# Giới thiệu CUDA C/C++ (phần 2)

Trần Trung Kiên – Phạm Trọng Nghĩa
ttkien@fit.hcmus.edu.vn
ptnghia@fit.hcmus.edu.vn


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

# 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 la

**Cấu trúc của một chươn**

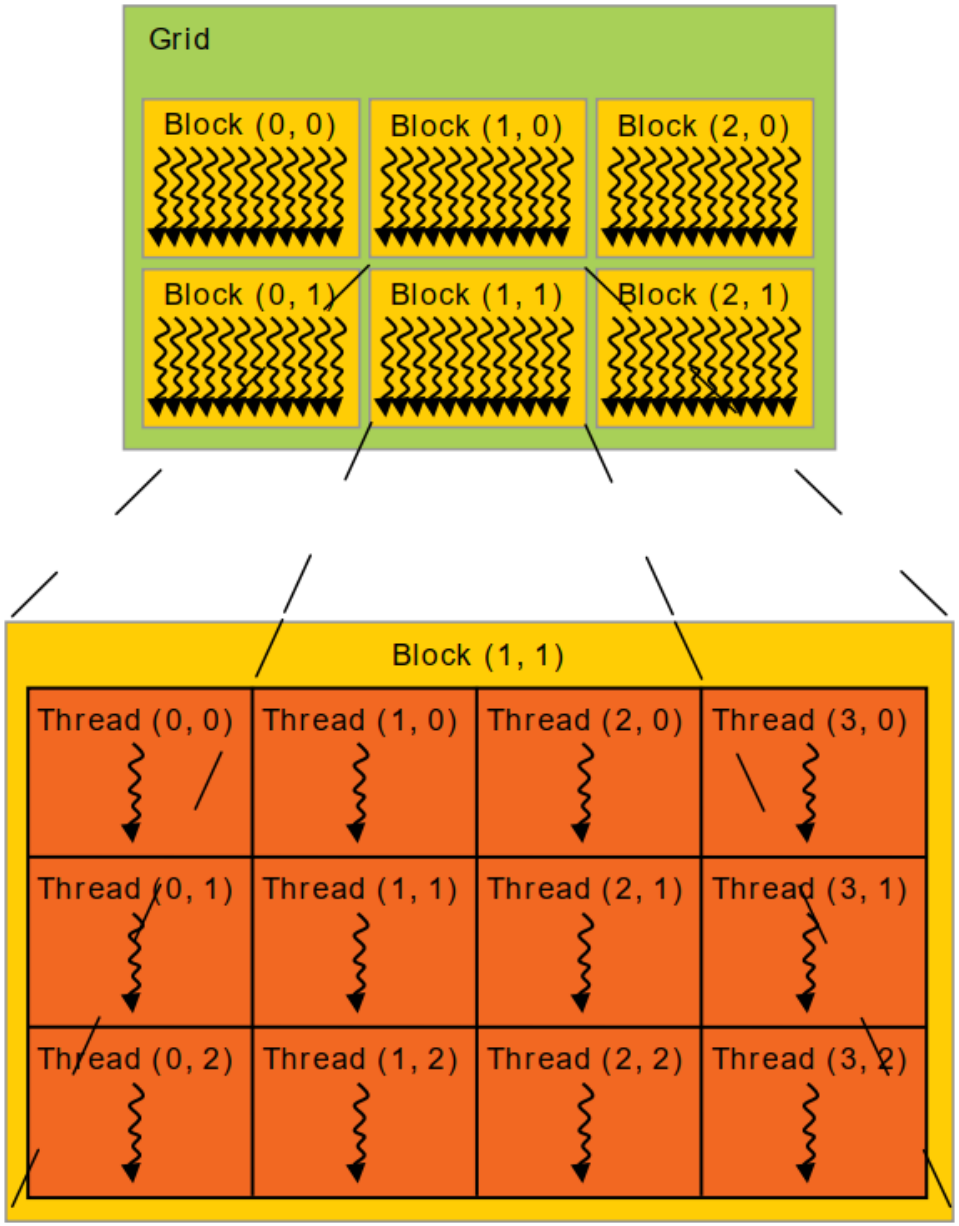
- Những phần tính toá
  những phần tính toán
  device (GPU)
- Từ host, để nhờ device
  - Host cấp phát các
    **`cudaMalloc`**
  - Host chép các dữ liệ
    device bằng hàm **`cuda`**
  - Host gọi hàm kernel
  - Host chép kết quả từ
  - Host giải phóng cá
    **`cudaFree`**

- Hàm kernel được thực thi song song ở device bởi rất nhiều tiểu trình (thread)
- Các tiểu trình này được tổ chức thành 2 cấp: lưới tiểu trình (grid) gồm các khối tiểu trình (block) có cùng kích thước, khối tiểu trình gồm các tiểu trình → Khi host gọi hàm kernel, host cần cho biết là lưới tiểu trình gồm bao nhiêu khối tiểu trình và mỗi khối gồm bao nhiêu tiểu trình
- Ở phía device, trong hàm kernel: mỗi tiểu trình 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

# Hôm nay

- Cộng 2 ma trận
  - Grid 2D và block 2D
  - Grid 1D và block 1D
  - Grid 2D và block 1D
- Truy vấn thông tin device

# Thread Hierarchy



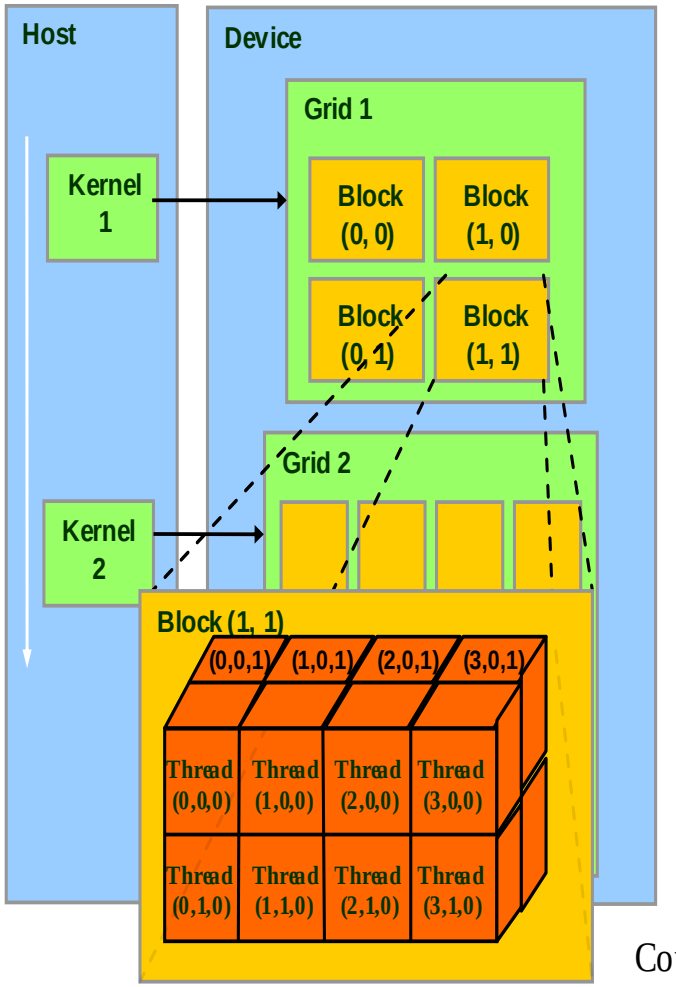
# Thread Hierarchy

```
dim3 threads_per_block( threads per block in x-dim,
                        threads per block in y-dim,
                        threads per block in z-dim);

dim3 blocks_in_grid( grid blocks in x-dim,
                     grid blocks in y-dim,
                     grid blocks in z-dim );
```

- threadIdx: 1D,2D, 3D. blockIdx: 1D,2D, 3D
- Số lượng thread tối đa mỗi block là giới hạn:
  - Current GPU: 1024 threads *

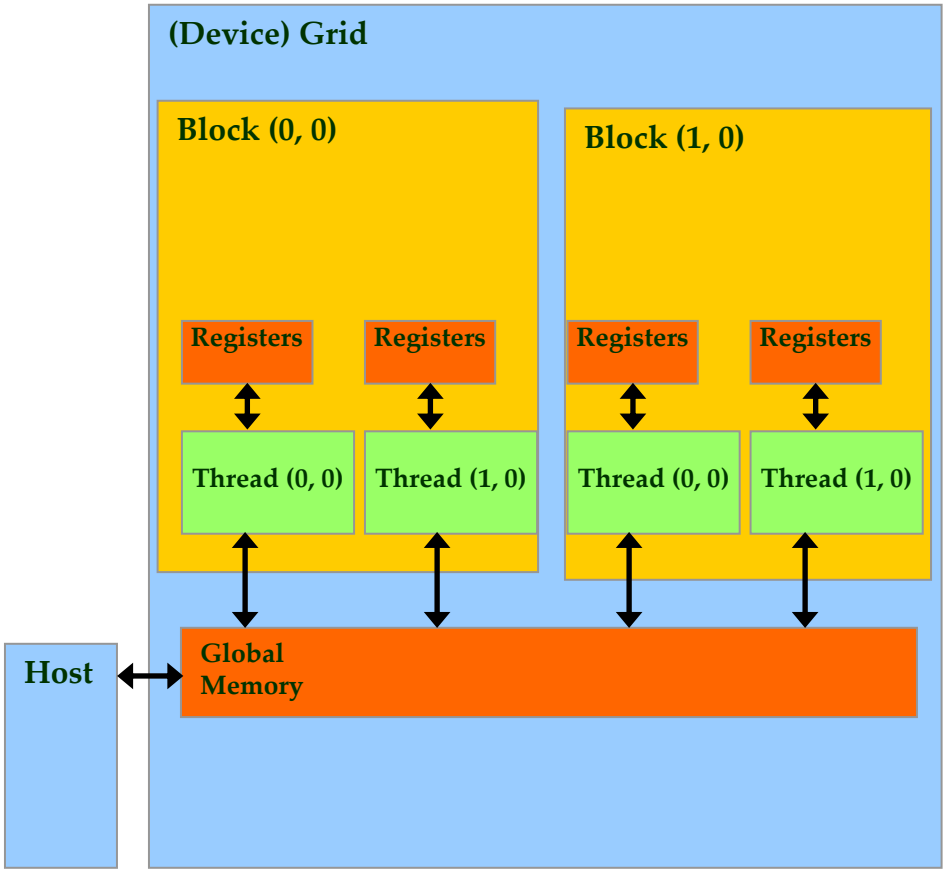
```
threads_per_block( 16, 16, 1 );   256   ✔
threads_per_block( 32, 32, 1 );   1024  ✔
threads_per_block( 64, 64, 1 );   4096  ✗
```

# Thread Hierarchy




Courtesy: NDVIA

# Sơ lược về bộ nhớ CUDA

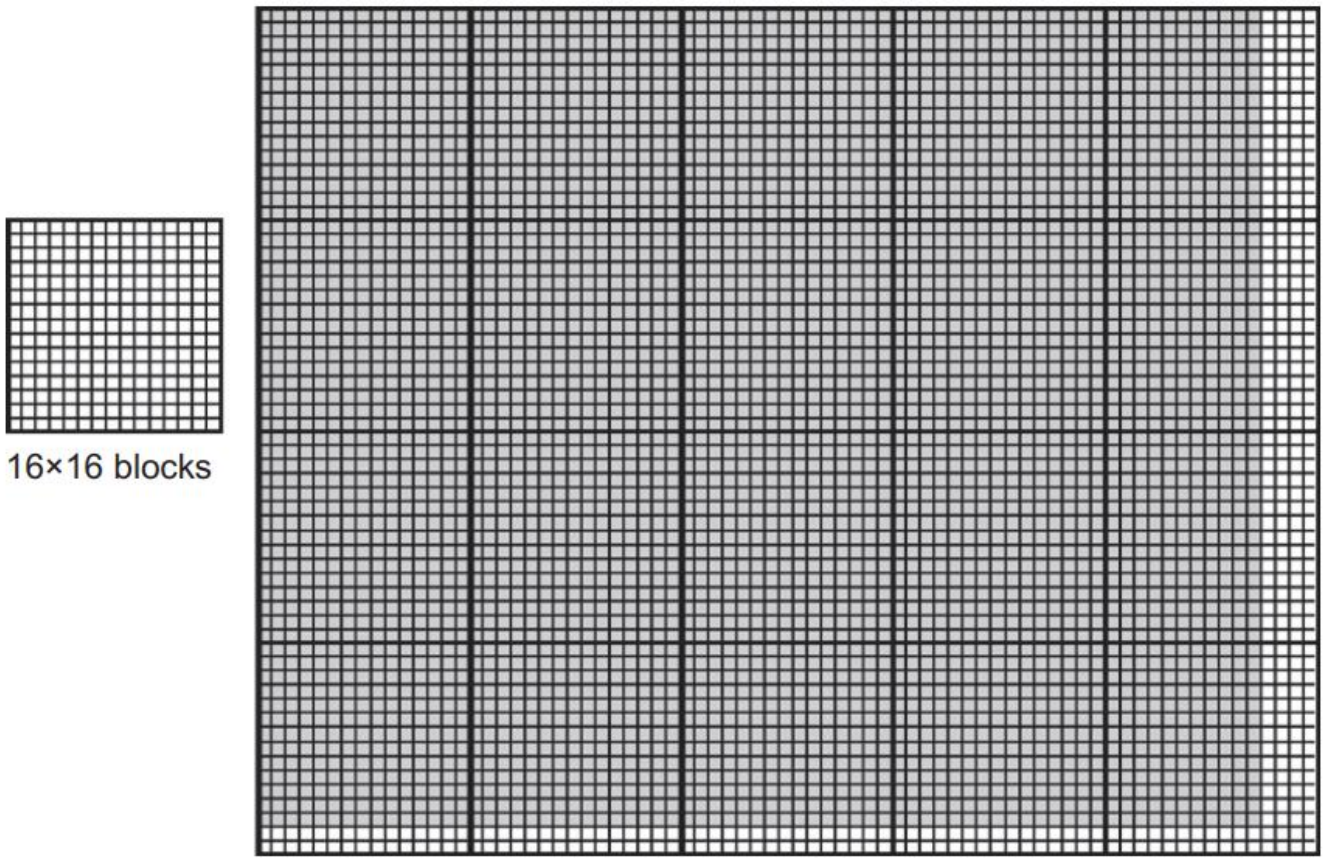


# CUDA C keywords for function declaration

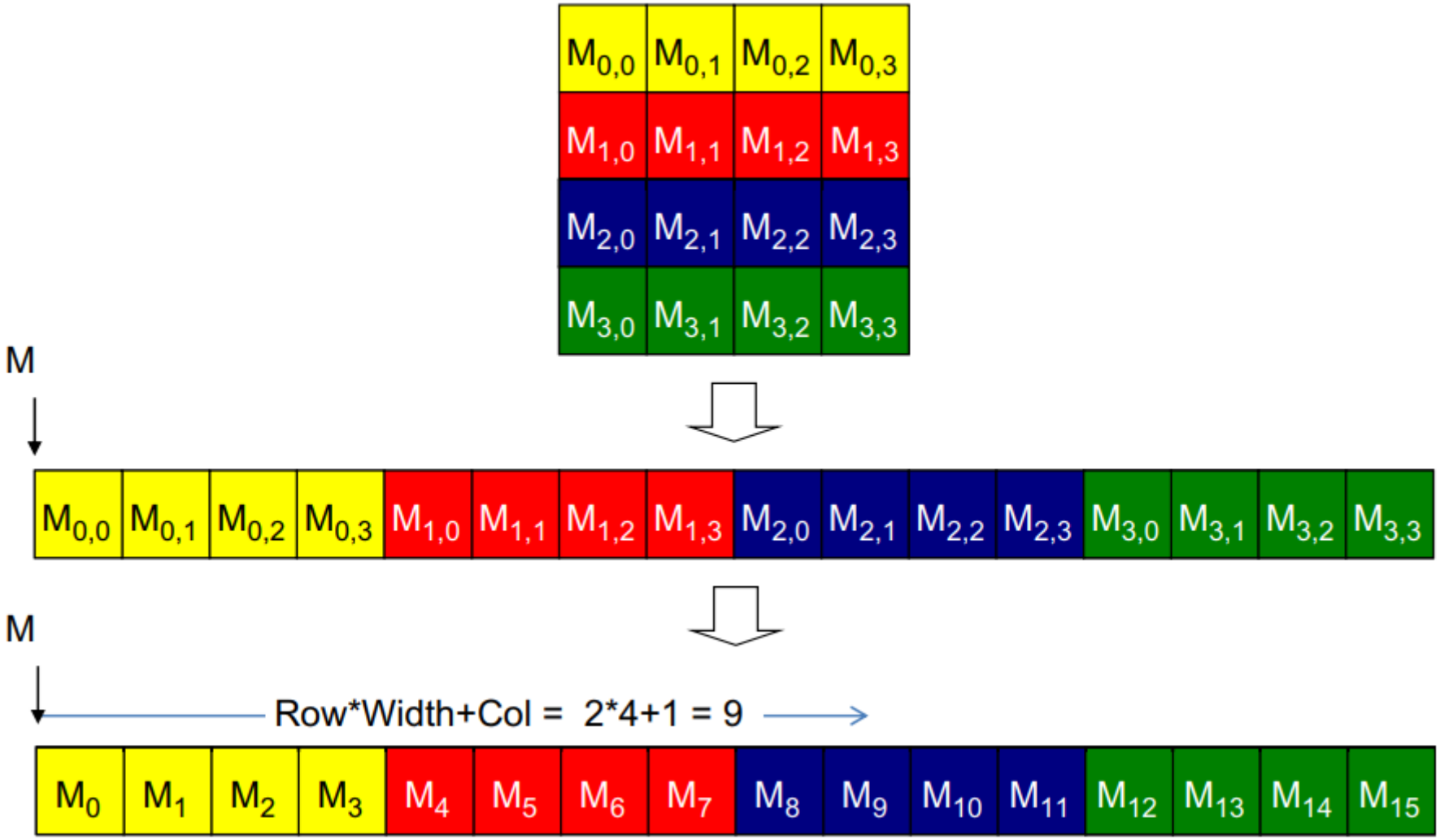
| | Executed on the: | Only callable from the: |
|---|---|---|
| `__device__ float DeviceFunc()` | device | device |
| `__global__ void KernelFunc()` | device | host |
| `__host__ float HostFunc()` | host | host |

# Thread to data

- Giả sử có bức ảnh: 76 x 62
- Block: 16x16. Cần grid có kích thước bao nhiêu.



# Major row layout

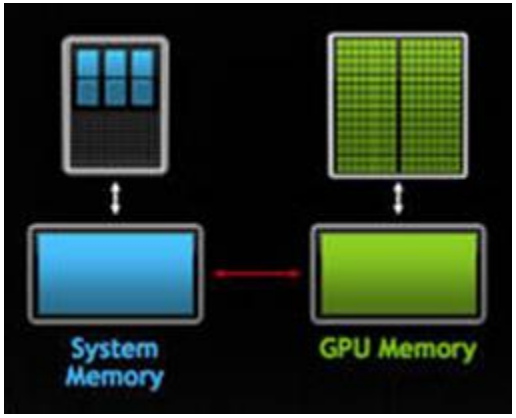


```
int main(int argc, char **argv)
{
   int nx, ny; // Matrix size (num columns, num rows)
   float *in1, *in2; // Input matrixes
   float *out;  // Output matrix

   // Input data into nx, ny
   …

   // Allocate memories for in1, in2, out
   …

   // Input data into in1, in2
   …

   // Add matrixes (on host)
   addMatOnHost(in1, in2, out, nx , ny);

   // Free memories
   …

   return 0;
}
```

Lưu trữ ma trận dưới dạng mảng một chiều bằng cách nối các dòng của ma trận lại với nhau

```
void addMatOnHost(float *in1, float *in2, float *out,
                                          int nx, int ny)
{
   for (int iy = 0; iy < ny; iy++)
   {
      for (int ix = 0; ix < nx; ix++)
      {
         int i = iy * nx + ix;
         out[i] = in1[i] + in2[i];
      }
   }
}
```

```
int main(int argc, char **argv)
{
   int nx, ny; // Matrix size (num columns, num rows)
   float *in1, *in2; // Input matrixes
   float *out;  // Output matrix

   // Input data into nx, ny
   ...

   // Allocate memories for in1, in2, out
   ...

   // Input data into in1, in2
   ...

   // Add matrixes (on host)
   addMatOnHost(in1, in2, out, nx , ny);

   // Free memories
   ...

   return 0;
}
```




```
// Host allocates memories on device
...

// Host copies data to device memories
...

// Host invokes kernel function to add matrixes on device
...

// Host copies result from device memory
...

// Host frees device memories
...
```

```
// Host allocates memories on device
float *d_in1, *d_in2, *d_out;
CHECK(cudaMalloc(&d_in1, nx * ny * sizeof(float)));
CHECK(cudaMalloc(&d_in2, nx * ny * sizeof(float)));
CHECK(cudaMalloc(&d_out, nx * ny * sizeof(float)));

// Host copies data to device memories
CHECK(cudaMemcpy(d_in1, in1, nx * ny * sizeof(float), cudaMemcpyHostToDevice));
CHECK(cudaMemcpy(d_in2, in2, nx * ny * sizeof(float), cudaMemcpyHostToDevice));

// Host invokes kernel function to add matrixes on device
...

// Host copies result from device memory
CHECK(cudaMemcpy(out, d_out, nx * ny * sizeof(float), cudaMemcpyDeviceToHost));

// Host frees device memories
CHECK(cudaFree(d_in1));
CHECK(cudaFree(d_in2));
CHECK(cudaFree(d_out));
```

# Cách "bày binh bố trận" 1 – grid 2D và block 2D

- Mỗi thread (tiểu trình) sẽ phụ trách tính một phần tử trong ma trận kết quả
- Dùng block (khối tiểu trình) có kích thước $32 \times 32$ → grid (lưới tiểu trình) sẽ có kích thước? (kích thước của ma trận là $nx \times ny$)

```
// Host allocates memories on device
float *d_in1, *d_in2, *d_out;
CHECK(cudaMalloc(&d_in1, nx * ny * sizeof(float)));
CHECK(cudaMalloc(&d_in2, nx * ny * sizeof(float)));
CHECK(cudaMalloc(&d_out, nx * ny * sizeof(float)));

// Host copies data to device memories
CHECK(cudaMemcpy(d_in1, in1, nx * ny * sizeof(float), cudaMemcpyHostToDevice));
CHECK(cudaMemcpy(d_in2, in2, nx * ny * sizeof(float), cudaMemcpyHostToDevice));

// Host invokes kernel function to add matrixes on device
dim3 blockSize(32, 32);
dim3 gridSize((nx - 1) / blockSize.x + 1, (ny - 1) / blockSize.y + 1);
addMatOnDevice2D<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, nx, ny);
cudaDeviceSynchronize();
CHECK(cudaGetLastError());

// Host copies result from device memory
CHECK(cudaMemcpy(out, d_out, nx * ny * sizeof(float), cudaMemcpyDeviceToHost));

// Host frees device memories
CHECK(cudaFree(d_in1));
CHECK(cudaFree(d_in2));
CHECK(cudaFree(d_out));
```

# Trong hàm kernel

1. Mỗi tiểu trình sẽ xác định chỉ số của phần tử trong ma trận kết quả mà mình sẽ phụ trách tính toán (dựa vào blockIdx và threadIdx)
2. Nếu chỉ số này hợp lệ thì tiểu trình sẽ thực hiện tính toán

```
...
// Host invokes kernel function to add matrixes on device
dim3 blockSize(32, 32);
dim3 gridSize((nx - 1) / blockSize.x + 1, (ny - 1) / blockSize.y + 1);
addMatOnDevice2D<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, nx, ny);
...
```

```
__global__ void addMatOnDevice2D(float *in1, float *in2, float *out, int nx, int ny)
{
   int ix =
   int iy =

   if (ix < nx && iy < ny)
   {
      int i =
      out[i] = in1[i] + in2[i];
   }
}
```

```
...
// Host invokes kernel function to add matrixes on device
dim3 blockSize(32, 32);
dim3 gridSize((nx - 1) / blockSize.x + 1, (ny - 1) / blockSize.y + 1);
addMatOnDevice2D<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, nx, ny);
...
```

```
__global__ void addMatOnDevice2D(float *in1, float *in2, float *out, int nx, int ny)
{
   int ix = threadIdx.x + blockIdx.x * blockDim.x;
   int iy = threadIdx.y + blockIdx.y * blockDim.y;

   if (ix < nx && iy < ny)
   {
      int i = iy * nx + ix;
      out[i] = in1[i] + in2[i];
   }
}
```

# Thí nghiệm

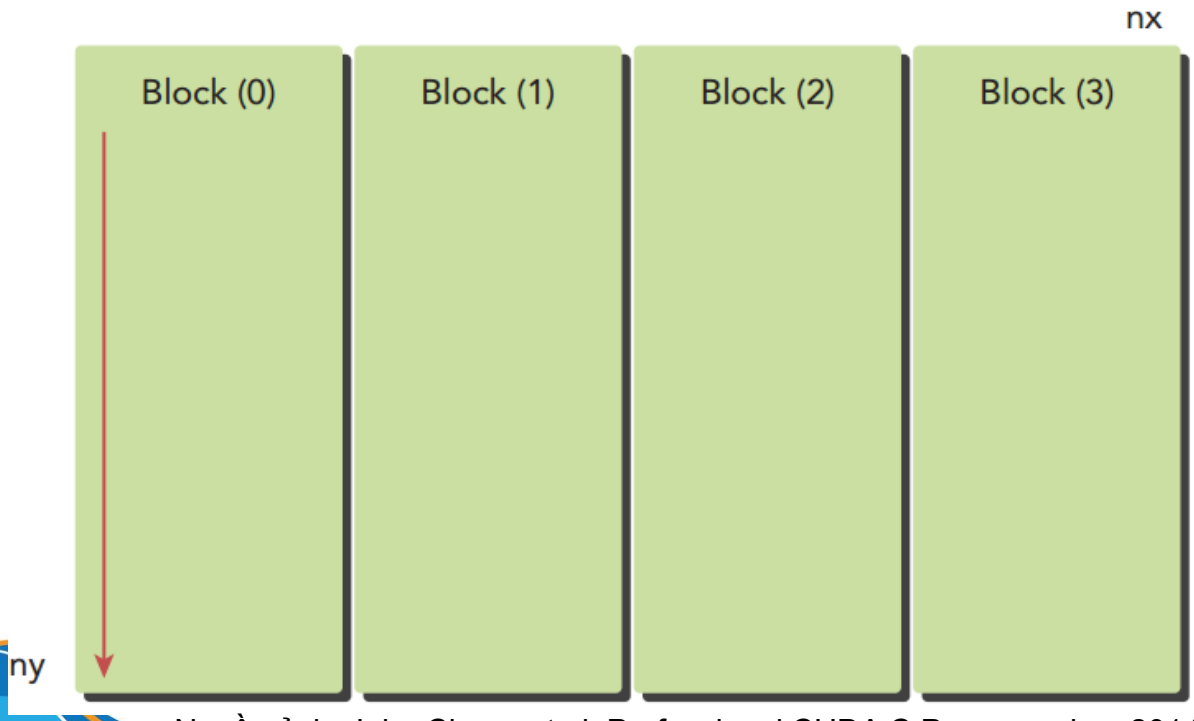
- Kích thước ma trận: $(2^{13} + 1) \times (2^{13} + 1)$
- Phát sinh ngẫu nhiên giá trị của các ma trận đầu vào trong [0, 1]
- So sánh thời gian chạy giữa các hàm:
  - addMatOnHost
  - addMatOnDevice2D với các kích thước block khác nhau
- GPU: GeForce GTX 560 Ti (compute capability 2.1)

# Kết quả thí nghiệm

| Function | Block size | Grid size | Time (ms) |
|---|---|---|---|
| addMatOnHost | | | 257.141 |
| addMatOnDevice2D | 32 x 32 | 257 x 257 | 14.676 |
| | 16 x 32 | 513 x 257 | 12.267 |
| | 32 x 16 | 257 x 513 | **9.911** |
| | 16 x 16 | 513 x 513 | 10.904 |

# Cách "bày binh bố trận" 2 – grid 1D và block 1D

- Mỗi thread sẽ phụ trách tính **một cột gồm ny phần tử** trong ma trận kết quả
- Dùng block có kích thước 32 → grid sẽ có kích thước? (kích thước của ma trận là nx × ny)




Nguồn ảnh: John Cheng et al. Professional CUDA C Programming. 2014

```
...
// Host invokes kernel function to add matrixes on device
dim3 blockSize(32);
dim3 gridSize((nx - 1) / blockSize.x + 1);
addMatOnDevice1D<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, nx, ny);
...
```

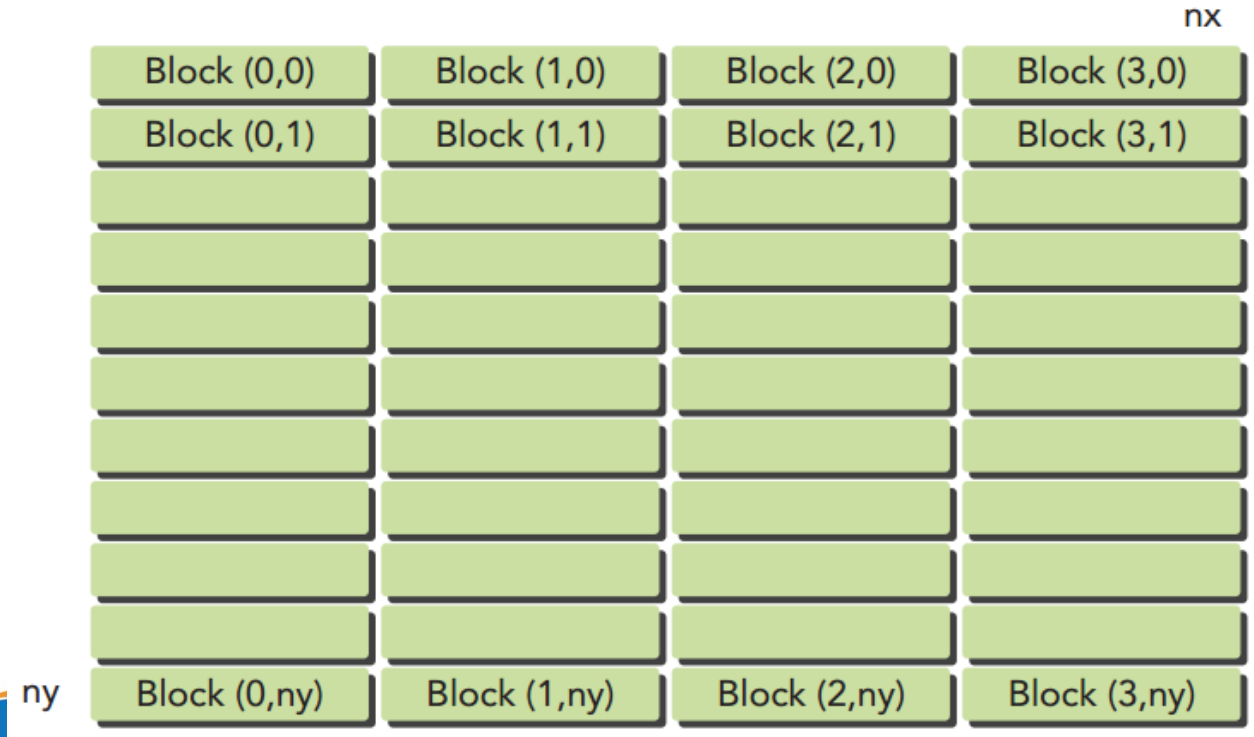
```
__global__ void addMatOnDevice1D(float *in1, float *in2, float *out, int nx, int ny)
{
   int ix = threadIdx.x + blockIdx.x * blockDim.x;

   if (ix < nx)
   {
      for (int iy = 0; iy < ny; iy++)
      {
         int i = iy * nx + ix;
         out[i] = in1[i] + in2[i];
      }
   }
}
```

# Kết quả thí nghiệm

| Function | Block size | Grid size | Time (ms) |
|---|---|---|---|
| addMatOnHost | | | 257.141 |
| addMatOnDevice2D | 32 x 32 | 257 x 257 | 14.676 |
| | 16 x 32 | 513 x 257 | 12.267 |
| | 32 x 16 | 257 x 513 | **9.911** |
| | 16 x 16 | 513 x 513 | 10.904 |
| addMatOnDevice1D | 32 x 1 | 257 x 1 | 20.464 |
| | 64 x 1 | 129 x 1 | **13.700** |
| | 128 x 1 | 65 x 1 | 15.998 |

# Cách "bày binh bố trận" 3 – grid 2D và block 1D

- Là một trường hợp của grid 2D và block 2D
- Mỗi thread sẽ phụ trách tính một phần tử trong ma trận kết quả
- Dùng block có kích thước 32 → grid sẽ có kích thước? (kích thước của ma trận là nx × ny)




Nguồn ảnh: John Cheng et al. Professional CUDA C Programming. 2014

```
...
// Host invokes kernel function to add matrixes on device
dim3 blockSize(32);
dim3 gridSize((nx - 1) / blockSize.x + 1, ny);
addMatOnDeviceMix<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, nx, ny);
...
```

```
__global__ void addMatOnDeviceMix(float *in1, float *in2, float *out, int nx, int ny)
{
   int ix = threadIdx.x + blockIdx.x * blockDim.x;
   int iy = blockIdx.y;

   if (ix < nx && iy < ny)
   {
      int i = iy * nx + ix;
      out[i] = in1[i] + in2[i];
   }
}
```

# Kết quả thí nghiệm

| Function | Block size | Grid size | Time (ms) |
|---|---|---|---|
| addMatOnHost | | | 257.141 |
| addMatOnDevice2D | 32 x 32 | 257 x 257 | 14.676 |
| | 16 x 32 | 513 x 257 | 12.267 |
| | 32 x 16 | 257 x 513 | **9.911** |
| | 16 x 16 | 513 x 513 | 10.904 |
| addMatOnDevice1D | 32 x 1 | 257 x 1 | 20.464 |
| | 64 x 1 | 129 x 1 | **13.700** |
| | 128 x 1 | 65 x 1 | 15.998 |
| addMatOnDevice2D | 32 x 1 | 257 x 8193 | 24.891 |
| | 64 x 1 | 129 x 8193 | 13.765 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Kết quả thí nghiệm

| Function | Block size | Grid size | Time (ms) |
|---|---|---|---|
| addMatOnHost | | | 257.141 |
| addMatOnDevice2D | 32 x 32 | 257 x 257 | 14.676 |
| | 16 x 32 | 513 x 257 | 12.267 |
| | 32 x 16 | 257 x 513 | **9.911** |
| | 16 x 16 | 513 x 513 | 10.904 |
| addMatOnDevice1D | 32 x 1 | 257 x 1 | 20.464 |
| | 64 x 1 | 129 x 1 | **13.700** |
| | 128 x 1 | 65 x 1 | 15.998 |
| addMatOnDevice2D | 32 x 1 | 257 x 8193 | 24.891 |
| | 64 x 1 | 129 x 8193 | 13.765 |
| | 128 x 1 | 65 x 8193 | 8.817 |
| | | | |
| | | | |
| | | | |
| | | | |

# Kết quả thí nghiệm

| Function | Block size | Grid size | Time (ms) |
|---|---|---|---|
| addMatOnHost | | | 257.141 |
| addMatOnDevice2D | 32 x 32 | 257 x 257 | 14.676 |
| | 16 x 32 | 513 x 257 | 12.267 |
| | 32 x 16 | 257 x 513 | **9.911** |
| | 16 x 16 | 513 x 513 | 10.904 |
| addMatOnDevice1D | 32 x 1 | 257 x 1 | 20.464 |
| | 64 x 1 | 129 x 1 | **13.700** |
| | 128 x 1 | 65 x 1 | 15.998 |
| addMatOnDevice2D | 32 x 1 | 257 x 8193 | 24.891 |
| | 64 x 1 | 129 x 8193 | 13.765 |
| | 128 x 1 | 65 x 8193 | 8.817 |
| | 256 x 1 | 33 x 8193 | 7.958 |
| | | | |
| | | | |
| | | | |

# Kết quả thí nghiệm

| Function | Block size | Grid size | Time (ms) |
|---|---|---|---|
| addMatOnHost | | | 257.141 |
| addMatOnDevice2D | 32 x 32 | 257 x 257 | 14.676 |
| | 16 x 32 | 513 x 257 | 12.267 |
| | 32 x 16 | 257 x 513 | **9.911** |
| | 16 x 16 | 513 x 513 | 10.904 |
| addMatOnDevice1D | 32 x 1 | 257 x 1 | 20.464 |
| | 64 x 1 | 129 x 1 | **13.700** |
| | 128 x 1 | 65 x 1 | 15.998 |
| addMatOnDevice2D | 32 x 1 | 257 x 8193 | 24.891 |
| | 64 x 1 | 129 x 8193 | 13.765 |
| | 128 x 1 | 65 x 8193 | 8.817 |
| | 256 x 1 | 33 x 8193 | 7.958 |
| | 512 x 1 | 17 x 8193 | 8.195 |
| | | | |
| | | | |

# Kết quả thí nghiệm

| Function | Block size | Grid size | Time (ms) |
|---|---|---|---|
| addMatOnHost | | | 257.141 |
| addMatOnDevice2D | 32 x 32 | 257 x 257 | 14.676 |
| | 16 x 32 | 513 x 257 | 12.267 |
| | 32 x 16 | 257 x 513 | **9.911** |
| | 16 x 16 | 513 x 513 | 10.904 |
| addMatOnDevice1D | 32 x 1 | 257 x 1 | 20.464 |
| | 64 x 1 | 129 x 1 | **13.700** |
| | 128 x 1 | 65 x 1 | 15.998 |
| addMatOnDevice2D | 32 x 1 | 257 x 8193 | 24.891 |
| | 64 x 1 | 129 x 8193 | 13.765 |
| | 128 x 1 | 65 x 8193 | 8.817 |
| | 256 x 1 | 33 x 8193 | 7.958 |
| | 512 x 1 | 17 x 8193 | 8.195 |
| | 1024 x 1 | 9 x 8193 | 12.685 |
| | 2048 x 1 | | |

# Kết quả thí nghiệm

| Function | Block size | Grid size | Time (ms) |
|---|---|---|---|
| addMatOnHost | | | 257.141 |
| addMatOnDevice2D | 32 x 32 | 257 x 257 | 14.676 |
| | 16 x 32 | 513 x 257 | 12.267 |
| | 32 x 16 | 257 x 513 | **9.911** |
| | 16 x 16 | 513 x 513 | 10.904 |
| addMatOnDevice1D | 32 x 1 | 257 x 1 | 20.464 |
| | 64 x 1 | 129 x 1 | **13.700** |
| | 128 x 1 | 65 x 1 | 15.998 |
| addMatOnDevice~~2D~~Mix | 32 x 1 | 257 x 8193 | ~~24.891~~ 24.320 |
| | 64 x 1 | 129 x 8193 | ~~13.765~~ 13.538 |
| | 128 x 1 | 65 x 8193 | ~~8.817~~ 8.612 |
| | 256 x 1 | 33 x 8193 | ~~**7.958**~~ **7.850** |
| | 512 x 1 | 17 x 8193 | ~~8.195~~ 8.083 |
| | 1024 x 1 | 9 x 8193 | ~~12.685~~ 12.110 |
| | 2048 x 1 | | |

# Một số ý rút ra từ phần thí nghiệm

- Với một bài toán, có nhiều cách để cài đặt hàm kernel; các cách cài đặt khác nhau sẽ cho các hiệu năng khác nhau
- Với một cách cài đặt hàm kernel, các kích thước block khác nhau sẽ cho các hiệu năng khác nhau

# 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)
```

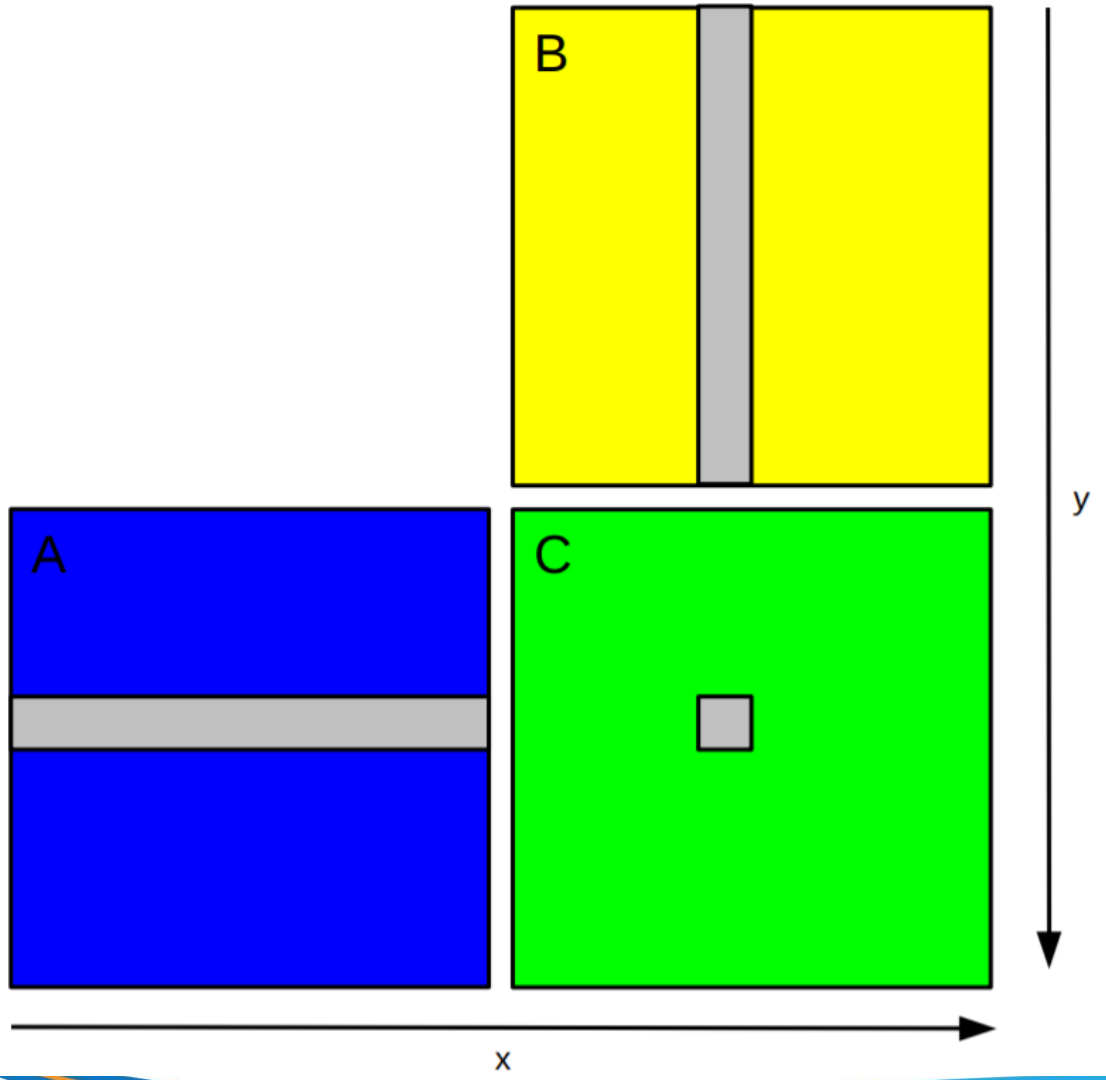
Xem các thành phần của struct cudaDeviceProp ở đây. Vd, devProp.maxThreadsPerBlock cho biết số lượng thread tối đa / block

# Kết quả truy vấn device của mình

Device 0: GeForce GTX 560 Ti

| | |
|---|---|
| Compute capability: | 2.1 |
| Total amount of global memory: | 1.00 GB |
| Maximum number of threads per block: | 1024 |
| Maximum sizes of each dimension of a block: | 1024 x 1024 x 64 |
| Maximum sizes of each dimension of a grid: | 65535 x 65535 x 65535 |
| … | |

# Nhân ma trận



# Nhân ma trận