# CUDA编程模型

NVIDIA

扫地的小何尚

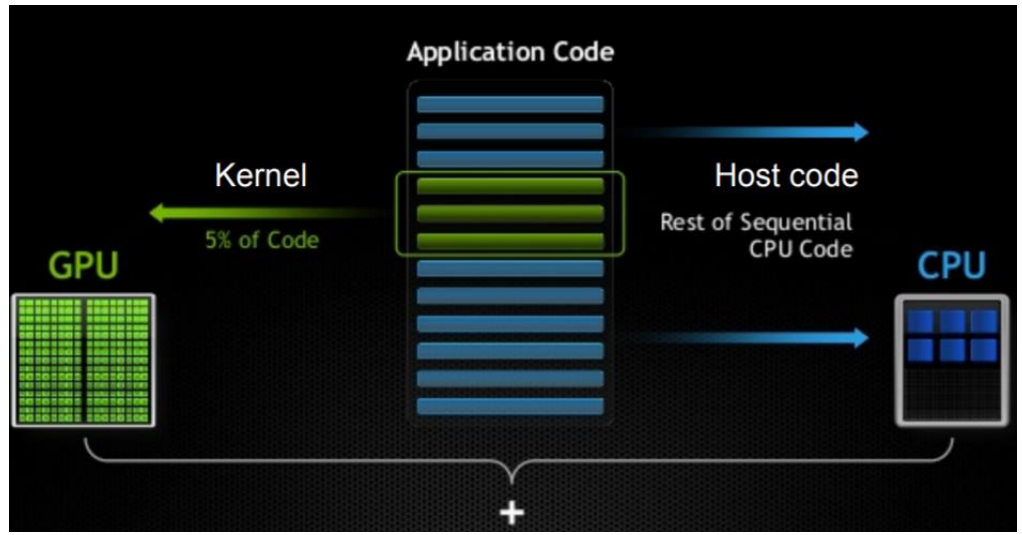


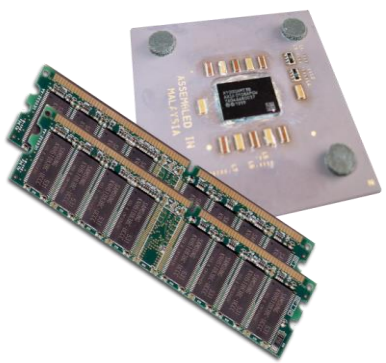

# CUDA并行计算基础

- 异构计算
- CUDA 安装
- CUDA程序的编写
- CUDA程序编译
- 利用NVProf查看程序执行情况

# 异构计算

- 术语:
  - *Host* CPU和内存(host memory)
  - *Device* GPU和显存(device memory)



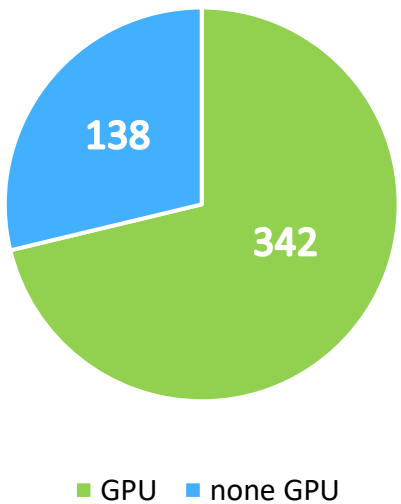



Host

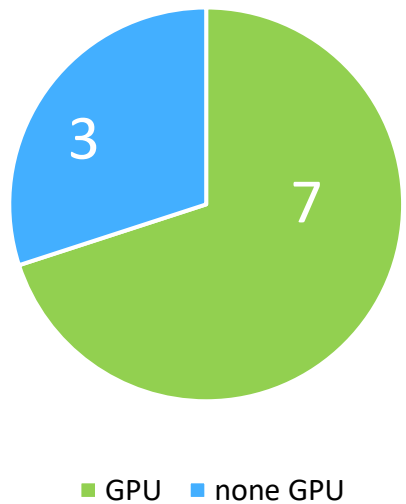

Device

# 异构计算

高性能计算大会ISCTOP 500




高性能计算大会ISC TOP10



# CUDA安装

- 适用设备:
  - *所有包含NVIDIA GPU的服务器，工作站，个人电脑，嵌入式设备等电子设备*
- 软件安装:
  - Windows：https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html
    只需安装一个.exe的可执行程序
  - Linux：https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html
    按照上面的教程，需要6 / 7 个步骤即可
  - Jetson： https://developer.nvidia.com/embedded/jetpack
    直接利用NVIDIA SDK Manager 或者 SD image进行刷机即可

# CUDA安装

- 软件安装:
  - 查看当前设备中GPU状态：
    服务器，工作站，个人电脑：nvidia-smi
    Jetson等设备： Jtop
    其他工具。

  - 查看当前设备参数：
  - 在CUDA sample中1_Utilities/deviceQuery文件夹下的deviceQuery程序。以Ubuntu为例，deviceQuery 程序在：/usr/local/cuda/samples/1_Utilities/deviceQuery

  - https://github.com/NVIDIA/cuda-samples

# CUDA程序的编写

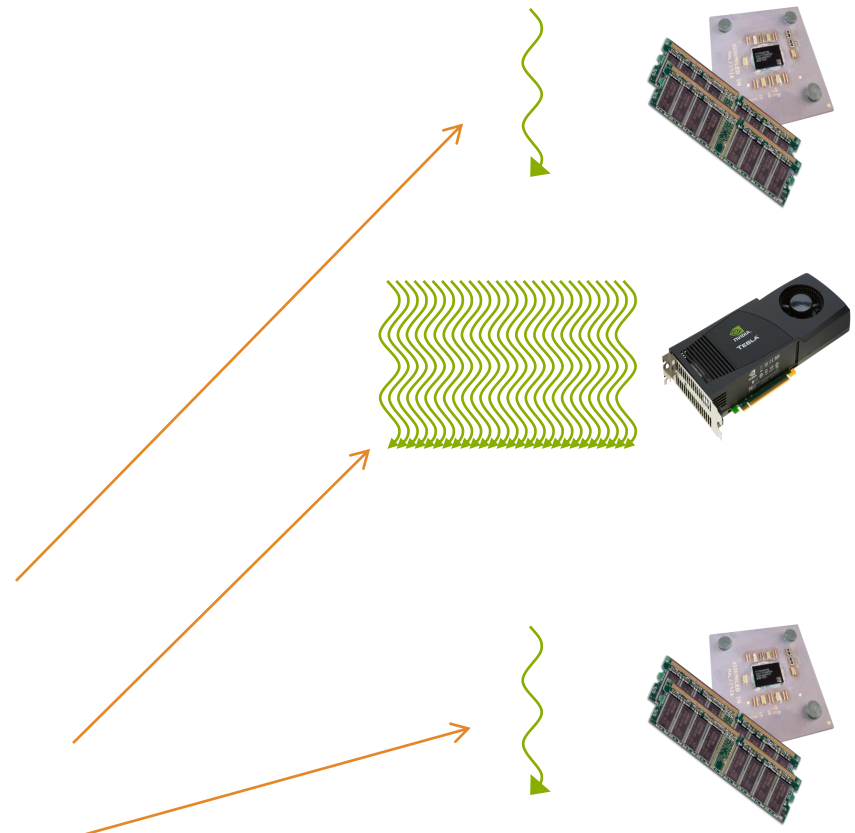
```
#include <iostream>
#include <algorithm>

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;
}
```

并行代码

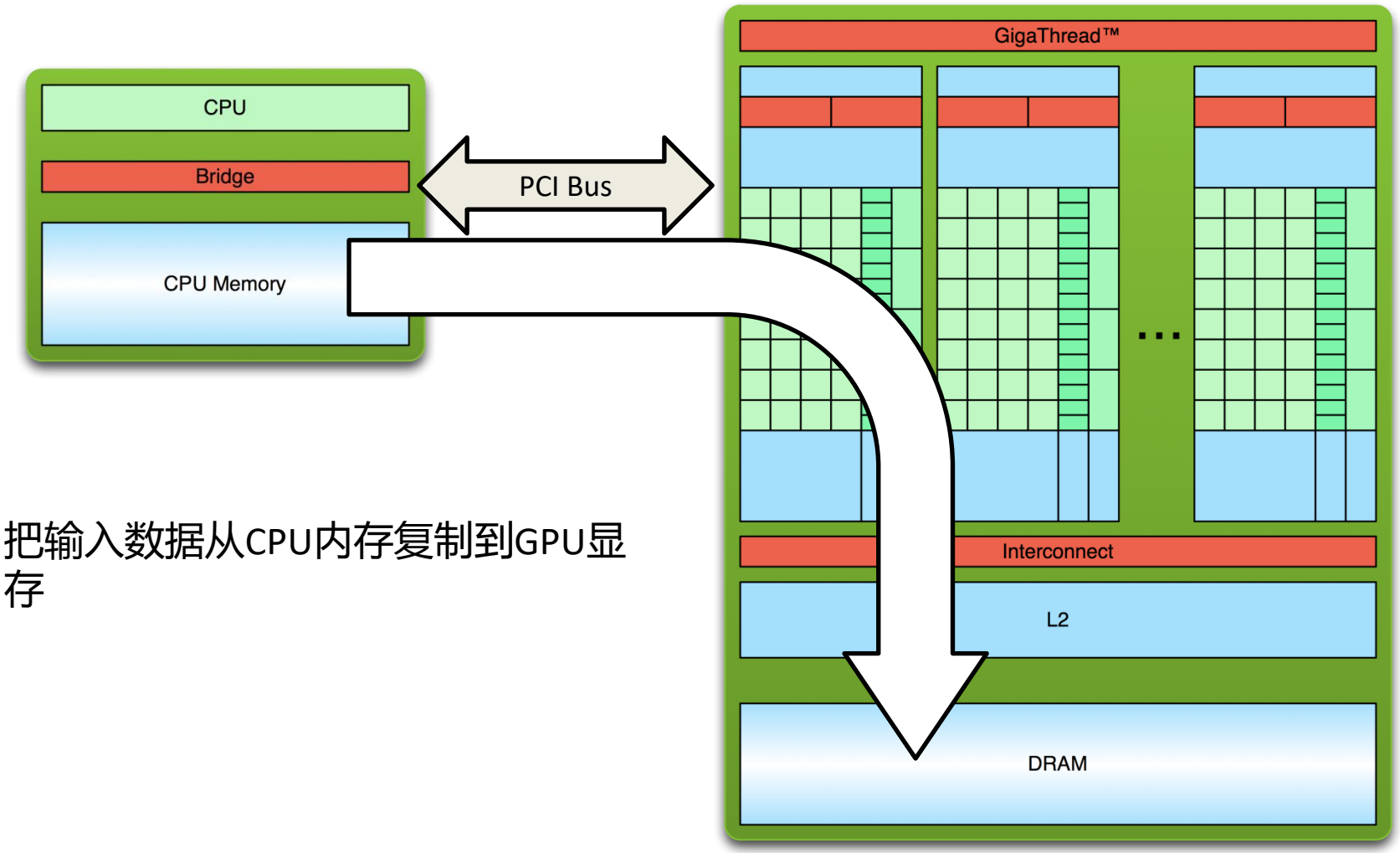
串行代码

并行代码

串行代码

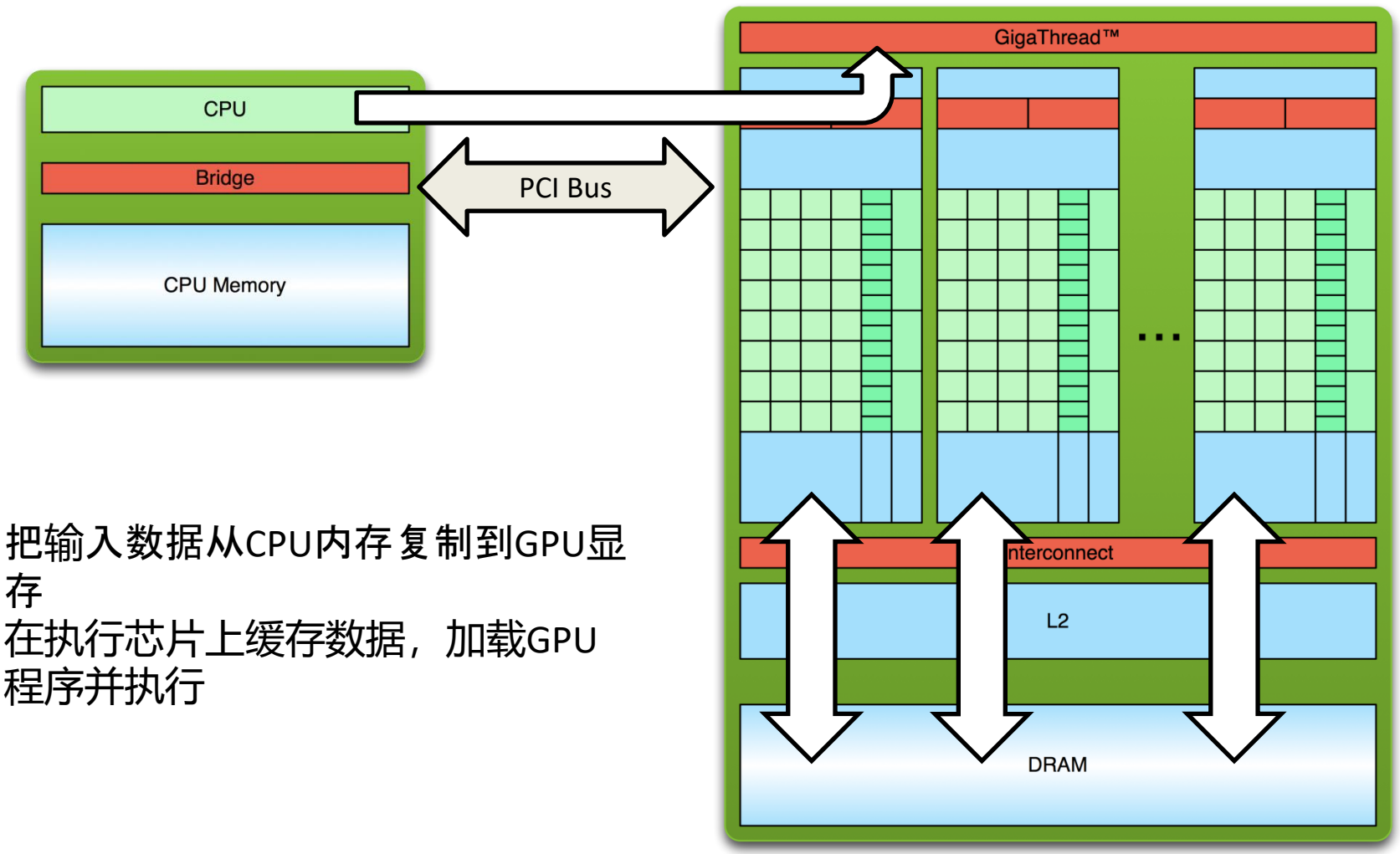


# CUDA程序的编写



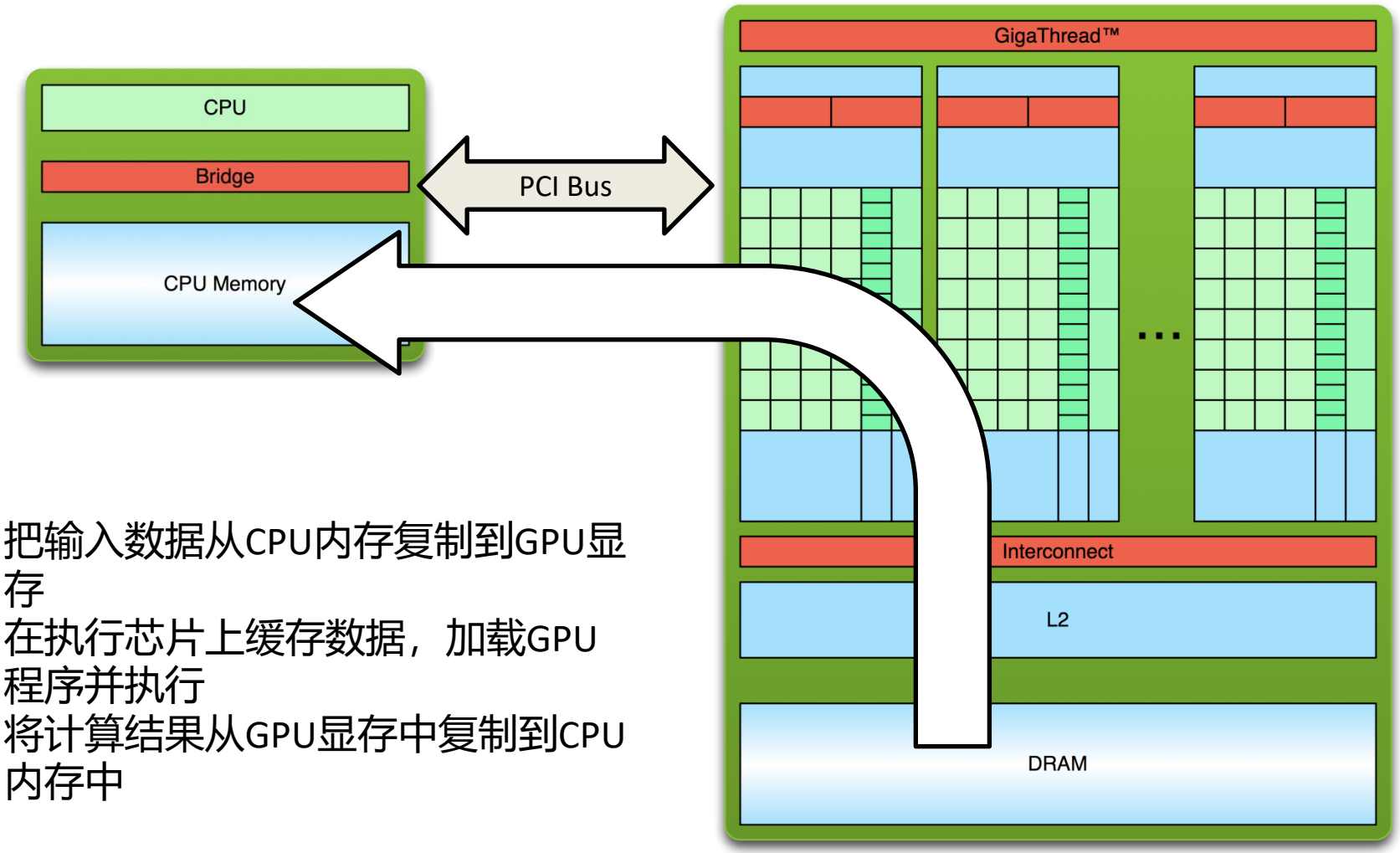

1. 把输入数据从CPU内存复制到GPU显存

# CUDA程序的编写




1. 把输入数据从CPU内存复制到GPU显存
2. 在执行芯片上缓存数据，加载GPU程序并执行

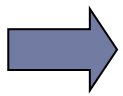
# CUDA程序的编写



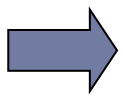

1. 把输入数据从CPU内存复制到GPU显存
2. 在执行芯片上缓存数据，加载GPU程序并执行
3. 将计算结果从GPU显存中复制到CPU内存中

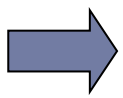
# CUDA编程模式： Extended C

- **Declspecs**
  - **global, device, shared, local, constant**
- **关键词**
  - **threadIdx, blockIdx**
- **Intrinsics**
  - **__syncthreads**
- **运行期API**
  - **Memory, symbol, execution management**
- **函数调用**

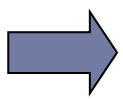


```
__device__ float filter[N];

__global__ void convolve (float *image)  {

  __shared__ float region[M];
  ...

  region[threadIdx] = image[i];

  __syncthreads()
  ...

  image[j] = result;
}
```





```
// Allocate GPU memory
void *myimage = cudaMalloc(bytes)
```



```
// 100 blocks, 10 threads per block
convolve<<<100, 10>>> (myimage);
```

# CUDA程序的编写

__global__

__global__执行空间说明符将函数声明为内核。 它的功能是：

- 在设备上执行，
- 可从主机调用，可在计算能力为 3.2 或更高的设备调用。
- __global__ 函数必须具有 void 返回类型，并且不能是类的成员。
- 对 __global__ 函数的任何调用都必须指定其执行配置。
- 对 __global__ 函数的调用是异步的，这意味着它在设备完成执行之前返回。

__device__

__device__ 执行空间说明符声明了一个函数：

- 在设备上执行，
- 只能从设备调用。
- __global__ 和 __device__ 执行空间说明符不能一起使用。

__host__

__host__ 执行空间说明符声明了一个函数：

- 在主机上执行，
- 只能从主机调用。
- __global__ 和 __host__ 执行空间说明符不能一起使用。但是， __device__ 和 __host__ 执行空间说明符可以一起使用，在这种情况下，该函数是为主机和设备编译的。

# CUDA程序的编写

| | 执行<br>位置 | 调用<br>位置 |
|---|---|---|
| __device__ float DeviceFunc() | device | device |
| __global__ void KernelFunc() | device | host & device（arch>3.2） |
| __host__ float HostFunc() | host | host |

- __global__ 定义一个 kernel 函数
  - 入口函数，CPU上调用，GPU上执行
  - 必须返回void
- __device__ and __host__ 可以同时使用

# CUDA程序的编写

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

核函数( Kernel function)

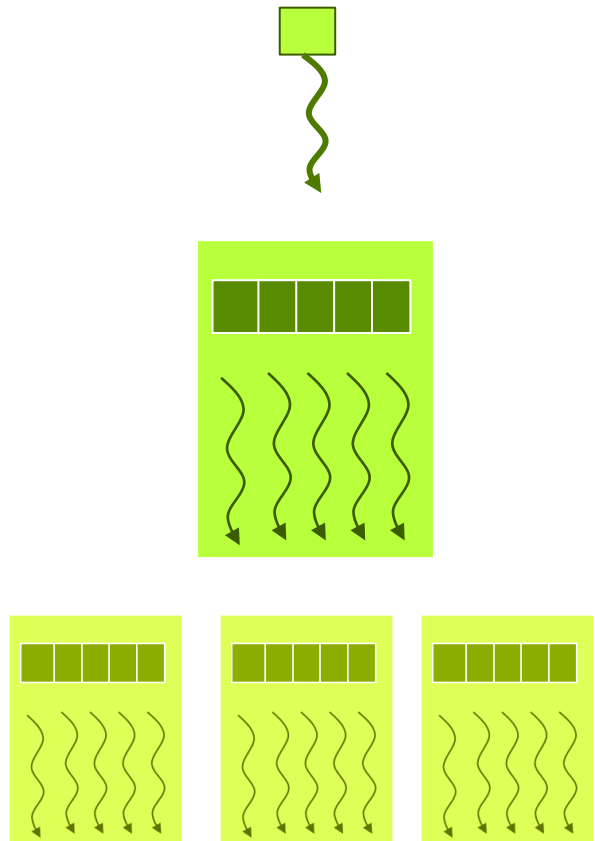
调用核函数( Kernel function)

执行设置(execution configuration)
定义调用了多少个线程

# CUDA线程层次

**HelloFromGPU** <<<grid_size, block_size>>>();

- Thread: sequential execution unit
  - 所有线程执行相同的核函数
  - 并行执行
- Thread Block: a group of threads
  - 执行在一个Streaming Multiprocessor (SM)
  - 同一个Block中的线程可以协作
- Thread Grid: a collection of thread blocks
  - 一个Grid当中的Block可以在多个SM中执行



# 线程层次

- 执行设置:

dim3 grid(3,2,1), block(5,3,1)

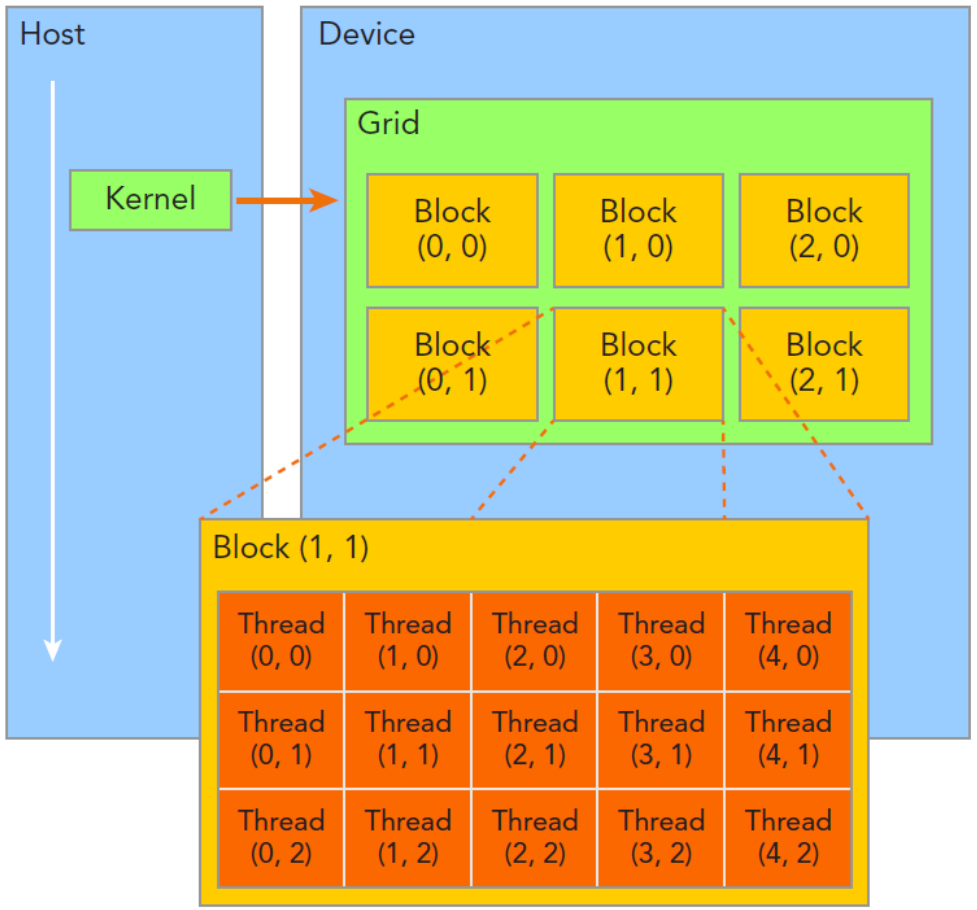
- Built-in variables:
  - threadIdx.[x y z]

    是执行当前kernel函数的线程在block中的索引值
  - blockIdx.[x y z]

    是指执行当前kernel函数的线程所在block，在grid中的索引值
  - blockDim.[x y z]

    表示一个block中包含多少个线程
  - gridDim.[x y z]

    表示一个grid中包含多少个block



# 线程层次

- 我们写的程序:

```
__global__ void add( int *a, int *b, int *c ) {
    c[threadIdx.x] = a[threadIdx.x] + b[threadIdx.x];
}
add<<<1,4>>>( a, b, c);
```

- 实际上在设备上运行的样子:

Thread 0

**c[0] = a[0] + b[0];**

Thread 1

**c[1] = a[1] + b[1];**

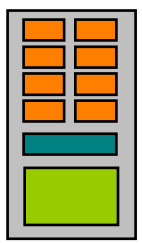
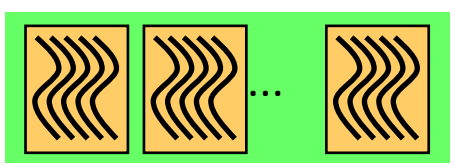
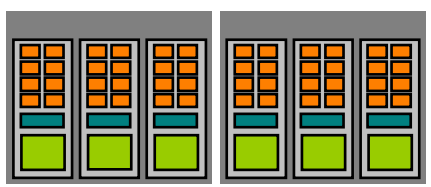
Thread 2

**c[2] = a[2] + b[2];**

Thread 3

**c[3] = a[3] + b[3];**

# 线程层次

| Software | GPU | |
|---|---|---|
|  Thread |  CUDA Core | **Threads are executed by cuda core** |
|  Block |  SM | **Thread blocks are executed on SM** |
|  Grid |  Device | **A kernel is launched as a grid of thread blocks** |

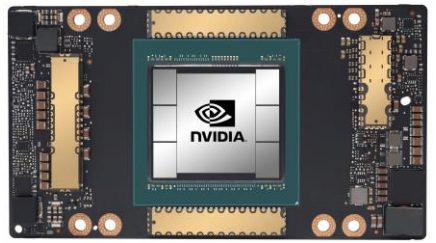
# 线程层次

# WHY BOTH BLOCK AND THREAD?

## 为什么不是只有线程？这样我们不就可以使用的更方便了吗？

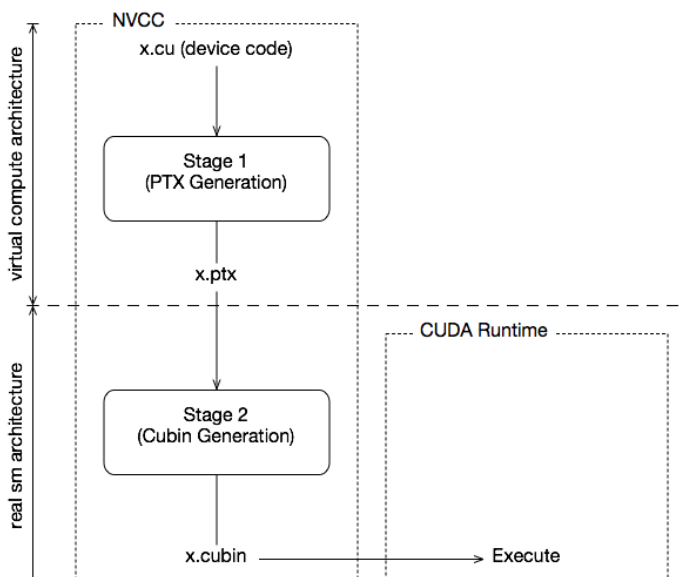
Blocks 好像也不是必须的

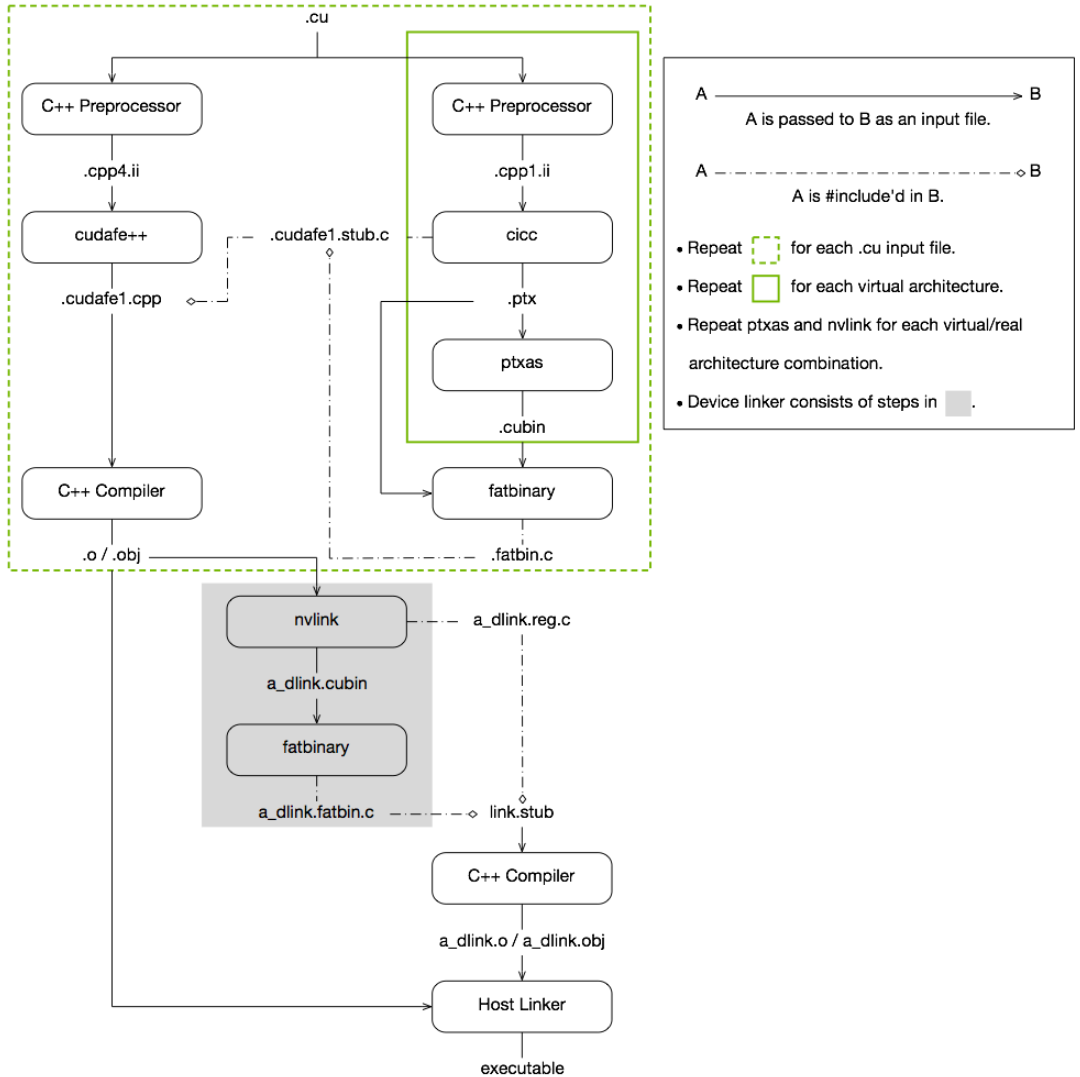
- 增加了一个层级的抽象，也增加了复杂度
- 使用Blocks或者Grid我们收获了什么?




**This is related to the GPU Architecture!**

# CUDA程序的编译




https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html

# CUDA程序的编译

```
nvcc x.cu --gpu-architecture=compute_50 --gpu-code=sm_50,sm_52

nvcc x.cu --gpu-architecture=compute_50

nvcc x.cu \
   --generate-code arch=compute_50,code=sm_50 \
   --generate-code arch=compute_50,code=sm_52 \
   --generate-code arch=compute_53,code=sm_53

nvcc x.cu \
   --generate-code
arch=compute_50,code=[sm_50,sm_52] \
   --generate-code arch=compute_53,code=sm_53
```

https://developer.nvidia.com/cuda-gpus#compute

| | |
|---|---|
| `sm_35` | Basic features<br>+ Kepler support<br>+ Unified memory programming<br>+ Dynamic parallelism support |
| `sm_50`, and `sm_53` | + Maxwell support |
| `sm_60`, `sm_61`, and `sm_62` | + Pascal support |
| `sm_70` and `sm_72` | + Volta support |
| `sm_75` | + Turing support |
| `sm_80` and `sm_86` | + Ampere support |

| | |
|---|---|
| `compute_35`, and `compute_37` | Kepler support<br>Unified memory programming<br>Dynamic parallelism support |
| `compute_50`, `compute_52`, and `compute_53` | + Maxwell support |
| `compute_60`, `compute_61`, and `compute_62` | + Pascal support |
| `compute_70` and `compute_72` | + Volta support |
| `compute_75` | + Turing support |
| `compute_80` and `compute_86` | + Ampere support |

# CUDA程序的编译

sm_35
Basic features
+ Kepler support
+ Unified memory programming
+ Dynamic parallelism support

sm_50, sm_52 and sm_53 + Maxwell support

sm_60, sm_61, and sm_62 + Pascal support

sm_70 and sm_72 + Volta support

sm_75 + Turing support

```
sm_35, sm_37, sm_50, sm_52, sm_53, sm_60, sm_61, sm_62, sm_70, sm_72, sm_73, sm_75, sm_80, sm_86
```

# CUDA程序的编译

compute_35, and compute_37
Kepler support
Unified memory programming
Dynamic parallelism support

compute_50, compute_52, and compute_53 + Maxwell support

compute_60, compute_61, and compute_62 + Pascal support

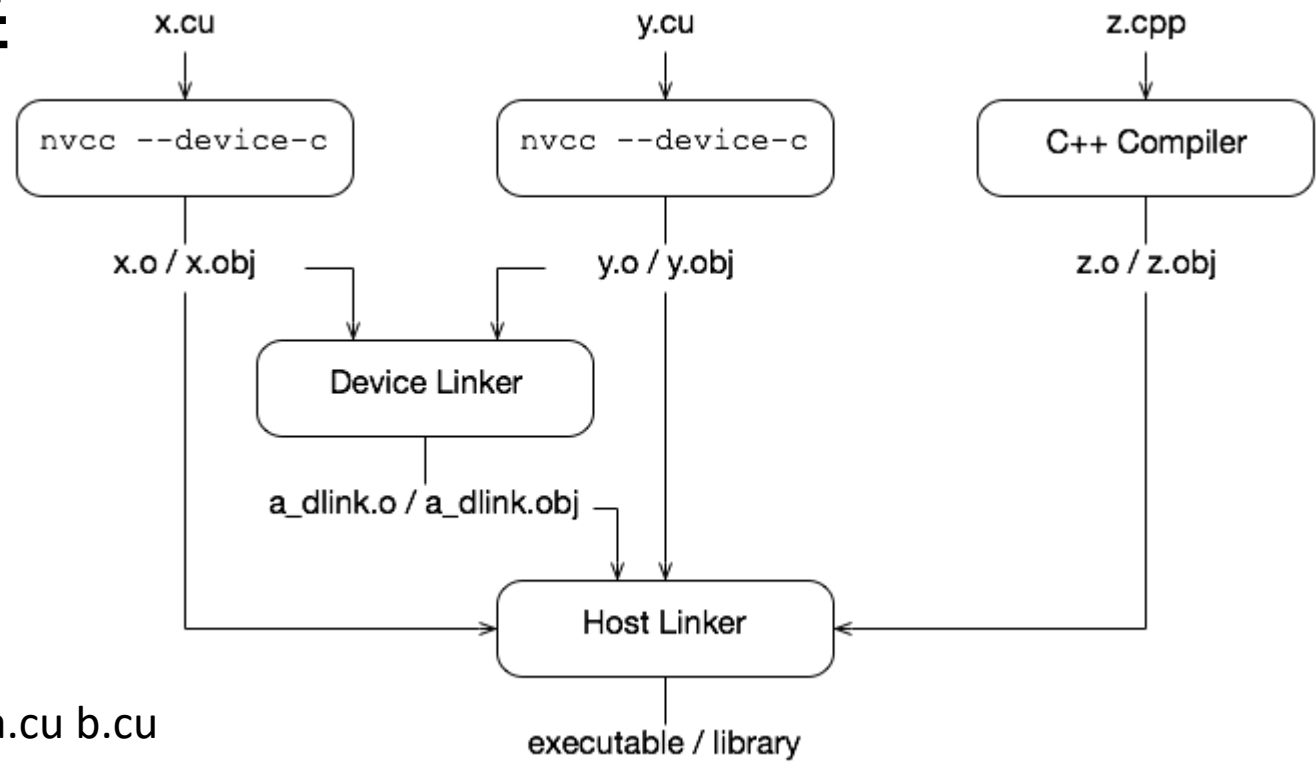
compute_70 and compute_72 + Volta support

compute_75 + Turing support

compute_80 + Ampere support

```
compute_35, compute_37, compute_50, compute_52, compute_53, compute_60, compute_61, compute_62, compute_70, compute_72, compute_73, compute_75, compute_80, compute_86
```

# CUDA程序的编译




nvcc --gpu-architecture=sm_50 --device-c a.cu b.cu
nvcc --gpu-architecture=sm_50 a.o b.o –o a.exe

# CUDA程序的编译

hello_from_gpu.cuh 文件

```
#include <stdio.h>

__global__ void hello_from_gpu();
```

hello_from_gpu.cu 文件

```
#include <stdio.h>
#include "hello_from_gpu.cuh"

__global__ void hello_from_gpu()
{
    printf("Hello World from the GPU!\n");
}
```

hello_from_gpu_main.cu 文件

```
#include <stdio.h>
#include "hello_from_gpu.cuh"

int main(void)
{
    hello_from_gpu<<<1, 1>>>();
    cudaDeviceSynchronize();
    return 0;
}
```

nvcc --device-c hello_from_gpu.cu -o hello_from_gpu.o
nvcc hello_from_gpu.o hello_cuda_main.cu -o hello_from_gpu

# NVPROF

Kernel Timeline 输出的是以gpu kernel 为单位的一段时间的运行时间线，我们可以通过它观察GPU在什么时候有闲置或者利用不够充分的行为，更准确地定位优化问题。nvprof是nvidia提供的用于生成gpu timeline的工具，其为cuda toolkit的自带工具。

非常方便的分析工具!

```
nvprof -o out.nvvp  a.exe
```

可以结合nvvp或者nsight进行可视化分析

https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview

# NVPROF

nvprof a.exe

```
==2189== NVPROF is profiling process 2189, command: ./test.exe
==2189== Warning: Unified Memory Profiling is not supported on the underlying platform. System requirements for unified memory can be found at: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#um-requirements
Hello World from the GPU!
==2189== Profiling application: ./test.exe
==2189== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  771.18us         1  771.18us  771.18us  771.18us  hello_from_gpu(void)
      API calls:   99.71%  348.59ms         1  348.59ms  348.59ms  348.59ms  cudaLaunchKernel
                    0.24%  854.14us         1  854.14us  854.14us  854.14us  cudaDeviceSynchronize
                    0.03%  119.17us        97  1.2280us     625ns  30.835us  cuDeviceGetAttribute
                    0.00%  12.709us         1  12.709us  12.709us  12.709us  cuDeviceTotalMem
                    0.00%  7.6550us         3  2.5510us  1.4580us  3.3850us  cuDeviceGetCount
                    0.00%  3.9070us         2  1.9530us  1.3550us  2.5520us  cuDeviceGet
                    0.00%  1.9270us         1  1.9270us  1.9270us  1.9270us  cuDeviceGetName
                    0.00%     990ns         1     990ns     990ns     990ns  cuDeviceGetUuid
```

# NVPROF

nvprof --print-gpu-trace a.exe

```
==2382== NVPROF is profiling process 2382, command: ./vectorAdd.exe
==2382== Warning: Unified Memory Profiling is not supported on the underlying platform. System requirements for unified memory can be found at: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#um-requirements
Pass
==2382== Profiling application: ./vectorAdd.exe
==2382== Profiling result:
   Start  Duration            Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Throughput  SrcMemType  DstMemType           Device
Context    Stream  Name
7.21002s  2.71933s                    -               -         -         -         -  762.94MB  280.56MB/s    Pageable      Device  NVIDIA Tegra X1
1         7  [CUDA memcpy HtoD]
10.0718s  3.47225s                    -               -         -         -         -  762.94MB  219.72MB/s    Pageable      Device  NVIDIA Tegra X1
1         7  [CUDA memcpy HtoD]
13.7002s  145.77ms         (781250 1 1)       (128 1 1)        10        0B        0B         -           -           -           -  NVIDIA Tegra X1
1         7  add(double const *, double const *, double*) [111]
13.8460s  3.00630s                    -               -         -         -         -  762.94MB  253.78MB/s      Device    Pageable  NVIDIA Tegra X1
1         7  [CUDA memcpy DtoH]

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and can be more than
what the compiler shows.
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
SrcMemType: The type of source memory accessed by memory operation/copy
DstMemType: The type of destination memory accessed by memory operation/copy
```

# NVPROF

## nvprof --print-api-trace a.exe

```
==2687== NVPROF is profiling process 2687, command: ./vectorAdd.exe
==2687== Warning: Unified Memory Profiling is not supported on the underlying platform. System
p://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#um-requirements
Pass
==2687== Profiling application: ./vectorAdd.exe
==2687== Profiling result:
   Start  Duration  Name
157.92ms  2.9170us  cuDeviceGetCount
157.92ms  1.5100us  cuDeviceGetCount
158.00ms  2.1350us  cuDeviceGet
158.00ms  1.7710us  cuDeviceGetAttribute
158.02ms  1.5620us  cuDeviceGetAttribute
158.02ms  1.3020us  cuDeviceGetAttribute
158.15ms  2.4480us  cuDeviceGetCount
158.16ms  1.2500us  cuDeviceGet
158.16ms  2.3960us  cuDeviceGetName
158.16ms  9.7920us  cuDeviceTotalMem
158.17ms  1.3540us  cuDeviceGetAttribute
158.18ms  1.0930us  cuDeviceGetAttribute
```

```
158.35ms     729ns  cuDeviceGetAttribute
158.35ms     782ns  cuDeviceGetAttribute
158.35ms     729ns  cuDeviceGetAttribute
158.36ms     989ns  cuDeviceGetUuid
158.38ms  990.63ms  cudaMalloc
1.14902s  1.69008s  cudaMalloc
2.83920s  2.72178s  cudaMalloc
5.56107s  2.04745s  cudaMemcpy
7.60853s  3.82079s  cudaMemcpy
11.4296s  34.226ms  cudaLaunchKernel (add(double const *, double const *, double*) [111])
11.4639s  3.58150s  cudaMemcpy
16.5855s  62.523ms  cudaFree
16.6480s  75.109ms  cudaFree
16.7231s  63.557ms  cudaFree
```

**CUresult cuDeviceGetAttribute ( int * *pi*, CUdevice_attribute *attrib*, CUdevice *dev* )**

Returns in `*pi` the integer value of the attribute `attrib` on device `dev`. The supported attributes are:

- **CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_BLOCK**: Maximum number of threads per block;
- **CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_X**: Maximum x-dimension of a block;
- **CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Y**: Maximum y-dimension of a block;
- **CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Z**: Maximum z-dimension of a block;
- **CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_X**: Maximum x-dimension of a grid;
- **CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Y**: Maximum y-dimension of a grid;
- **CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Z**: Maximum z-dimension of a grid;
- **CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_BLOCK**: Maximum amount of shared memory available to a thread b
- **CU_DEVICE_ATTRIBUTE_TOTAL_CONSTANT_MEMORY**: Memory available on device for __constant__ variables in a CUDA C
- **CU_DEVICE_ATTRIBUTE_WARP_SIZE**: Warp size in threads;
- **CU_DEVICE_ATTRIBUTE_MAX_PITCH**: Maximum pitch in bytes allowed by the memory copy functions that involve memory regio
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_WIDTH**: Maximum 1D texture width;
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_WIDTH**: Maximum 2D texture width;
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_HEIGHT**: Maximum 2D texture height;
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_WIDTH**: Maximum 3D texture width;
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_HEIGHT**: Maximum 3D texture height;
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_DEPTH**: Maximum 3D texture depth;
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_LAYERED_WIDTH**: Maximum 1D layered texture width;
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_LAYERED_LAYERS**: Maximum layers in a 1D layered texture;
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_WIDTH**: Maximum 2D layered texture width;
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_HEIGHT**: Maximum 2D layered texture height;
- **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_LAYERS**: Maximum layers in a 2D layered texture;
- **CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_BLOCK**: Maximum number of 32-bit registers available to a thread block; this
- **CU_DEVICE_ATTRIBUTE_CLOCK_RATE**: Peak clock frequency in kilohertz;
- **CU_DEVICE_ATTRIBUTE_TEXTURE_ALIGNMENT**: Alignment requirement; texture base addresses aligned to textureAlign bytes
- **CU_DEVICE_ATTRIBUTE_GPU_OVERLAP**: 1 if the device can concurrently copy memory between host and device while executi
- **CU_DEVICE_ATTRIBUTE_MULTIPROCESSOR_COUNT**: Number of multiprocessors on the device;
- **CU_DEVICE_ATTRIBUTE_KERNEL_EXEC_TIMEOUT**: 1 if there is a run time limit for kernels executed on the device, or 0 if not;
- **CU_DEVICE_ATTRIBUTE_INTEGRATED**: 1 if the device is integrated with the memory subsystem, or 0 if not;
- **CU_DEVICE_ATTRIBUTE_CAN_MAP_HOST_MEMORY**: 1 if the device can map host memory into the CUDA address space, or
- **CU_DEVICE_ATTRIBUTE_COMPUTE_MODE**: Compute mode that device is currently in. Available modes are as follows:
  - **CU_COMPUTEMODE_DEFAULT**: Default mode - Device is not restricted and can have multiple CUDA contexts present at a
  - **CU_COMPUTEMODE_EXCLUSIVE**: Compute-exclusive mode - Device can have only one CUDA context present on it at a t
  - **CU_COMPUTEMODE_PROHIBITED**: Compute-prohibited mode - Device is prohibited from creating new CUDA contexts.
  - **CU_COMPUTEMODE_EXCLUSIVE_PROCESS**: Compute-exclusive-process mode - Device can have only one context used
- **CU_DEVICE_ATTRIBUTE_CONCURRENT_KERNELS**: 1 if the device supports executing multiple kernels within the same context should not be relied upon for correctness;
- **CU_DEVICE_ATTRIBUTE_ECC_ENABLED**: 1 if error correction is enabled on the device, 0 if error correction is disabled or not su
- **CU_DEVICE_ATTRIBUTE_PCI_BUS_ID**: PCI bus identifier of the device;
- **CU_DEVICE_ATTRIBUTE_PCI_DEVICE_ID**: PCI device (also known as slot) identifier of the device;
- **CU_DEVICE_ATTRIBUTE_TCC_DRIVER**: 1 if the device is using a TCC driver. TCC is only available on Tesla hardware running W
- **CU_DEVICE_ATTRIBUTE_MEMORY_CLOCK_RATE**: Peak memory clock frequency in kilohertz;
- **CU_DEVICE_ATTRIBUTE_GLOBAL_MEMORY_BUS_WIDTH**: Global memory bus width in bits;
- **CU_DEVICE_ATTRIBUTE_L2_CACHE_SIZE**: Size of L2 cache in bytes. 0 if the device doesn't have L2 cache;
- **CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_MULTIPROCESSOR**: Maximum resident threads per multiprocessor;
- **CU_DEVICE_ATTRIBUTE_UNIFIED_ADDRESSING**: 1 if the device shares a unified address space with the host, or 0 if not;

# 更多资源：

何琨-Ken

北京 密云

扫一扫上面的二维码图案，加我微信

NVIDIA


## 加入 NVIDIA 开发者计划

获取最新版本软件、工具及开发信息

扫码了解详情


NVIDIA