Giới thiệu CUDA C/C++

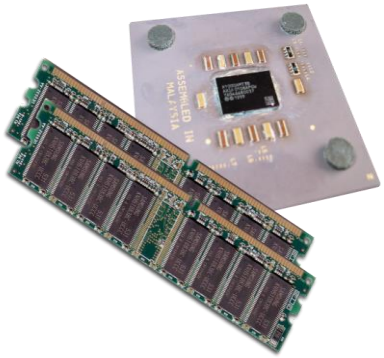
Trần Trung Kiên
ttkien@fit.hcmus.edu.vn

Cập nhật lần cuối: 23/09/2021

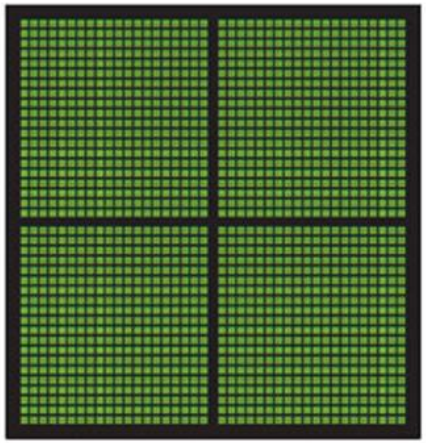
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

fit@hcmus

CPU vs GPU



CPU
MULTIPLE CORES



GPU
THOUSANDS OF CORES

Tối ưu hóa **độ trễ (latency)**

Tối ưu hóa **băng thông (throughput)**

Nguồn ảnh: http://www.nvidia.com/object/what-is-gpu-computing.html

CUDA C/C++: là ngôn ngữ C/C++ được mở rộng, cho phép viết chương trình chạy trên CPU (những phần tính toán tuần tự) + GPU (những phần tính toán song song ở cấp độ lớn)

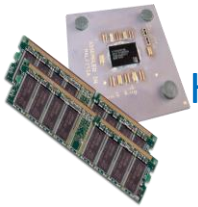
```
#include <iostream>
#include <algorithm>

using namespace std;

#define N          1024
#define RADIUS     3
#define BLOCK_SIZE 16

__global__ void stencil_1d(int *in, int *out) {
        __shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
        int gindex = threadIdx.x + blockIdx.x * blockDim.x;
        int lindex = threadIdx.x + RADIUS;

        // Read input elements into shared memory
        temp[lindex] = in[gindex];
        if (threadIdx.x < RADIUS) {
                temp[lindex - RADIUS] = in[gindex - RADIUS];
                temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
        }

        // Synchronize (ensure all the data is available)
        __syncthreads();

        // Apply the stencil
        int result = 0;
        for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
                result += temp[lindex + offset];

        // Store the result
        out[gindex] = result;
}

void fill_ints(int *x, int n) {
        fill_n(x, n, 1);
}

int main(void) {
        int *in, *out;          // host copies of a, b, c
        int *d_in, *d_out;      // device copies of a, b, c
        int size = (N + 2*RADIUS) * sizeof(int);

        // Alloc space for host copies and setup values
        in  = (int *)malloc(size); fill_ints(in,  N + 2*RADIUS);
        out = (int *)malloc(size); fill_ints(out, N + 2*RADIUS);

        // Alloc space for device copies
        cudaMalloc((void **)&d_in,  size);
        cudaMalloc((void **)&d_out, size);

        // Copy to device
        cudaMemcpy(d_in,  in,  size, cudaMemcpyHostToDevice);
        cudaMemcpy(d_out, out, size, cudaMemcpyHostToDevice);

        // Launch stencil_1d() kernel on GPU
        stencil_1d<<<N/BLOCK_SIZE,BLOCK_SIZE>>>(d_in + RADIUS,
d_out + RADIUS);

        // Copy result back to host
        cudaMemcpy(out, d_out, size, cudaMemcpyDeviceToHost);

        // Cleanup
        free(in); free(out);
        cudaFree(d_in); cudaFree(d_out);
        return 0;
}
```

parallel function

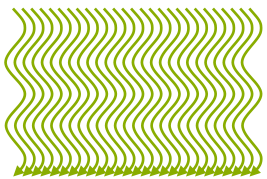
serial code

parallel code

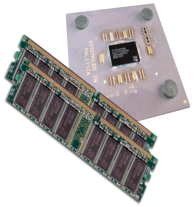
serial code



Host = CPU
(+ bộ nhớ)



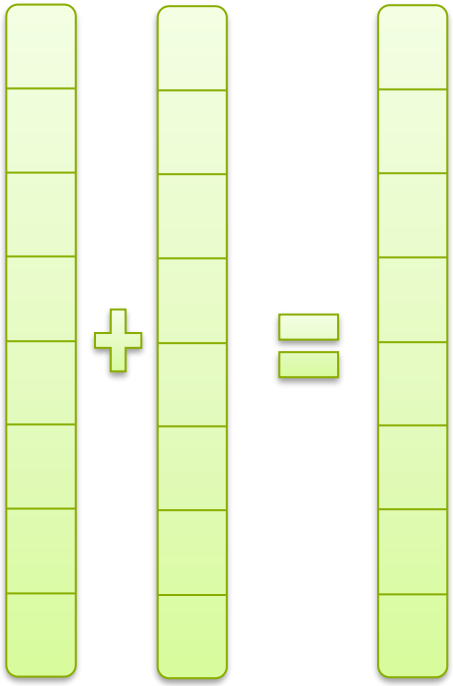
Device = GPU



Nguồn ảnh: NVIDIA. CUDA C/C++ Basics

Chương trình CUDA đơn giản: cộng 2 véc-tơ

- Cộng 2 véc-tơ tuần tự bằng host
- Cộng 2 véc-tơ song song bằng device: để mỗi thread ở device phụ trách tính một phần tử trong véc-tơ kết quả, và các thread này cùng chạy song song với nhau
- Ai thắng?

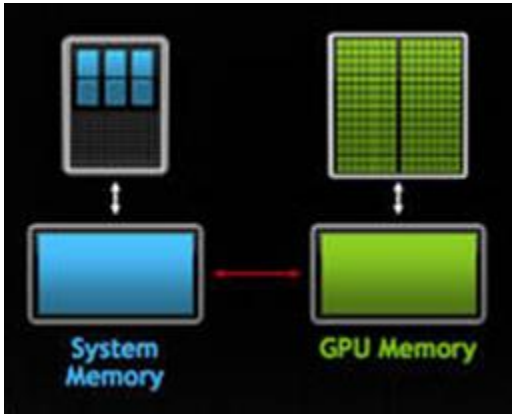


Nguồn ảnh: NVIDIA. CUDA C/C++ Basics

```
int main(int argc, char **argv)
{
   int n; // Vector size
   float *in1, *in2; // Input vectors
   float *out;  // Output vector

   // Input data into n
   …

   // Allocate memories for in1, in2, out
   …

   // Input data into in1, in2
   …

   // Add vectors (on host)
   addVecOnHost(in1, in2, out, n);

   // Free memories
   …

   return 0;
}
```

```
void addVecOnHost(float* in1, float* in2, float* out, int n)
{
   for (int i = 0; i < n; i++)
      out[i] = in1[i] + in2[i];
}
```

```
int main(int argc, char **argv)
{
   int n; // Vector size
   float *in1, *in2; // Input vectors
   float *out;  // Output vector

   // Input data into n
   …

   // Allocate memories for in1, in2, out
   …

   // Input data into in1, in2
   …

   // Add vectors (on host)
   addVecOnHost(in1, in2, out, n);

   // Free memories
   …

   return 0;
}
```




```
// Host allocates memories on device
…

// Host copies data to device memories
…

// Host invokes kernel function to add vectors on device
…

// Host copies result from device memory
…

// Host frees device memories
…
```

Nguồn ảnh: Mark Harris. Unified Memory in CUDA 6

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

// Host copies data to device memories
…

// Host invokes kernel function to add vectors on device
…

// Host copies result from device memory
…

// Host frees device memories
…
```

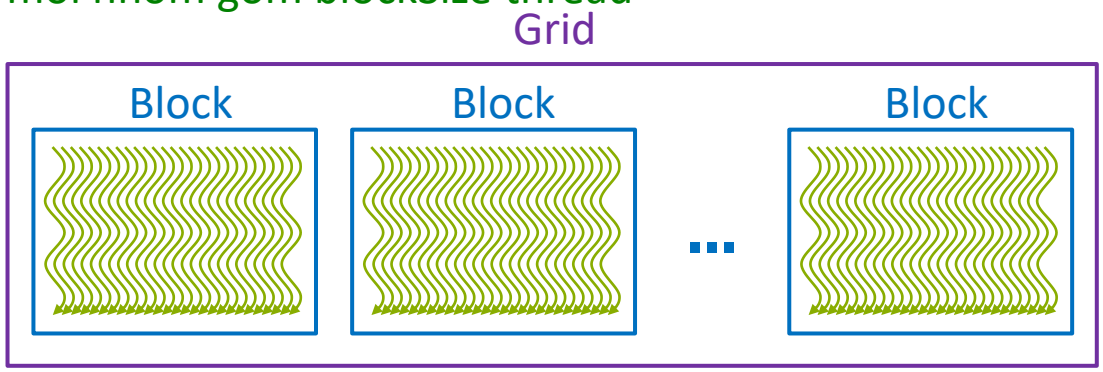
```
// Host allocates memories on device
float *d_in1, *d_in2, *d_out;
cudaMalloc(&d_in1,  n * sizeof(float));
cudaMalloc(&d_in2,  n * sizeof(float));
cudaMalloc(&d_out,  n * sizeof(float));

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

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

// Host copies result from device memory
...

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

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

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

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

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

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

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

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

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

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

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

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

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

// Host invokes kernel function to add vectors on device
dim3 blockSize(256); // Để đơn giản, hiện giờ bạn cứ xem blockSize là 1 con số
dim3 gridSize((n - 1) / blockSize.x + 1); // Tương tự, bạn cứ xem gridSize là 1 con số
addVecOnDevice<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, n);
```

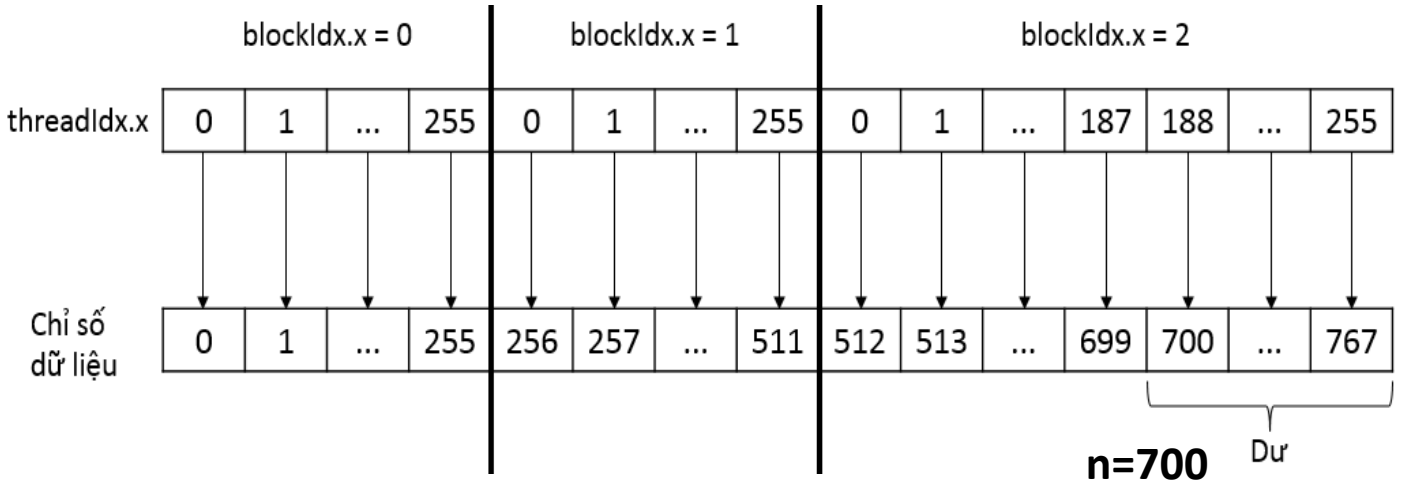
Câu lệnh này tạo ra ở device một đống các thread (gọi là một **grid**) cùng thực thi song song hàm addVecOnDevice; các thread này được tổ chức thành gridSize nhóm (**block**), mỗi nhóm gồm blockSize thread



```
...
// Host invokes kernel function to add vectors on device
dim3 blockSize(256);
dim3 gridSize((n - 1) / blockSize.x + 1);
addVecOnDevice<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, n);
...
```

Bắt buộc phải là void

```
__global__ void addVecOnDevice(float* in1, float* in2, float* out, int n)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   if (i < n)
      out[i] = in1[i] + in2[i];
}
```



Tính "không đồng bộ" của hàm kernel

Một tính chất của hàm kernel là **"không đồng bộ" (asynchronous)**: sau khi host gọi hàm kernel ở device, host sẽ được tự do tiếp tục làm các công việc của mình mà không phải chờ hàm kernel ở device thực hiện xong

```
...
// Host invokes kernel function to add vectors on device
dim3 blockSize(256);
dim3 gridSize((n - 1) / blockSize.x + 1);
addVecOnDevice<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, n);

// Host copies result from device memory
cudaMemcpy(out, d_out, n * sizeof(float), cudaMemcpyDeviceToHost); // OK?
```

OK, do hàm cudaMemcpy sẽ bắt host chờ cho tới khi hàm kernel chạy xong rồi mới copy

Tính "không đồng bộ" của hàm kernel

```
…
// Host invokes kernel function to add vectors on device
dim3 blockSize(256);
dim3 gridSize((n - 1) / blockSize.x + 1);
double start = seconds(); // seconds is my function to get the current time
addVecOnDevice<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, n);
double time = seconds() - start; // OK?
…
```

Tính "không đồng bộ" của hàm kernel

```
…
// Host invokes kernel function to add vectors on device
dim3 blockSize(256);
dim3 gridSize((n - 1) / blockSize.x + 1);
double start = seconds(); // seconds is my function to get the current time
addVecOnDevice<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, n);
cudaDeviceSynchronize(); // Host waits here until device has completed its work
double time = seconds() - start; // ✓
…
```

Kiểm lỗi khi gọi các hàm CUDA API

- Nhiều khi có lỗi nhưng chương trình CUDA vẫn chạy bình thường và cho ra kết quả sai

  → không biết sai ở đâu cả ☹

  → để biết sai ở đâu thì nên luôn tiến hành kiểm lỗi khi gọi các hàm CUDA API

- Để cho tiện, có để định nghĩa một macro kiểm lỗi:

```
#define CHECK(call)                                                          \
{                                                                            \
   cudaError_t err = call;                                                   \
   if (err != cudaSuccess)                                                   \
   {                                                                         \
      printf("%s in %s at line %d!\n", cudaGetErrorString(err), __FILE__, __LINE__); \
      exit(EXIT_FAILURE);                                                    \
   }                                                                         \
}
```

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

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

// Host invokes kernel function to add vectors on device
dim3 blockSize(256);
dim3 gridSize((n - 1) / blockSize.x + 1);
addVecOnDevice<<<gridSize, blockSize>>>(d_in1, d_in2, d_out, n);

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

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

Kiểm lỗi hàm kernel?

Đọc ở đây, mục “Handling CUDA Errors”

Thí nghiệm: host vs device

- Phát sinh ngẫu nhiên giá trị của các véc-tơ đầu vào trong [0, 1]
- So sánh thời gian chạy giữa host (hàm `addVecOnHost`) và device (hàm `addVecOnDevice`, kích thước block là 512) với các kích thước véc-tơ khác nhau
- GPU: GeForce GTX 560 Ti (compute capability 2.1)

Thí nghiệm: host vs device

Vec size	Host time (ms)	Device time (ms)	Host time / Device time
64			

Thí nghiệm: host vs device

Vec size	Host time (ms)	Device time (ms)	Host time / Device time
64	**0.001**	0.040	0.024

Thí nghiệm: host vs device

Vec size	Host time (ms)	Device time (ms)	Host time / Device time
64	**0.001**	0.040	0.024
256			

Thí nghiệm: host vs device

Vec size	Host time (ms)	Device time (ms)	Host time / Device time
64	**0.001**	0.040	0.024
256	**0.002**	0.018	0.118

Thí nghiệm: host vs device

Vec size	Host time (ms)	Device time (ms)	Host time / Device time
64	**0.001**	0.040	0.024
256	**0.002**	0.018	0.118
1024			

Thí nghiệm: host vs device

Vec size	Host time (ms)	Device time (ms)	Host time / Device time
64	**0.001**	0.040	0.024
256	**0.002**	0.018	0.118
1024	**0.006**	0.017	0.347

Thí nghiệm: host vs device

Vec size	Host time (ms)	Device time (ms)	Host time / Device time
64	**0.001**	0.040	0.024
256	**0.002**	0.018	0.118
1024	**0.006**	0.017	0.347
4096			

Thí nghiệm: host vs device

Vec size	Host time (ms)	Device time (ms)	Host time / Device time
64	**0.001**	0.040	0.024
256	**0.002**	0.018	0.118
1024	**0.006**	0.017	0.347
4096	0.030	**0.017**	1.775

Thí nghiệm: host vs device

Vec size	Host time (ms)	Device time (ms)	Host time / Device time
64	**0.001**	0.040	0.024
256	**0.002**	0.018	0.118
1024	**0.006**	0.017	0.347
4096	0.030	**0.017**	1.775
16384	0.127	**0.017**	7.403
65536	0.516	**0.055**	9.409
262144	1.028	**0.197**	5.220
1048576	3.773	**0.277**	13.619
4194304	13.870	**0.617**	22.479
16777216	55.177	**1.993**	27.683