# Giới thiệu CUDA C/C++ (phần 2)

Trần Trung Kiên
ttkien@fit.hcmus.edu.vn

Cập nhật lần cuối: 29/09/2021

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

fit@hcmus

# Ôn lại buổi trước

**Cấu trúc của một chương trình CUDA đơn giản**

- Những phần tính toán tuần tự chạy ở host (CPU), những phần tính toán song song (mức độ lớn) chạy ở device (GPU)
- Từ host, để nhờ device tính toán song song:
  - Host cấp phát các vùng nhớ ở device bằng hàm **`cudaMalloc`**
  - Host chép các dữ liệu cần thiết sang các vùng nhớ ở device bằng hàm **`cudaMemcpy`**
  - Host gọi hàm kernel
  - Host chép kết quả từ device về bằng hàm **`cudaMemcpy`**
  - Host giải phóng các vùng nhớ ở device bằng hàm **`cudaFree`**

# Ôn lạ

**Cấu trúc của một chươn**

- Những phần tính toá
  những phần tính toán
  device (GPU)
- Từ host, để nhờ device
  - Host cấp phát các
    **`cudaMalloc`**
  - Host chép các dữ liệ
    device bằng hàm **`cuda`**
  - Host gọi hàm kernel
  - Host chép kết quả từ
  - Host giải phóng cá
    **`cudaFree`**

- Hàm kernel được thực thi song song ở device bởi rất nhiều thread (tiểu trình)
- Các thread này được tổ chức như sau:
  - grid (lưới tiểu trình) gồm các block (khối tiểu trình) có cùng kích thước
  - block gồm các thread

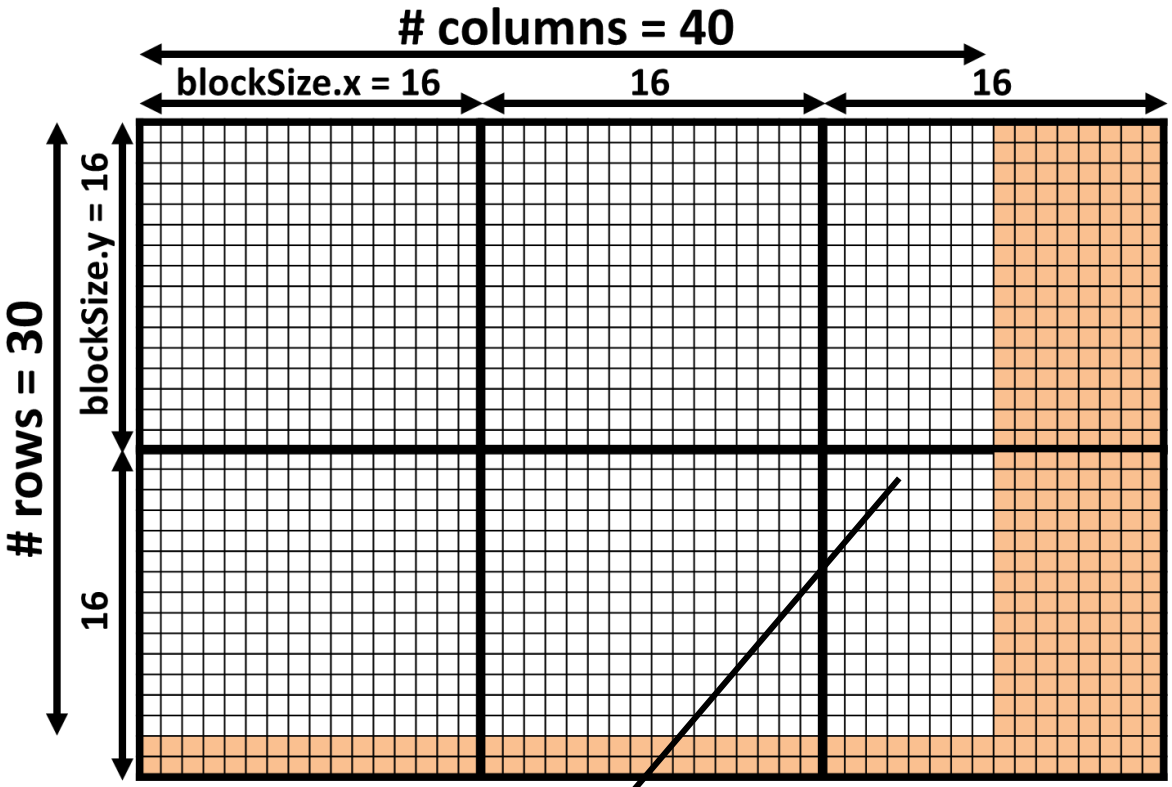
  Khi host gọi hàm kernel, host cần cho biết là grid gồm bao nhiêu block và mỗi block gồm bao nhiêu thread
- Ở phía device, trong hàm kernel: mỗi thread có thể sử dụng các biến hệ thống **`blockIdx`**, **`threadIdx`**, **`blockDim`**, **`gridDim`** để xác định phần dữ liệu mà mình sẽ phụ trách tính toán

# Buổi này

- Làm với dữ liệu 2D: cộng 2 ma trận
- Truy vấn thông tin device

# Cộng 2 ma trận

- **Cài đặt tuần tự:** dễ ☺
- **Cài đặt song song:** không khó ☺
  - Để mỗi thread phụ trách tính một phần tử trong ma trận kết quả
  - Với dữ liệu 2D, một cách tự nhiên là ta sẽ tổ chức các thread dưới dạng block 2D và grid 2D




blockIdx.x = 2 threadIdx.x = 3
blockIdx.y = 1 threadIdx.y = 1

r = blockIdx.y*blockDim.y + threadIdx.y
= 1*16 + 1
= 17
c = blockIdx.x*blockDim.x + threadIdx.x
= 2*16 + 3
= 35

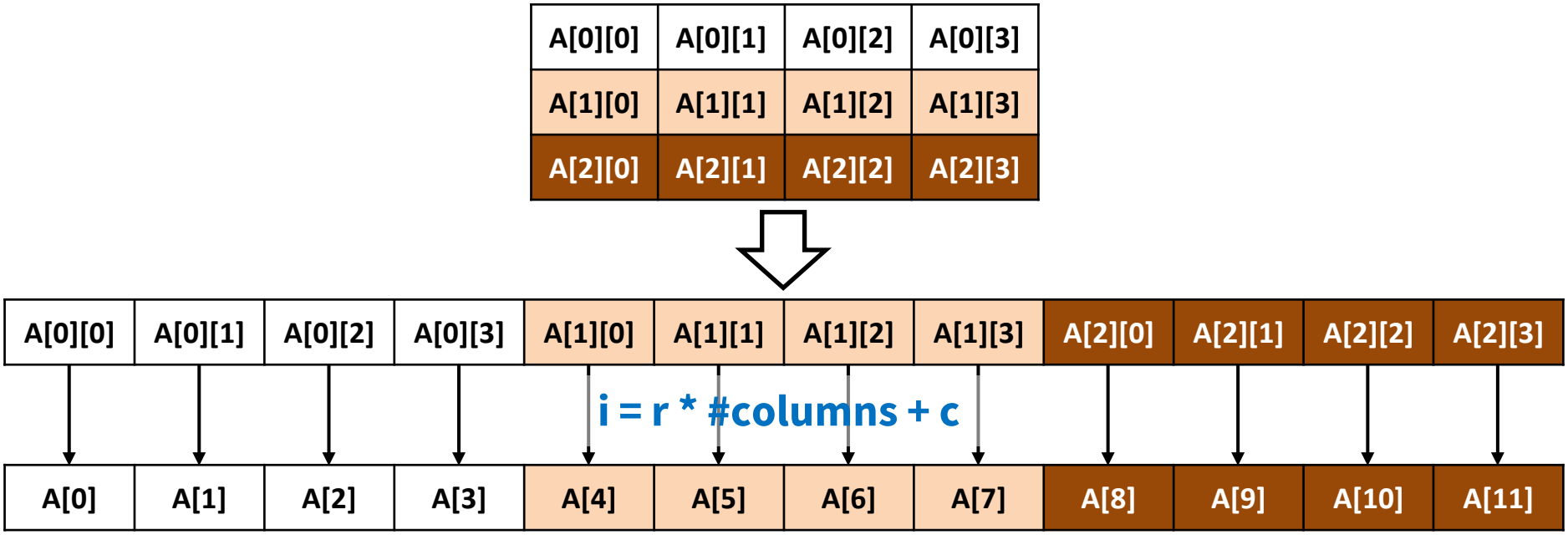
# Cộng 2 ma trận

- **Cài đặt tuần tự:** dễ ☺
- **Cài đặt song song:** không khó ☺
  - Để mỗi thread phụ trách tính một phần tử trong ma trận kết quả
  - Với dữ liệu 2D, một cách tự nhiên là ta sẽ tổ chức các thread dưới dạng block 2D và grid 2D

- Một xem xét cuối cùng trước khi bắt đầu code: ta nên lưu trữ ma trận như thế nào?
  - Lưu dưới dạng mảng 2D (con trỏ 2 cấp)

    Sẽ có rất nhiều vấn đề (nếu không tin thì bạn có thể thử …)
  - Lưu dưới dạng mảng 1D (con trỏ 1 cấp) bằng cách nối các dòng lại với nhau

    Sẽ giúp mọi chuyện đơn giản hơn …




Khi cài đặt các hàm làm với mảng 2D (được lưu dưới dạng mảng 1D):

- Ta có thể xem mảng là mảng 1D luôn, nhưng cách làm không hoạt động trong mọi trường hợp
- Một cách khác tổng quát hơn là xem mảng là mảng 2D, tính chỉ số dòng và cột bình thường, và khi cần truy xuất thì ta chuyển chỉ số dòng và cột sang chỉ số của mảng 1D bên dưới (ta sẽ làm theo cách này)

# Cộng 2 ma trận: live coding

# Buổi này

- ☐ Làm với dữ liệu 2D: cộng 2 ma trận
- ☐ Truy vấn thông tin device

# Truy vấn thông tin device

**Trong** chương trình CUDA, ta có thể có nhu cầu truy xuất thông tin device:

- Chẳng hạn, muốn truy xuất số thread tối đa / block và số block tối đa / grid của device và thiết lập block size và grid size phù hợp với device đó → khi thay đổi device sẽ không phải sửa lại code
- Hoặc muốn truy xuất thông tin của tất cả các device hiện có trên máy và chọn ra device mạnh nhất để chạy
- Hoặc chỉ đơn giản là muốn truy xuất các thông tin của device và in ra màn hình để xem

# Truy vấn thông tin device

**Truy vấn số lượng device**

```
int devCount;
cudaGetDeviceCount(&devCount);
```

**Chọn device để sử dụng (trong trường hợp có nhiều device), vd device 0**

```
cudaSetDevice(0);
```

**Truy vấn thông tin của một device, vd device 0**

```
cudaDeviceProp devProp;
cudaGetDeviceProperties(&devProp, 0)
```

Xem các thành phần của struct cudaDeviceProp ở đây. Vd, devProp.maxThreadsPerBlock cho biết số lượng thread tối đa / block

# Kết quả truy vấn device của mình

Device 0: GeForce GTX 560 Ti
Compute capability: 2.1
GMEM: 1.00 GB
Max number of threads per block: 1024
Max size of each dimension of a block: 1024 x 1024 x 64
Max size of each dimension of a grid: 65535 x 65535 x 65535
…

# Kết quả truy vấn device của Colab (năm ngoái)

Device 0: Tesla P100-PCIE-16GB
Compute capability: 6.0
GMEM: 15.90 GB
Max number of threads per block: 1024
Max size of each dimension of a block: 1024 x 1024 x 64
Max size of each dimension of a grid: 2147483647 x 65535 x 65535
…

# Kết quả truy vấn device của Colab (năm nay)

Device 0: Tesla K80
Compute capability: 3.7
GMEM: 11.17 GB
Max number of threads per block: 1024
Max size of each dimension of a block: 1024 x 1024 x 64
Max size of each dimension of a grid: 2147483647 x 65535 x 65535
…