

# GPU并行计算与CUDA编程 第2课

- 1. 并行编程的通讯模式
  - 1.1 什么是通讯模式
  - 1.2 常见通讯模式的类型和原来
- 2. GPU硬件模式
  - 2.1 GPU , SM(流处理器) , Kernel(核) , thread block(线程块) , 线程
- 3. CUDA编程模型
  - 3.1 CUDA编程模型的优点和缺点
  - 3.2 CUDA编程编程模型的一些原则
  - 3.3 CUDA内存模型
  - 3.4 同步性synchronisation和屏障barrier
  - 3.5 编程模型
- 4. 开始编写CUDA程序
  - 4.1 GPU程序的一般步骤
  - 4.2 第一个GPU程序讲解——并行求平方

# 1. 并行编程的通讯模式 ( Communication Patterns )

## 1.1 什么是通讯模式

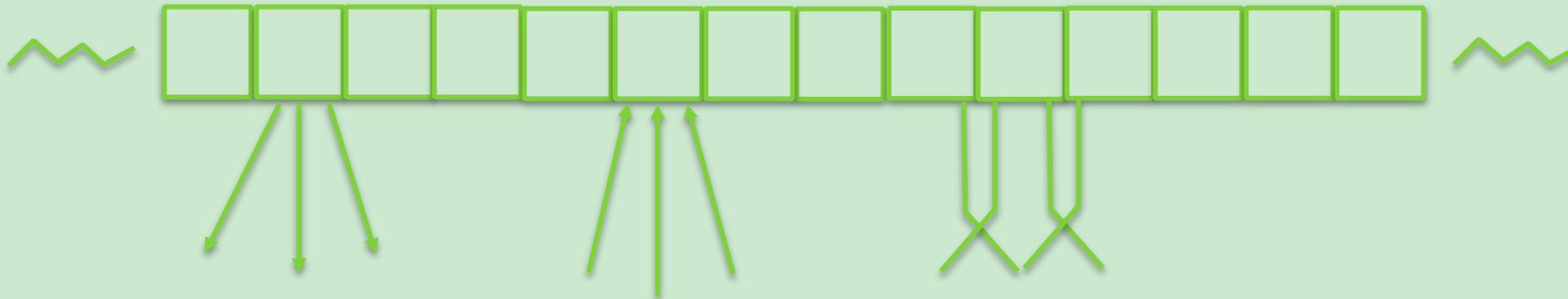
## 1.2 通讯模式的类型和原理

# 1.1 通讯模式(Communication Patterns)

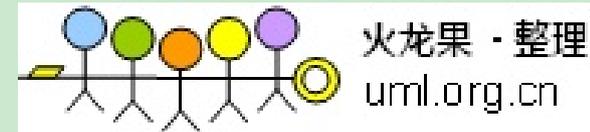
- 并行计算：非常多的线程在**合作**解决问题

Communication

- 内存：

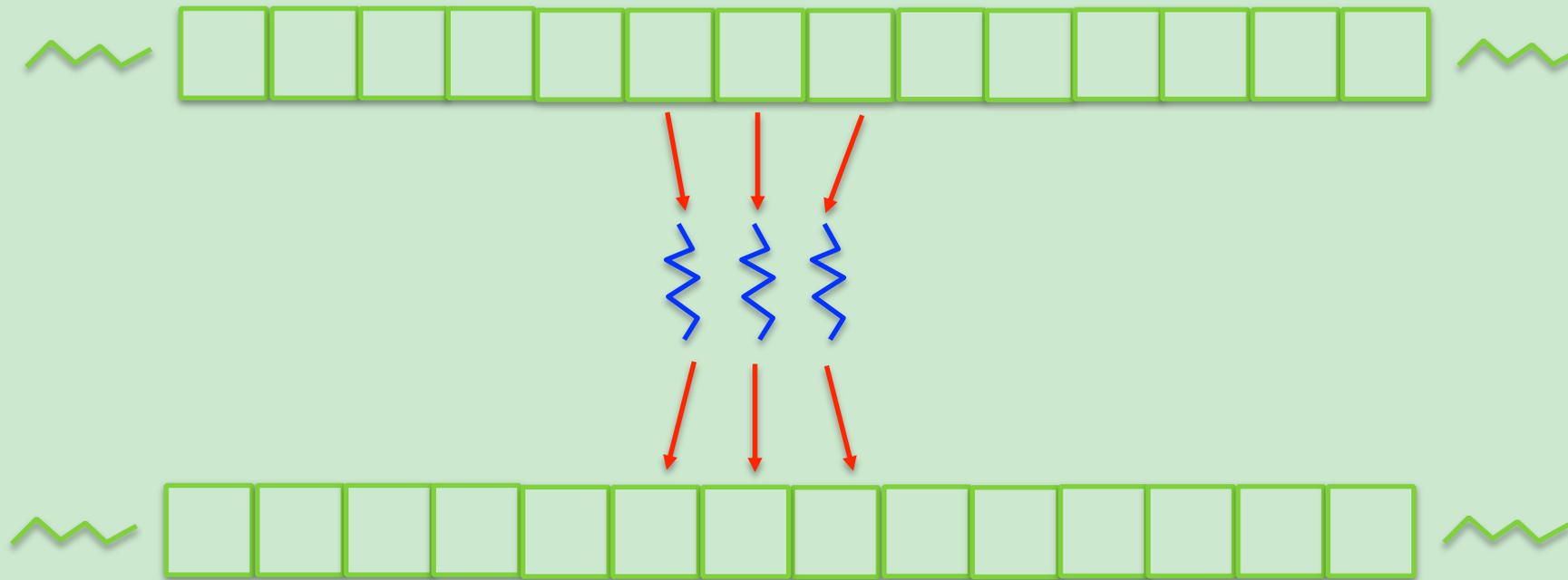


# 1.2 常见通信模式



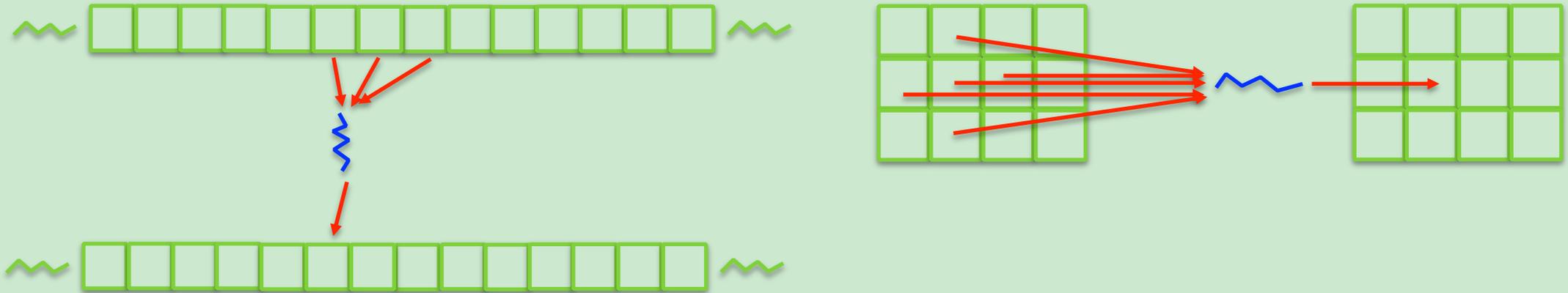
- 1. 映射Map
- 2. 聚合gather
- 3. 分散scatter
- 4. 模板stencil
- 5. 转换transpose
- 6. 压缩reduce
- 7. 重排scan/sort

## 1. 映射Map



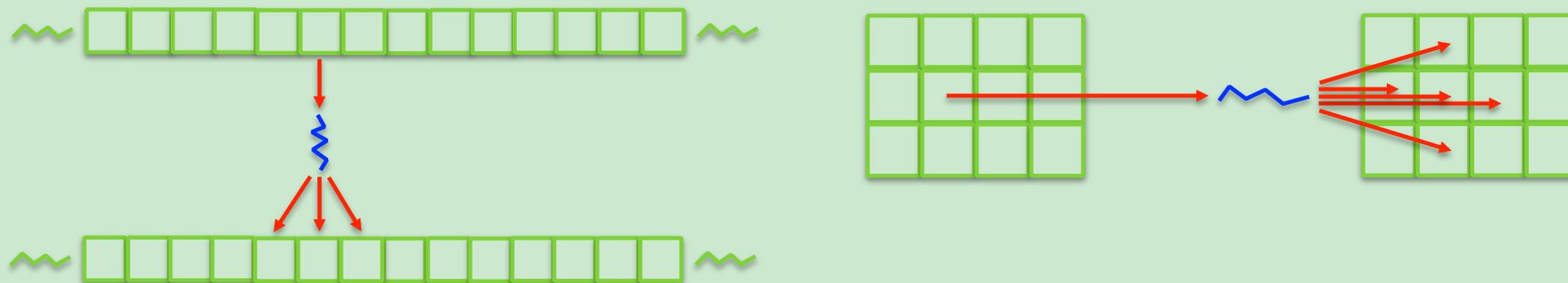
- 输入输出关系：一一对应(one-to-one)
- 例子：每个元素倍数扩大， $y[i]=3*x[i]$

- 2.聚合gatter



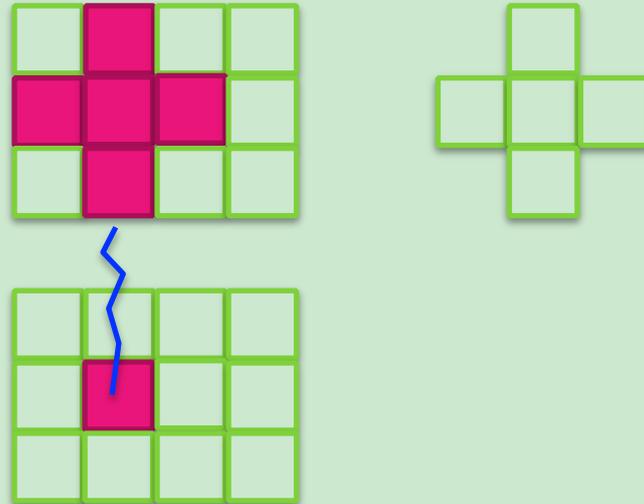
- 输入输出关系：多对一(many-to-one)
- 例子：每相邻3个元素求平均， $y[i]=(x[i-1]+x[i]+x[i+1])/3$

- 3.分散scatter



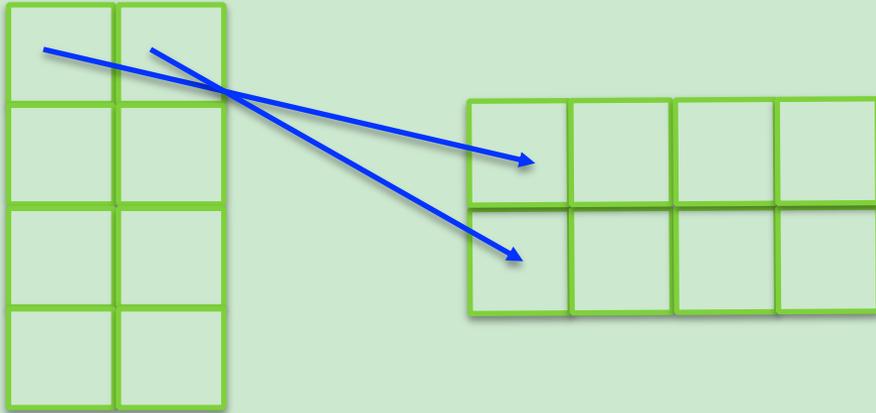
- 输入输出关系：一对多(one-to-many)

- 4.模板stencil：以固定的模式读取相邻的内存数值



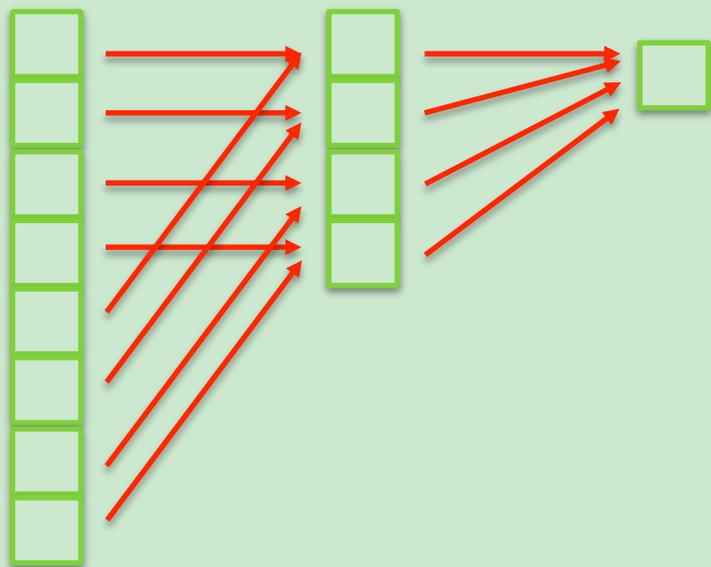
- 输入输出关系：several-to-one

- 5.转置transpose



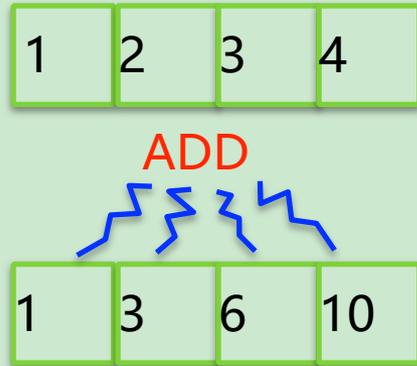
- 输入输出关系：一对一 ( one-to-one )

- 6.压缩reduce



- 输入输出关系：多对一(all-to-one)

- 7.重排scan/sort



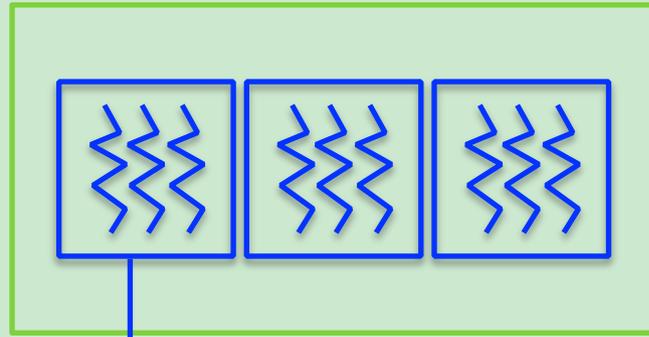
- 输入输出关系：多对多(all-to-all)

# 2.GPU硬件模式

## 2.1 GPU , SM(流处理器) , Kernel(核) , thread block(线程块) , 线程

# 线程块

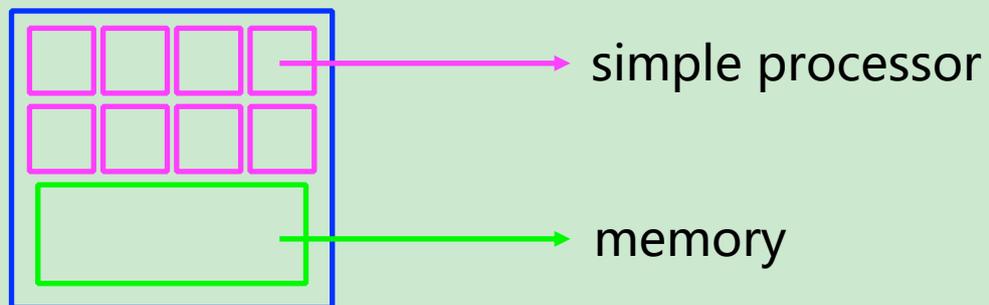
- Kernel核: 可以理解为C/C++中的一个函数function



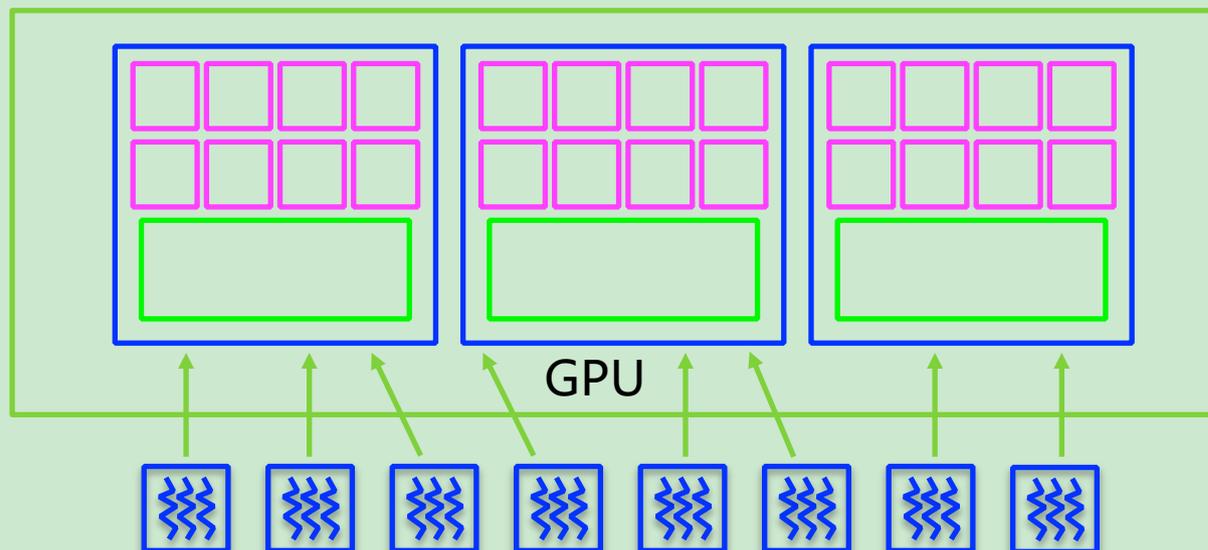
Thread Blocks: group of thread blocks to solve a function

Thread Block: a group of threads that cooperate to solve a (sub)problem  
线程块

- SM ( stream multiprocessor ) : 流处理器



- GPU:每个GPU有若干个SM，最少有1个，目前16个算大的，每个SM并行而独立运行



# 3.CUDA编程模型

3.1 CUDA编程模型的优点和缺点

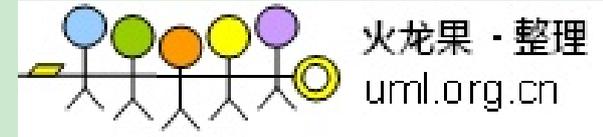
3.2 CUDA编程模型的一些原则

3.3 CUDA内存模型

3.4 同步性synchronisation和屏障barrier

3.5 编程模型

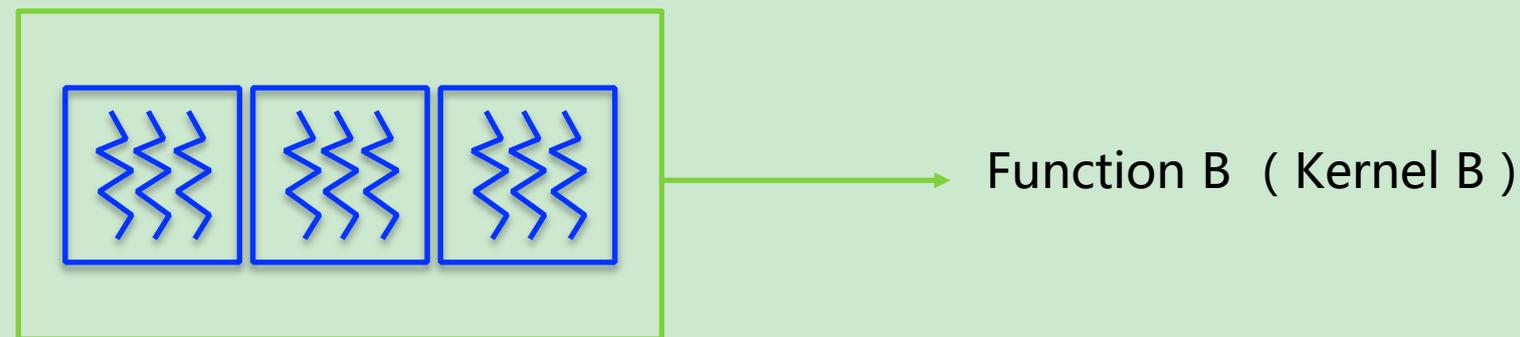
# 3.1 CUDA编程的优点和后果



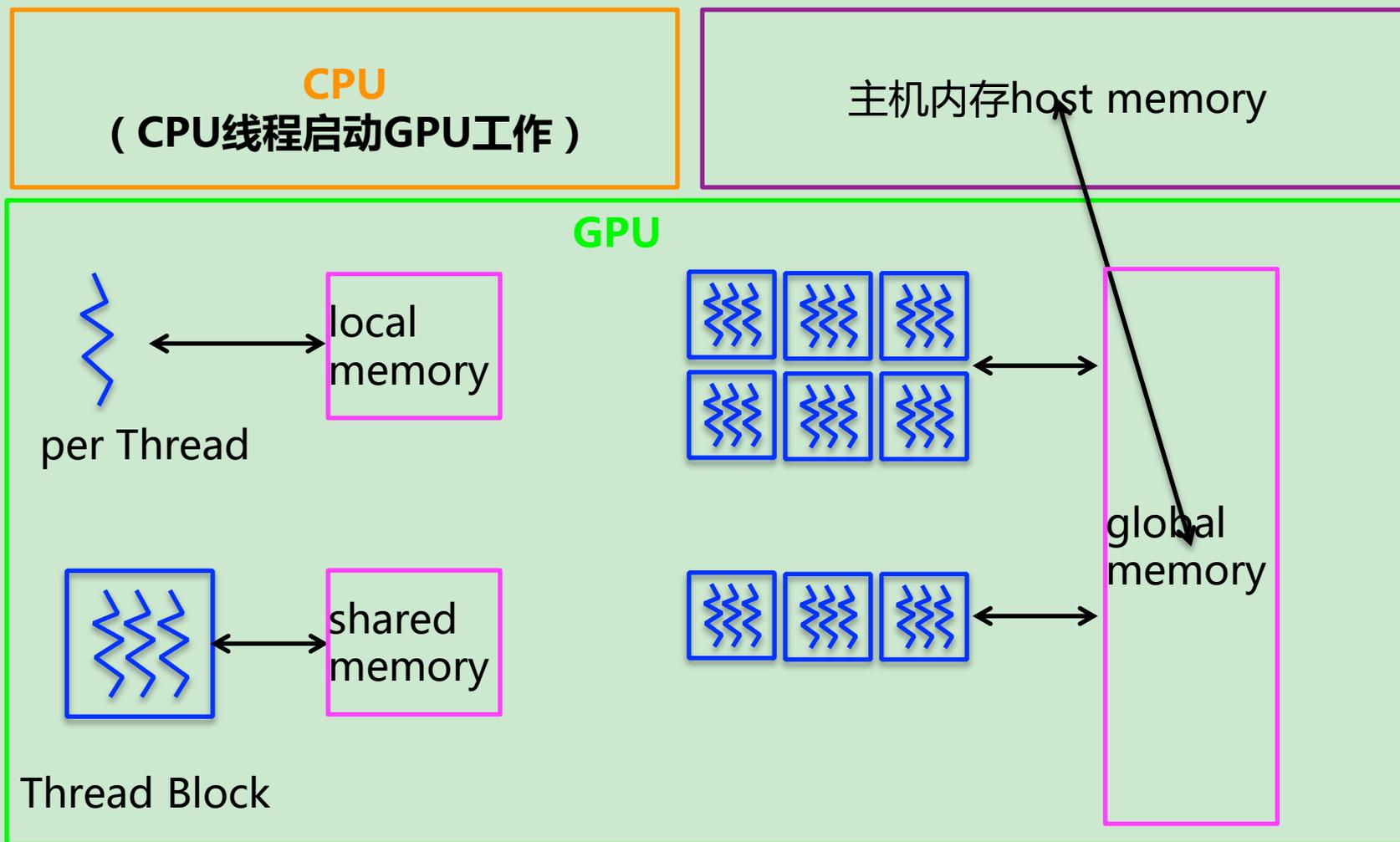
- **CUDA最大的特点：对线程块将在何处、何时运行不作保证。**
- 优点：
  1. 硬件真正有效的运行，灵活
  2. 无需要线程间互相等待
  3. 可扩展性强
- 后果：
  1. 对于那个块在那个SM上运行无法进行任何假设
  2. 无法获取块之间的明确通讯 ( hard to get communications between blocks )
    - dead lock ( 并行死锁 )
    - 线程退出

## 3.2 CUDA编程模型的原则

- 1. 所有在同一个线程块上的线程必然会在同一时间运行在同一个SM上
- 2. 同一个内核的所有线程块必然会全部完成了后，才会运行下一个内核



# 3.3 内存模型

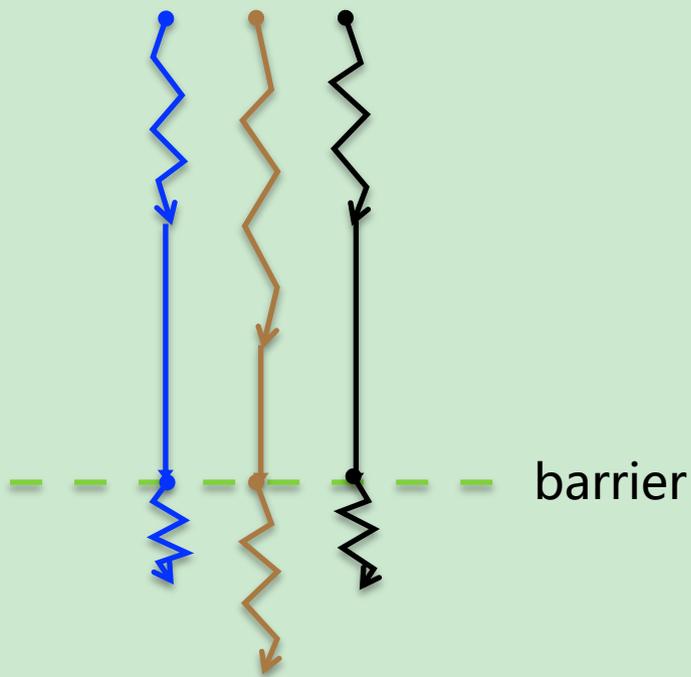


# 内存速度比较

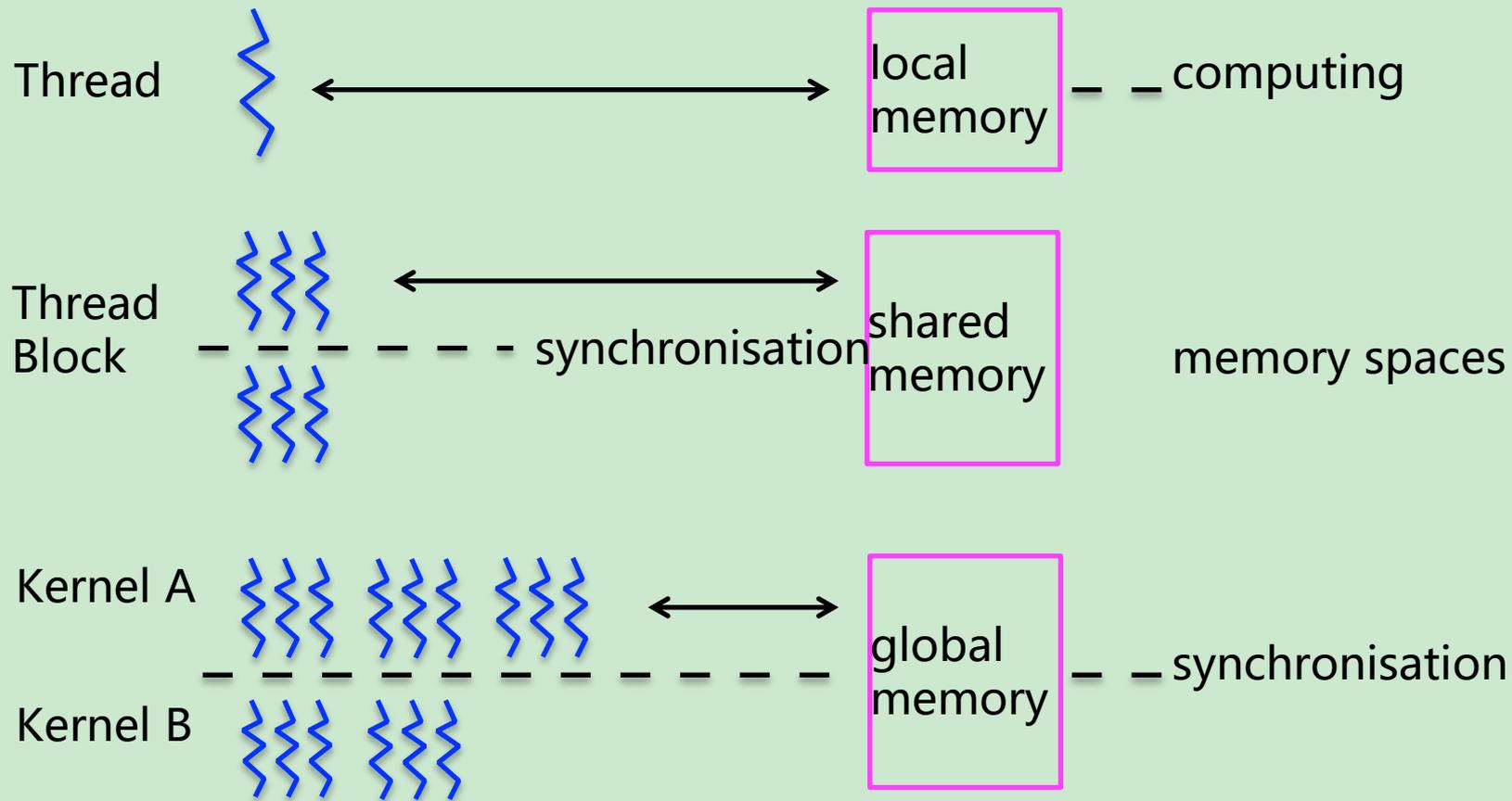


## 3.4 同步性synchronisation和屏障barrier

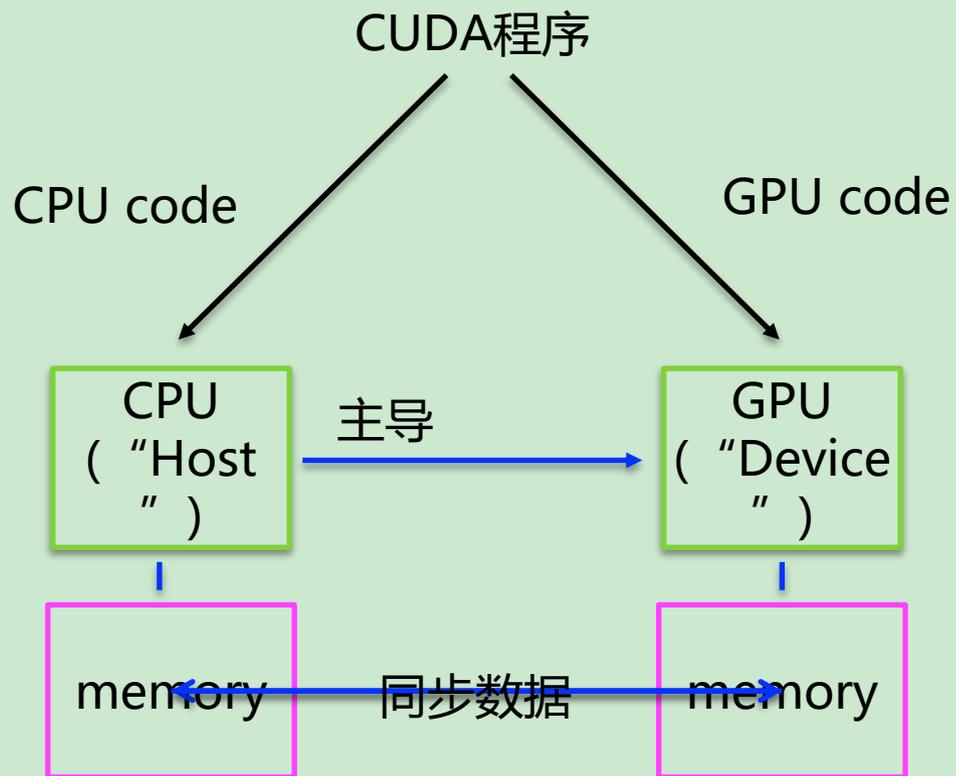
- 不同的线程在共享和全局内存中读写数据需要有先后的控制，所以引入了同步性的概念。
- 屏障的作用：用来控制多个线程的停止与等待，当所有线程都到达了屏障点，程序才继续进行。



# 3.5 CUDA编程模型



# CUDA编程模型示意图

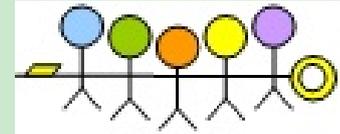


- CUDA程序中CPU是主导地位，负责完成以下的事情：
  1. 从CPU同步数据到GPU
  2. 从GPU同步数据到CPU
- ( 1、2使用cudaMemcpy )
- 3. 给GPU分配内存 ( cudaMalloc )
- 4. 加载Kernel到GPU上，launch kernel on GPU

# 4.开始编写CUDA程序

## 4.1 GPU程序的一般步骤

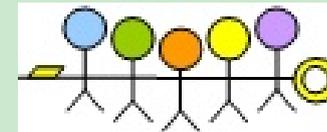
## 4.2 第一个GPU程序讲解——并行求平方



## 4.1 GPU程序一般步骤

- 1. CPU分配空间给GPU ( cudaMalloc )
  - 2. CPU复制数据给GPU ( cudaMemcpy )
  - 3. CPU加载kernels给GPU做计算
  - 4. CPU把GPU计算结果复制回来
- 
- 过程中，一般要尽量降低数据通讯的消耗，所以如果程序需要复制大量的数据到GPU，显然不是很合适使用GPU运算，最理想的情况是，每次复制的数据很小，然后运算量很大，输出的结果还是很小，复制回CPU。

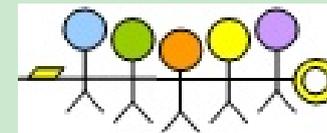
## 4.2 第一个GPU程序



- 程序讲解

```
#include <stdio.h>

__global__ void square(float* d_out, float* d_in){
    int idx = threadIdx.x;
    float f = d_in[idx];
    d_out[idx] = f * f;
}
```



```
int main(int argc, char** argv){
    const int ARRAY_SIZE = 64;
    const int ARRAY_BYTES = ARRAY_SIZE * sizeof(float);

    // generate the input array on the host
    float h_in[ARRAY_SIZE];
    for(int i=0; i<ARRAY_SIZE; i++){
        h_in[i] = float(i);
    }
    float h_out[ARRAY_SIZE];

    // declare GPU memory pointers
    float* d_in;
    float* d_out;

    // allocate GPU memory
    cudaMalloc((void**) &d_in, ARRAY_BYTES);
    cudaMalloc((void**) &d_out, ARRAY_BYTES);

    // transfer the array to GPU
    cudaMemcpy(d_in, h_in, ARRAY_BYTES, cudaMemcpyHostToDevice);
```

```
    // launch the kernel
    square<<<1, ARRAY_SIZE>>>(d_out, d_in);

    // copy back the result array to the GPU
    cudaMemcpy(h_out, d_out, ARRAY_BYTES, cudaMemcpyDeviceToHost);

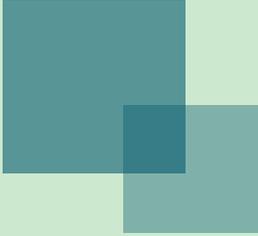
    // print out the resulting array
    for(int i=0; i<ARRAY_SIZE; i++){
        printf("%f", h_out[i]);
        printf(((i%4) != 3) ? "\t" : "\n");
    }

    // free GPU memory allocation
    cudaFree(d_in);
    cudaFree(d_out);

    return 0;
}
```

# 本周作业

- 根据课程的讲解程序square.cu改写其中kernel部分，可以改成做cube或者其他的自定义运算，然后使用nvcc编译，把成功编译并运行的截图提交即可。



# Thanks

**FAQ时间**