cách độc lập. Phần cứng SIMT tìm sự song song dữ liệu giữa các luồng độc lập, trong khi phần cứng SIMD yêu cầu phần mềm thể hiện tính song song dữ liệu một cách rõ ràng trong mỗi lệnh vectơ.

Bộ xử lý đa năng SIMT đặt ra ít chi phí vì nó là phần cứng đa luồng với đồng bộ hóa rào cản phần cứng

4.10. Kết luận

Bộ xử lý đa xử lý GPU mẫu dựa trên kiến trúc Tesla có tính đa luồng cao. Nó sử dụng một biến thể trên kiến trúc SIMD và đa luồng được gọi là SIMT (đa luồng lệnh đơn) để phát hiệu quả một lệnh.

PTX ISA là ISA vô hướng tải / lưu trữ dựa trên thanh ghi mô tả việc thực thi một luồng đơn lẻ. Vì các lệnh PTX được tối ưu hóa và dịch sang các vi lệnh nhị phân cho một GPU cụ thể, các lệnh phần cứng có thể phát triển nhanh chóng mà không làm gián đoạn các trình biên dịch và các công cụ phần mềm tạo ra các lệnh PTX.

5. HỆ THỐNG BỘ NHỚ SONG SONG

Phần 5 sẽ tiến hành trình bày các hệ thống bộ nhớ song song được sử dụng với bộ xử lý đồ họa GPU.

5.1. Đặc điểm

Ngoài bản thân GPU, hệ thống con bộ nhớ là yếu tố quan trọng nhất quyết định hiệu suất của hệ thống đồ họa. Khối lượng công việc đồ họa đòi hỏi tốc độ truyền tải đến và từ bộ nhớ rất cao. Các thao tác ghi và pha trộn pixel (đọc-sửa-ghi), đọc và ghi bộ đệm độ sâu, đọc bản đồ kết cấu, cũng như đọc lệnh và đỉnh đối tượng và dữ liệu thuộc tính, bao gồm phần lớn lưu lượng bộ nhớ.

Hệ thống bộ nhớ GPU có các đặc điểm sau:

- Có một số lượng lớn các chân để truyền tải dữ liệu giữa GPU và các thiết bị bộ nhớ của nó và bản thân mảng bộ nhớ bao gồm nhiều chip DRAM
- Các kỹ thuật báo hiệu tích cực được sử dụng để tối đa hóa tốc độ dữ liệu (bps) trên mỗi chân
- GPU tìm cách sử dụng mọi chu kỳ có sẵn để truyền dữ liệu đến hoặc từ mảng bộ nhớ.
- Các kỹ thuật nén được sử dụng, cả kỹ thuật mất mát.
- Bộ nhớ đệm và cấu trúc liên kết công việc được sử dụng để giảm lượng lưu lượng truy cập off-chip cần thiết.

5.2. Cân nhắc về DRAM

Một giải pháp là để bộ điều khiển bộ nhớ của GPU duy trì các đống lưu lượng riêng biệt bị ràng buộc cho các ngân hàng DRAM khác nhau và đợi cho đến khi đủ lưu lượng cho một hàng DRAM cụ thể đang chờ xử lý trước khi kích hoạt hàng đó và chuyển tất cả lưu lượng cùng một lúc. Thiết kế phải cẩn thận để không có yêu cầu cụ thể nào chờ quá lâu, nếu không một số đơn vị xử lý có thể chết đói khi chờ dữ liệu và cuối cùng khiến các bộ xử lý lân cận trở nên nhàn rỗi.

Để đạt được cân bằng tải tốt nhất và do đó đạt đến hiệu suất lý thuyết của n phân vùng, các hệ thống con bộ nhớ GPU được sắp xếp thành nhiều phân vùng

bộ nhớ, mỗi phân vùng bao gồm một bộ điều khiển bộ nhớ hoàn toàn độc lập và một hoặc hai thiết bị DRAM thuộc sở hữu hoàn toàn và độc quyền của phân vùng đó.

5.3. Caches

Khối lượng công việc GPU thường có các bộ làm việc rất lớn — theo thứ tự hàng trăm megabyte để tạo ra một khung hình đồ họa duy nhất. Không giống như với CPU, việc xây dựng bộ nhớ đệm trên chip đủ lớn để chứa bất cứ thứ gì gần với toàn bộ hoạt động của một ứng dụng đồ họa là không thực tế. Trong khi CPU có thể giả định tỷ lệ truy cập bộ nhớ cache rất cao (99,9% trở lên), GPU có tỷ lệ truy cập gần 90% và do đó phải đối phó với nhiều lần bỏ lỡ trong chuyến bay. Mặc dù CPU có thể được thiết kế hợp lý để tạm dừng trong khi chờ một lần bỏ lỡ bộ nhớ cache hiếm gặp, GPU cần phải xử lý các lần bỏ lỡ và truy cập xen kẽ. Chúng tôi gọi đây là kiến trúc bộ nhớ đệm trực tuyến.

Bộ nhớ đệm GPU phải cung cấp băng thông rất cao cho các máy khách của họ. Hãy xem xét trường hợp của một bộ đệm kết cấu. Một đơn vị kết cấu điển hình có thể đánh giá hai nội suy song tuyến cho mỗi bốn pixel trên mỗi chu kỳ đồng hồ và một GPU có thể có nhiều đơn vị kết cấu như vậy, tất cả đều hoạt động độc lập. Mỗi phép nội suy song tuyến yêu cầu bốn texel riêng biệt và mỗi texel có thể là một giá trị 64 bit. Bốn thành phần 16-bit là điển hình. Như vậy, tổng băng thông là $2 \times 4 \times 4 \times 64 = 2048$ bit trên mỗi đồng hồ. Mỗi texel 64-bit riêng biệt được đánh địa chỉ độc lập, vì vậy bộ nhớ đệm cần phải xử lý 32 địa chỉ duy nhất trên mỗi đồng hồ. Điều này đương nhiên ủng hộ sự sắp xếp đa ngân hàng và/hoặc đa cổng của mảng SRAM.

5.4. MMU

Các GPU hiện đại có khả năng dịch địa chỉ ảo sang địa chỉ vật lý. Trên GeForce 8800, tất cả các đơn vị xử lý tạo ra địa chỉ bộ nhớ trong không gian địa chỉ ảo 40 bit. Đối với tính toán, tải và lưu trữ các lệnh luồng sử dụng địa chỉ byte 32 bit, địa chỉ này được mở rộng thành địa chỉ ảo 40 bit bằng cách thêm 1328 độ lệch 40 bit. Một đơn vị quản lý bộ nhớ thực hiện dịch địa chỉ ảo sang địa chỉ vật lý; phần cứng đọc các bảng trang từ bộ nhớ cục bộ để phản hồi các lần bỏ sót thay mặt cho một hệ thống phân cấp của bộ đệm giao diện dịch được trải ra giữa các bộ xử lý và công cụ kết xuất. Ngoài các bit trang vật lý, các mục trong bảng trang GPU

chỉ định thuật toán nén cho mỗi trang. Kích thước trang nằm trong khoảng từ 4 đến 128 kilobyte.

5.5. Không gian bộ nhớ

Bộ nhớ chung được lưu trữ trong DRAM bên ngoài; nó không phải là cục bộ cho bất kỳ bộ xử lý đa xử lý trực tuyến vật lý nào (SM) vì nó dành cho giao tiếp giữa các CTA khác nhau (khối luồng) trong các lưới khác nhau. Trên thực tế, nhiều CTA tham chiếu đến một vị trí trong bộ nhớ chung có thể không thực thi trong GPU cùng một lúc; theo thiết kế, trong CUDA, một lập trình viên không biết thứ tự tương đối mà các CTA được thực thi. Vì không gian địa chỉ được phân bổ đồng đều giữa tất cả các phân vùng bộ nhớ, nên phải có một đường dẫn đọc / ghi từ bất kỳ bộ xử lý đa luồng nào đến bất kỳ phân vùng DRAM nào.

Quyền truy cập vào bộ nhớ chung của các luồng khác nhau (và các bộ xử lý khác nhau) không được đảm bảo là có tính nhất quán tuần tự. Các chương trình chủ đề xem mô hình sắp xếp bộ nhớ thoải mái Trong một luồng, thứ tự của bộ nhớ đọc và ghi vào cùng một địa chỉ được giữ nguyên, nhưng thứ tự của các truy cập đến các địa chỉ khác nhau có thể không được giữ nguyên. Việc đọc và ghi bộ nhớ được yêu cầu bởi các luồng khác nhau là không có thứ tự. Trong một CTA, thanh hướng dẫn đồng bộ hóa hàng rào.sync có thể được sử dụng để có được thứ tự bộ nhớ nghiêm ngặt giữa các luồng của CTA.

Bộ nhớ chung

Bộ nhớ chung được lưu trữ trong DRAM bên ngoài; nó không phải là cục bộ cho bất kỳ bộ xử lý đa xử lý trực tuyến vật lý nào (SM) vì nó dành cho giao tiếp giữa các CTA khác nhau (khối luồng) trong các lưới khác nhau.

Quyền truy cập vào bộ nhớ chung của các luồng khác nhau (và các bộ xử lý khác nhau) không được đảm bảo có tính nhất quán tuần tự. Các chương trình chủ đề xem mô hình sắp xếp bộ nhớ thoải mái. Trong một chuỗi, thứ tự bộ nhớ đọc và ghi vào cùng một địa chỉ được giữ nguyên, nhưng thứ tự truy cập đến các địa chỉ khác nhau có thể không được giữ nguyên. Việc đọc và ghi bộ nhớ được yêu cầu bởi các luồng khác nhau là không có thứ tự.

Bộ nhớ được chia sẻ

Bộ nhớ được chia sẻ trên mỗi CTA chỉ hiển thị đối với các chuỗi thuộc về CTA đó và bộ nhớ được chia sẻ chỉ chiếm dung lượng lưu trữ từ khi CTA được tạo đến khi nó kết thúc. Do đó, bộ nhớ dùng chung có thể nằm trên chip. Cách tiếp cận này có nhiều lợi ích. Đầu tiên, traffc bộ nhớ chia sẻ không cần phải cạnh tranh với băng thông offchip hạn chế cần thiết cho các tham chiếu bộ nhớ toàn cầu. Thứ hai, thực tế là xây dựng cấu trúc bộ nhớ băng thông rất cao trên chip để hỗ trợ nhu cầu đọc / ghi của mỗi bộ xử lý đa luồng. Trên thực tế, bộ nhớ dùng chung được kết hợp chặt chẽ với bộ xử lý đa xử lý trực tuyến.

Mỗi bộ xử lý đa luồng chứa tám bộ xử lý luồng vật lý. Trong một chu kỳ xung nhịp bộ nhớ chia sẻ, mỗi bộ xử lý luồng có thể xử lý hai luồng lệnh có giá trị của hai luồng, do đó, yêu cầu bộ nhớ dùng chung có giá trị 16 luồng phải được xử lý trong mỗi xung nhịp. Bởi vì mỗi luồng có thể tạo địa chỉ riêng và các địa chỉ này thường là duy nhất, bộ nhớ dùng chung được xây dựng bằng 16 ngân hàng SRAM có thể định địa chỉ độc lập. Đối với các kiểu truy cập thông thường, 16 ngân hàng là đủ để duy trì thông lượng, nhưng có thể xảy ra các trường hợp bệnh lý; ví dụ: tất cả 16 luồng có thể tình cờ truy cập vào một địa chỉ khác trên một ngân hàng SRAM. Phải có thể định tuyến một yêu cầu từ bất kỳ làn luồng nào đến bất kỳ ngân hàng nào của SRAM, vì vậy cần có mạng kết nối 16 x 16.

Bộ nhớ cục bộ

Bộ nhớ cục bộ trên mỗi luồng là bộ nhớ riêng chỉ hiển thị cho một luồng duy nhất. Bộ nhớ cục bộ lớn hơn về mặt kiến trúc so với tệp đăng ký của chuỗi và một chương trình có thể tính toán các địa chỉ vào bộ nhớ cục bộ. Để hỗ trợ phân bổ lớn bộ nhớ cục bộ (gọi lại tổng phân bổ là phân bổ mỗi luồng nhân với số luồng hoạt động), bộ nhớ cục bộ được cấp phát trong DRAM bên ngoài.

Mặc dù bộ nhớ cục bộ toàn cầu và mỗi luồng nằm ngoài chip, nhưng chúng rất phù hợp để được lưu vào bộ nhớ đệm trên chip.

Bộ nhớ không đổi

Bộ nhớ không đổi là bộ nhớ chỉ đọc đối với một chương trình chạy trên SM (nó có thể được ghi thông qua các lệnh tới GPU). Nó được lưu trữ trong DRAM bên ngoài và được lưu trong bộ nhớ cache trong SM. Bộ nhớ đệm không đổi được thiết kế để phát các giá trị vô hướng cho các luồng trong mỗi sợi dọc.

Bộ nhớ kết cấu

Bộ nhớ kết cấu chứa các mảng dữ liệu lớn chỉ đọc. Các kết cấu cho máy tính có các thuộc tính và khả năng giống như các kết cấu được sử dụng với đồ họa 3D.

Tìm nạp kết cấu được lưu vào bộ nhớ đệm trong một hệ thống phân cấp bộ đệm trực tuyến được thiết kế để tối ưu hóa thông lượng của các lần tìm nạp kết cấu từ hàng nghìn luồng đồng thời. Một số chương trình sử dụng kết cấu tìm nạp như một cách để lưu bộ nhớ chung vào bộ nhớ đệm.

5.6. Surfaces

Surfaces là một thuật ngữ chung để chỉ mảng một chiều, hai chiều hoặc ba chiều của các giá trị pixel và định dạng được liên kết. Một loạt các định dạng được xác định; ví dụ, một pixel có thể được định nghĩa là bốn thành phần số nguyên RGBA 8 bit hoặc bốn thành phần dấu phẩy động 16 bit. Một nhân chương trình không cần biết kiểu bề mặt. Một lệnh tex tính lại các giá trị kết quả của nó dưới dạng dấu phẩy động, tùy thuộc vào định dạng bề mặt.

5.7. Truy cập tải/lưu trữ

Các lệnh tải / lưu trữ với địa chỉ byte số nguyên cho phép viết và biên dịch các chương trình bằng các ngôn ngữ thông thường như C và C ++. Các chương trình CUDA sử dụng hướng dẫn tải / lưu trữ để truy cập bộ nhớ.

Để cải thiện băng thông bộ nhớ và giảm chi phí, các lệnh tải / lưu trữ cục bộ và toàn cục kết hợp các yêu cầu luồng song song riêng lẻ từ cùng một sợi dọc với nhau thành một yêu cầu khối bộ nhớ duy nhất khi các địa chỉ nằm trong cùng một khối và đáp ứng các tiêu chí liên kết.

5.8. ROP

Mỗi đơn vị ROP được ghép nối với một phân vùng bộ nhớ cụ thể. Các phân vùng ROP được cấp từ các SM thông qua một mạng kết nối. Mỗi ROP chịu trách nhiệm kiểm tra và cập nhật độ sâu và stencil, cũng như pha trộn màu sắc. ROP và bộ điều khiển bộ nhớ hợp tác để thực hiện nén độ sâu và màu không mất dữ liệu (lên đến 8: 1) để giảm nhu cầu băng thông bên ngoài. Các đơn vị ROP cũng thực hiện các hoạt động nguyên tử trên bộ nhớ.

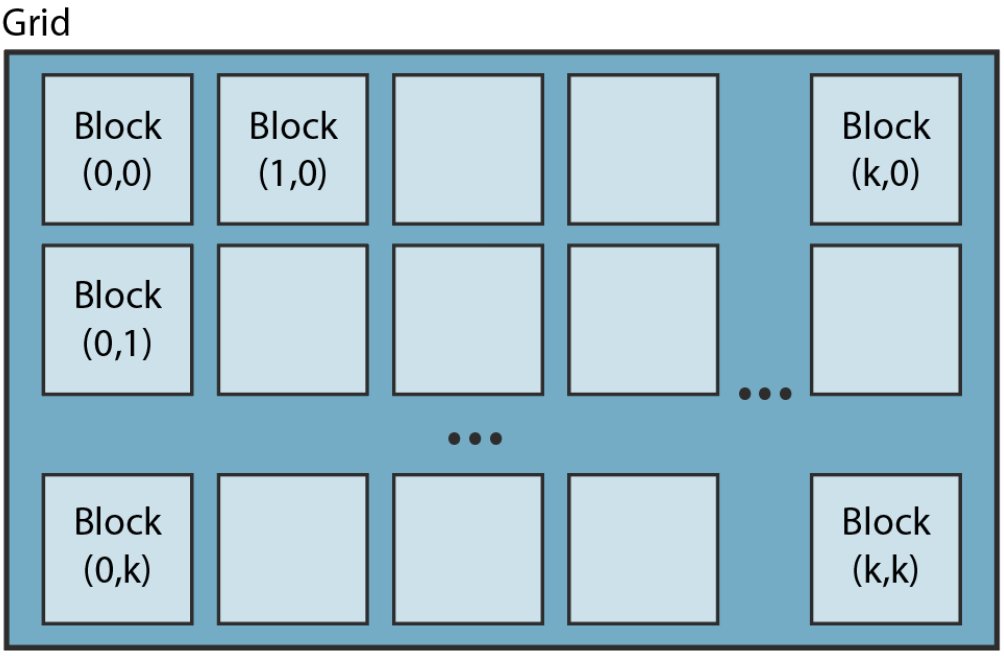
6. TRIỂN KHAI PHÉP NHÂN MA TRẬN & BỘ LỌC TRUNG VỊ

6.1. Cơ sở lý thuyết

6.1.1. Phép nhân ma trận hai chiều, sử dụng GPU

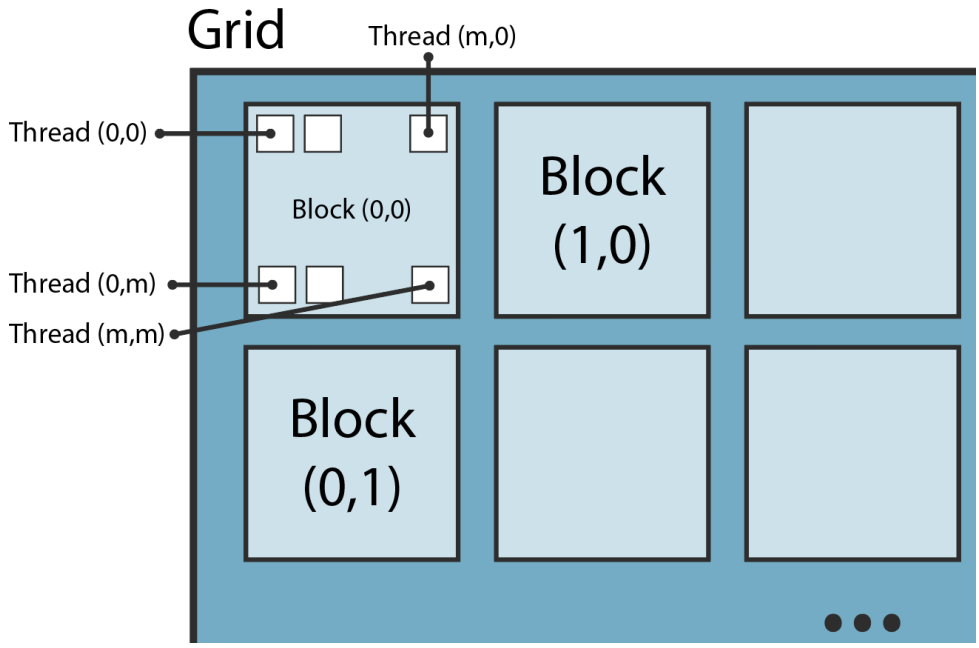
Giả sử chúng ta có 2 ma trận A, B. Ma trận A có kích thước $x \times m$, B có kích thước $m \times y$. Kế quả khi AxB sẽ được ma trận mới có kích thước $x \times y$ chúng ta gọi là C.

Mỗi luồng sẽ tính ra được một phần tử trong ma trận C (thread(x,y)) sẽ tính ra được phần tử C(x,y) bằng cách sử dụng hang của ma trận A và cột của ma trận B. Số chiều của khối (block) là giới hạn chúng ta sẽ xử dụng mảng 2 chiều cho lưới (grid) với k x k blocks như hình 6. 1.



Hình 6. 1: Lưới (grid) tính toán với kích cỡ k x k khối (block)

Mảng hai chiều cũng được sử dụng để tính toán cho luồng của mỗi khối với kích thước mỗi khối là m x m.



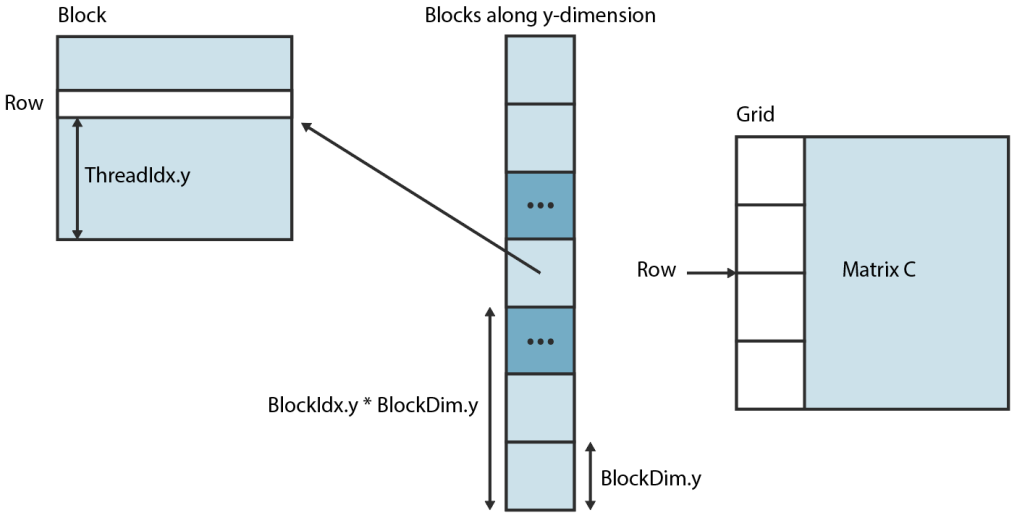
Hình 6. 2: Khối (block) tính toán với kích cỡ m x m luồng (thread)

Phương pháp tính toán:

$$\begin{cases} k = \frac{n}{m}, nếu\ n\ chia\ hết\ cho\ m \\ k = \frac{n}{m} + 1, nếu\ n\ không\ chia\ hết\ cho\ m \end{cases}$$

Tổng hợp các kết quả, ta thu được ma trận C với kích thước n x n.

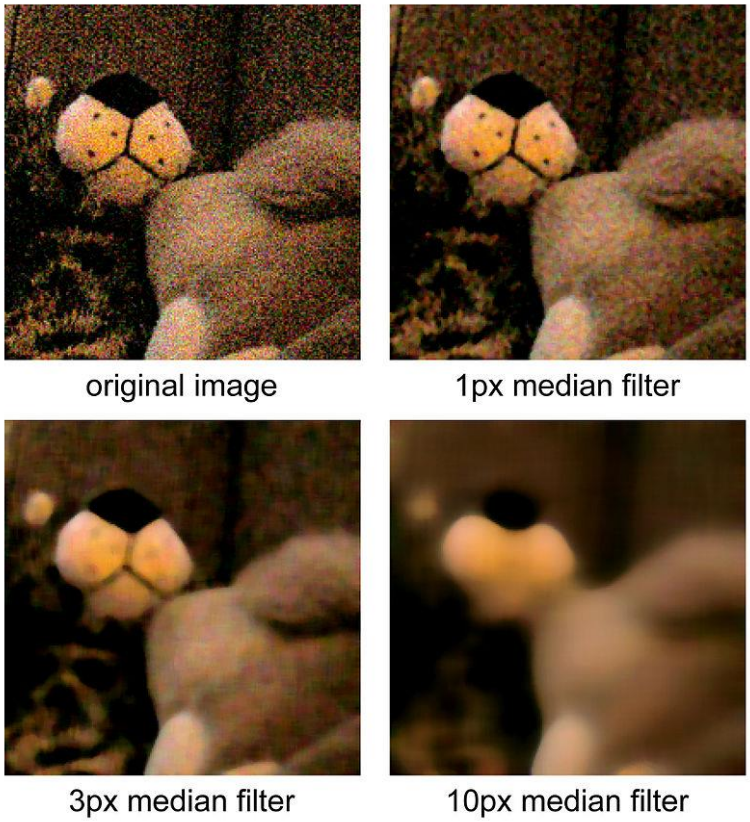
Số lượng luồng tối đa của một khối là 2048 luồng. Số lượng khối tối đa trong một lưới tính toán là 1024 khối.



Hình 6. 3: Quan hệ tính toán giữa luồng, khối và lưới.

6.1.2. Bộ lọc trung vị

Bộ lọc trung vị (Median filter) là một trong những phương pháp xử lý ảnh phi tuyến tính, thường được sử dụng để loại bỏ các tín hiệu nhiễu/tạp âm ra khỏi dữ liệu hình ảnh hoặc tín hiệu. Đây có thể gọi là một bước tiền xử lý quan trọng trước khi đưa dữ liệu vào các phần xử lý tại phía sau, ví dụ như trong ứng dụng tìm vùng bao của vật thể ở trong hình ảnh. Bộ lọc trung vị được sử dụng rất phổ biến trong lĩnh vực xử lý ảnh số, bởi trong các điều kiện nhất định, bộ lọc này có thể loại bỏ các nhiễu khỏi hình ảnh trong khi vẫn giữ được độ sắc nét của của viền các vật thể nằm trong hình ảnh.

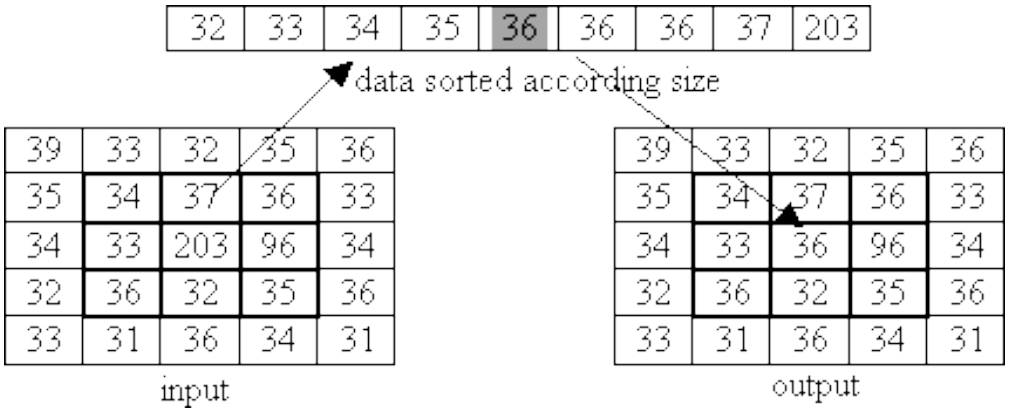


Hình 6. 4: Ví dụ về hình ảnh có nhiễu và được xử lý qua bộ lọc trung vị với kích thước lọc khác nhau [7]

Phương pháp xử lý:

Ý tưởng lớn trong thuật toán xử lý của bộ lọc trung vị là lượt qua các tín hiệu (hay cụ thể hơn là các điểm ảnh trong hình ảnh), tính toán giá trị mức sáng median của khu vực đã quét, và kiểm tra lại các giá trị nằm trong vùng đã quét. Nếu giá trị của một điểm ảnh nhất định vượt quá nhiều lần giá trị

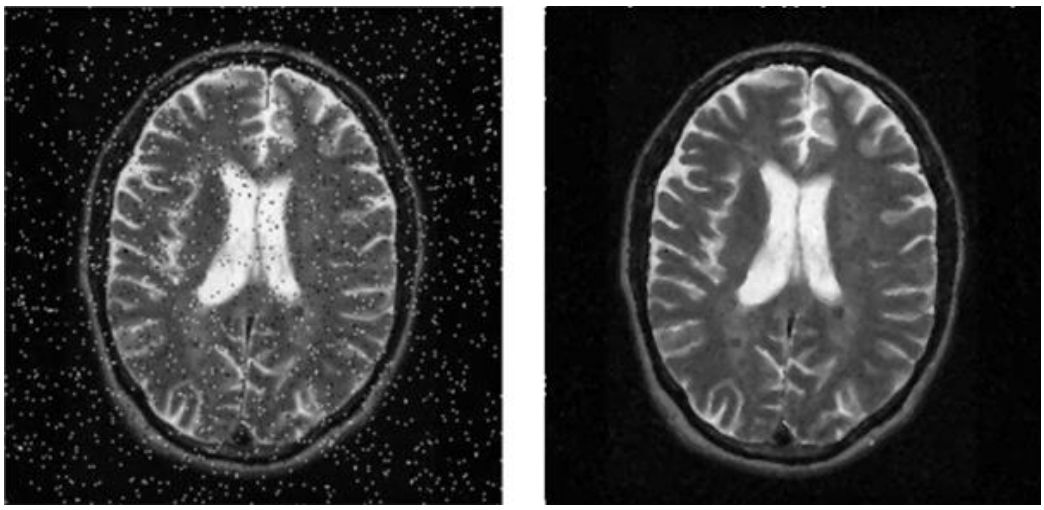
median của vùng được quét, giá trị tại vị trí này sẽ được đặt lại bằng giá trị median của vùng.



Hình 6. 5: Ví dụ về cách xử lý của bộ lọc trung vị, kích thước 3 x 3

Trong ví dụ tại hình 6. 5, một vùng điểm ảnh (hay ma trận) với kích thước 3 x 3 được kiểm tra. Tại đây, median của vùng được xác định là giá trị 36. Tiếp đến, tại tọa độ (1,1) của vùng được quét, giá trị đầu vào đọc được là 203, lớn hơn rất nhiều so với giá trị median bằng 36 của vùng. Vì vậy, giá trị 203 được thay đổi lại bằng giá trị 36.

Với thuật toán xử lý này, bộ lọc trung vị có khả năng lọc nhiễu khá tốt, đặc biệt là với loại nhiễu "muối tiêu" (salt-and-pepper) – một khái niệm đặc trưng cho các điểm nhiễu trắng (salt) và các điểm nhiễu đen (pepper) trên hình ảnh. Các điểm nhiễu này thường có giá trị nằm ra khỏi median của vùng lân cận chúng.



Hình 6. 6: Hình ảnh gốc với nhiễu (trái) và
hình ảnh sau khi qua bộ lọc trung vị (phải) [6]

6.2. Lập trình phép nhân ma trận & bộ lọc trung vị chạy trên GPU

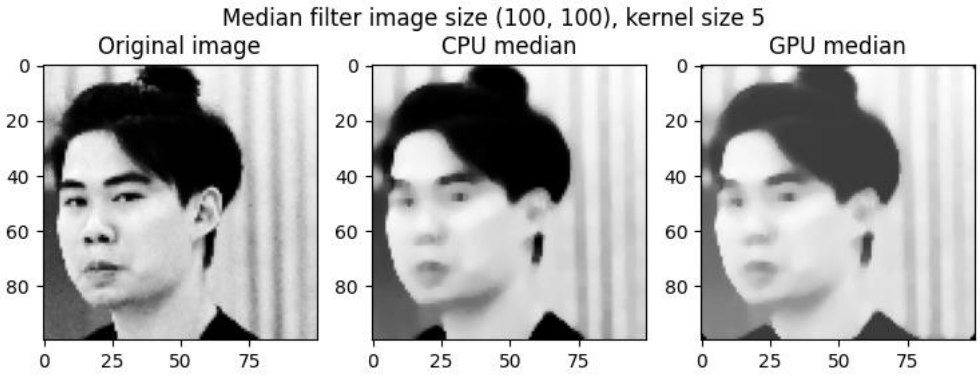
Dựa vào các lý thuyết đã đưa ra ở trên, em tiến hành vào việc lập trình cho phép nhân ma trận & tạo bộ lọc trung vị cho hình ảnh. Sau đó, em tiến hành kiểm nghiệm hiệu quả hoạt động của lập trình dựa trên hình ảnh của cá nhân, với các kích thước khác nhau và kích thước của bộ lọc khác nhau, từ đó thu được kết quả thể hiện như ở các hình dưới.

Thông số lập trình:

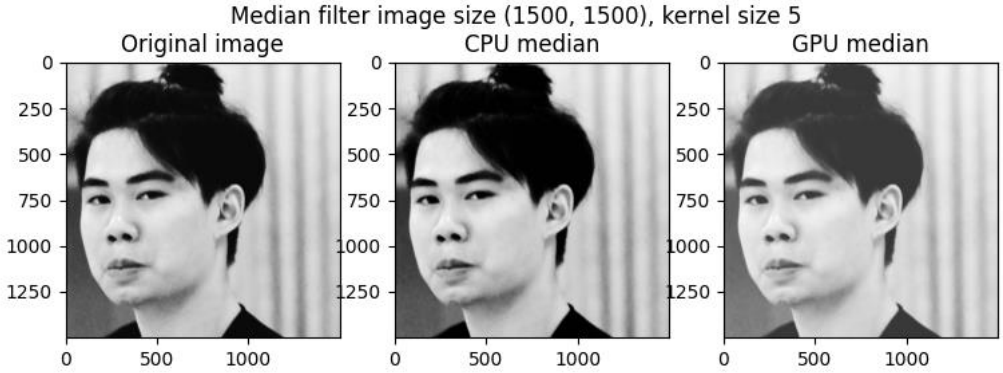
- Ngôn ngữ lập trình: Python
- Phiên bản Python: Python 3.8, sử dụng các thư viện lập trình
    - OpenCV-Python: thư viện xử lý ảnh
    - numba: thư viện tính toán, sử dụng để lập trình CUDA
    - numpy: thư viện tính toán, sử dụng để lập trình tính toán ma trận
    - matplotlib: thư viện đồ họa, sử dụng để hiển thị kết quả/vẽ đồ thị
- Cấu hình máy tính sử dụng:
    - CPU: Intel® Xeon® Bronze 3104
    - RAM: 31.7GB DDR4, 2133 Mhz
    - GPU: Nvidia® Quadro RTX 4000

Kết quả thử nghiệm:

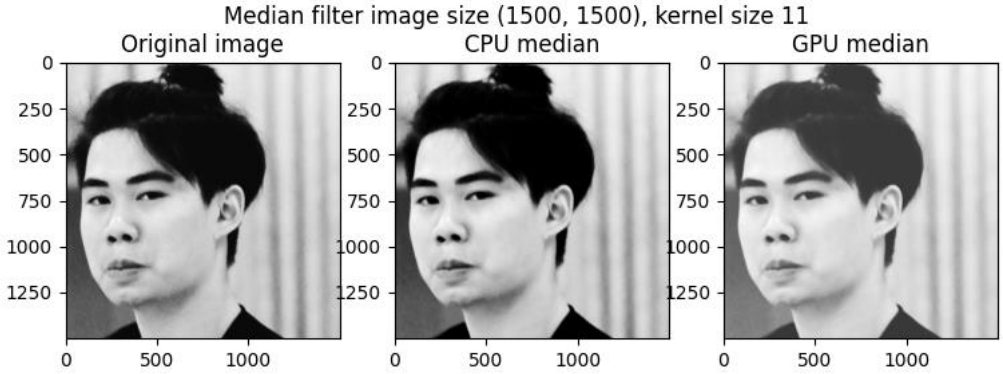
- Bộ lọc trung vị kích thước 5 x 5, kích thước ảnh 100 x 100:



Hình 6. 7: Hình ảnh cỡ 100 x 100, qua bộ lọc trung vị kích cỡ 5 x 5



Hình 6. 8: Hình ảnh cỡ 1500 x 1500 (kích cỡ ảnh gốc), qua bộ lọc trung vị 5 x 5



Hình 6. 9: Hình ảnh cỡ 1500 x 1500 (kích cỡ ảnh gốc), qua bộ lọc trung vị 11 x 11

Bảng tổng kết hiệu năng hoạt động theo thời gian:

- Bộ lọc trung vị kích thước 5 x 5:

Kích thước ảnh/Thời gian trung bình	**CPU**	**GPU**
100 x 100	0.0002487301826477051	0.0007467269897460938
250 x 250	0.0009942054748535156	0.00024884939193725586
500 x 500	0.0012440085411071777	0.00024890899658203125
750 x 750	0.004977524280548096	0.00024890899658203125
1000 x 1000	0.0054743289947509766	0.00024896860122680664
1500 x 1500	0.014685332775115967	0.00024896860122680664

Bảng 6. 1: Kết quả hiệu năng thời gian hoạt động của bộ lọc trung vị, kích cỡ 5 x 5

- Bộ lọc trung vị kích thước 11 x 11:

Kích thước ảnh/Thời gian trung bình	CPU	GPU
100 x 100	0.0015038251876831055	0.0004978179931640625
250 x 250	0.0070798397064208984	0.000663704342312283
500 x 500	0.026328855090671115	0.00044266382853190106
750 x 750	0.06294427977667914	0.0007746219635009766
1000 x 1000	0.09579857190450032	0.00044282277425130207
1500 x 1500	0.16350579261779785	0.00022141138712565103

Bảng 6. 2: Kết quả hiệu năng thời gian hoạt động của bộ lọc trung vị, kích cỡ 11 x 11

Nhận xét:

- Về kết quả hình ảnh:
  - Việc áp dụng bộ lọc trung vị cho hình ảnh lớn hơn (hình ảnh với kích thước các chiều lớn hơn) sẽ phù hợp hơn, do hình ảnh với kích thước nhỏ vốn đã không có nhiều dữ liệu để hoạt động. Việc áp dụng thêm bộ lọc trung vị sẽ khiến ảnh bị mờ (như ví dụ tại hình 6. 7, kích thước ảnh 100 x 100)
  - Việc áp dụng bộ lọc trung vị với kích thước càng nhỏ sẽ giữ lại được các chi tiết của ảnh tốt hơn, tuy nhiên, điều này đồng nghĩa với việc có thể sẽ bỏ sót nhiễu hoặc xử lý sai.
    Việc áp dụng kích thước bộ lọc lớn hơn cho khả năng sinh ra ảnh với nhiễu thấp hơn, tuy nhiên nếu áp dụng cho ảnh có kích thước nhỏ thì khả năng ảnh bị mờ/nhòe cũng lớn hơn.
  - Với hình ảnh có kích thước lớn (như ví dụ tại hình 6. 8, hình 6. 9), độ chênh lệch của kích thước độ lọc là rất nhỏ đối với kích thước ảnh, do đó, ảnh hưởng bị mờ do kích thước bộ lọc lớn là không đáng kể
- Về kết quả hiệu năng hoạt động theo thời gian:

- Tại kích thước ảnh và kích thước bộ lọc nhỏ nhất, việc xử lý trên CPU vẫn chiếm ưu thế.
- Khi tăng dần kích thước hình ảnh phải xử lý và kích thước bộ lọc, việc xử lý song song trên GPU cho thấy khả năng vượt trội hơn hẳn. Thời gian xử lý hình ảnh trên GPU luôn giữ dưới mức 1ms trong khi thời gian xử lý trên CPU tăng một cách đáng kể, đặc biệt với kích thước ảnh lớn nhất đã lên tới hàng chục ms.

→ Các kết quả này rất phù hợp với những nhận định đã đưa ra tại phía trên, phần 1.8: *"…GPU là một bộ xử lý phù hợp cho các bài toán yêu cầu xử lý dữ liệu song song, đặc biệt trong các bài toán lớn, và ít hơn trong các bài toán nhỏ, thường xuyên hơn."*

Lưu ý: *Các kết quả về hiệu năng hoạt động trong bài làm chỉ mang tính tương đối, sử dụng để làm một thước đo so sánh hiệu năng hoạt động giữa GPU và CPU. Các kết quả trên phụ thuộc rất lớn vào cấu hình các phần cứng được sử dụng, phụ thuộc vào giới hạn của phần cứng cũng như cách thức tối ưu việc sử dụng phần cứng trong lập trình. Để có được một kết quả chính xác nhất cho hiệu năng hoạt động trên từng thiết bị, việc chạy lại mã nguồn (có thể tìm thấy tại [2]) là cần thiết!*

7. KẾT LUẬN

GPU quả thực là một phát minh sáng giá trong ngành công nghiệp bán dẫn từ khi xuất hiện cho đến tận ngày nay. Tuy khởi điểm với mục đích sử dụng để hỗ trợ cho CPU xử lý các tác vụ liên quan tới đồ họa, ngày nay, những ứng dụng của GPU đã vượt ra ngoài giới hạn trong việc xử lý đồ họa, trở thành một công cụ rất hữu hiệu trong tính toán dữ liệu lớn, hay thực hiện các ứng dụng của học máy, học sâu, v.v..

Với khả năng xử lý song song, xử lý các dữ liệu lớn, và nền tảng lập trình CUDA, chắc chắn, các bộ xử lý đồ họa GPU sẽ còn được bổ sung vào thêm càng nhiều tính năng nữa, áp vào nhiều ứng dụng hơn nữa trong ngành điện toán. Đặc biệt đi cùng với sự phát triển không ngừng nghỉ của ngành công nghiệp bán dẫn, hướng tới các mật độ tích hợp nhỏ hơn, tiêu tốn ít năng lượng hơn, có lẽ chúng ta sẽ còn thấy nhiều khía cạnh mới về GPU được nảy sinh trong tương lai.

Một lần nữa, em xin chân thành cảm ơn cô Tạ Thị Kim Huệ, người đã hướng dẫn và cung cấp cho em những kiến thức để em có thể hoàn thiện phần tìm hiểu của em về đề tài này.

TÀI LIỆU THAM KHẢO

[1] Appendix B: Graphics and Computing GPUs, Computer Organization and Design RISC-V Edition – The hardware software interface, 2nd edition, David A.Patterson & John L.Hennessy

[2] Đường dẫn tới mã nguồn của đề tài, GitHub:

https://github.com/ducpham2476/median_filter_gpu_cuda

[3] What is a GPU – Intel:
https://www.intel.com/content/www/us/en/products/docs/processors/what-is-a-gpu.html

[4]: CPU vs GPU – Omnisci.com:

https://www.omnisci.com/technical-glossary/cpu-vs-gpu#:~:text=The%20main%20difference%20between%20CPU,resolution%20images%20and%20video%20concurrently.

[5] Video list: A Brief History of Graphics – Ahoy, YouTube:

https://www.youtube.com/watch?v=QyjyWUrHsFc

[6] Median Filtering with Python and OpenCV:

https://medium.com/@florestony5454/median-filtering-with-python-and-opencv-2bce390be0d1

[7] Hình ảnh minh họa về bộ lọc trung vị, Wikipedia:

https://en.wikipedia.org/wiki/Median_filter#/media/File:Median_filter_example.jpg