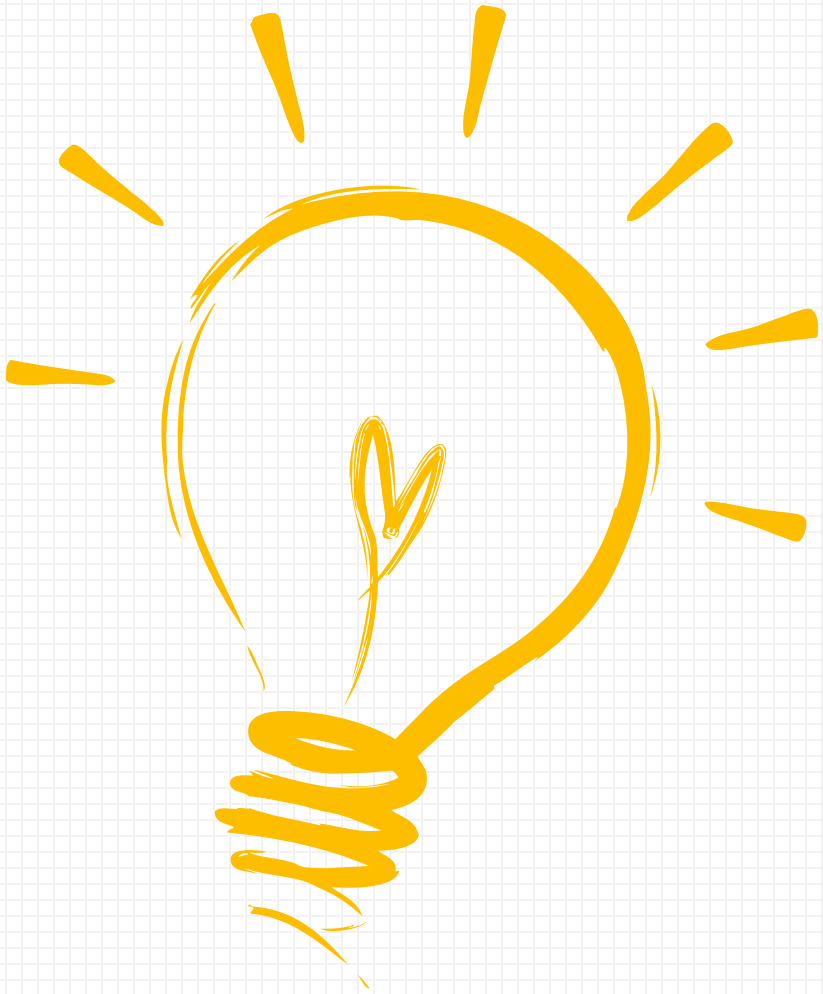
CUDA矩阵加法运算程序

CUDA并行编程系列课程

主讲：权双

CONTENTS



CUDA程序基本框架

```
#include <头文件>

__global__ void 函数名(参数…)
{
    核函数内容
}
int main(void)
{
    设置GPU设备;
    分配主机和设备内存;
    初始化主机中的数据;
    数据从主机复制到设备;
    调用核函数在设备中进行计算;
    将计算得到的数据从设备传给主机;
    释放主机与设备内存;
}
```

```
int main(void)
{
    // 1、设置GPU设备
    setGPU();

    // 2、分配主机内存和设备内存，并初始化
    int iElemCount = 512;                       // 设置元素数量
    size_t stBytesCount = iElemCount * sizeof(float); // 字节数

    // （1）分配主机内存，并初始化
    float *fpHost_A, *fpHost_B, *fpHost_C;
    fpHost_A = (float *)malloc(stBytesCount);
    fpHost_B = (float *)malloc(stBytesCount);
    fpHost_C = (float *)malloc(stBytesCount);
    if (fpHost_A != NULL && fpHost_B != NULL && fpHost_C != NULL)
    {
        memset(fpHost_A, 0, stBytesCount);  // 主机内存初始化为0
        memset(fpHost_B, 0, stBytesCount);
        memset(fpHost_C, 0, stBytesCount);

    }
```

设置GPU设备

★ 1、获取GPU设备数量

代码：

```
int iDeviceCount = 0;
cudaGetDeviceCount(&iDeviceCount);
```

★ 2、设置GPU执行时使用的设备

代码：

```
int iDev = 0;
cudaSetDevice(iDev)
;
```

__host____device__cudaError_t cudaGetDeviceCount (int *count)

Returns the number of compute-capable devices.

Parameters

count

- Returns the number of devices with compute capability greater or equal to 2.0

Returns

cudaSuccess

Description

Returns in `*count` the number of devices with compute capability greater or equal to 2.0 that are available for execution.

__host__cudaError_t cudaSetDevice (int device)

Set device to be used for GPU executions.

Parameters

device

- Device on which the active host thread should execute the device code.

Returns

cudaSuccess, cudaErrorInvalidDevice, cudaErrorDeviceUnavailable,

内存管理

★ CUDA 通过内存分配、数据传递、内存初始化、内存释放进行内存管理

★ 标准C语言内存管理函数----------CUDA内存管理函数

STANDARD C FUNCTIONS	CUDA C FUNCTIONS
malloc	cudaMalloc
memcpy	cudaMemcpy
memset	cudaMemset
free	cudaFree

内存分配

★ 主机分配内存：extern void *malloc(unsigned int num_bytes);

代码：
```
float *fpHost_A;
fpHost_A = (float *)malloc(nBytes);
```

★ 设备分配内存：

代码：
```
float *fpDevice_A;
cudaMalloc((float**)&fpDevice_A, nBytes);
```

__host____device__cudaError_t cudaMalloc (void **devPtr, size_t size)

Allocate memory on the device.

Parameters

devPtr
- Pointer to allocated device memory

size
- Requested allocation size in bytes

Returns

cudaSuccess, cudaErrorInvalidValue, cudaErrorMemoryAllocation

数据拷贝

★ 主机数据拷贝： void *memcpy(void *dest, const void *src, size_t n);

代码： memcpy((void*)d, (void*)s, nBytes);

★ 设备数据拷贝：

代码： cudaMemcpy(Device_A, Host_A, nBytes, cudaMemcpyHostToHost)

__host__cudaError_t cudaMemcpy (void *dst, const void *src, size_t count, cudaMemcpyKind kind)

Copies data between host and device.

Parameters

dst
- Destination memory address

src
- Source memory address

count
- Size in bytes to copy

kind
- Type of transfer

Returns

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidMemcpyDirection

kind:

cudaMemcpyHostToHost 主机→主机

cudaMemcpyHostToDevice 主机→设备

cudaMemcpyDeviceToHost 设备→主机

cudaMemcpyDeviceToDevice 设备→设备

cudaMemcpyDefault 默认

默认方式只允许在支持统一虚拟寻址的系统上使用。

内存初始化

★ 主机内存初始化：void *memset(void *str, int c, size_t n);

代码：

```
memset(fpHost_A, 0, nBytes);
```

★ 设备内存初始化：

代码：

```
cudaMemset(fpDevice_A, 0, nBytes);
```

__host__cudaError_t cudaMemset (void *devPtr, int value, size_t count)

Initializes or sets device memory to a value.

Parameters

devPtr
- Pointer to device memory

value
- Value to set for each byte of specified memory

count
- Size in bytes to set

Returns

cudaSuccess, cudaErrorInvalidValue,

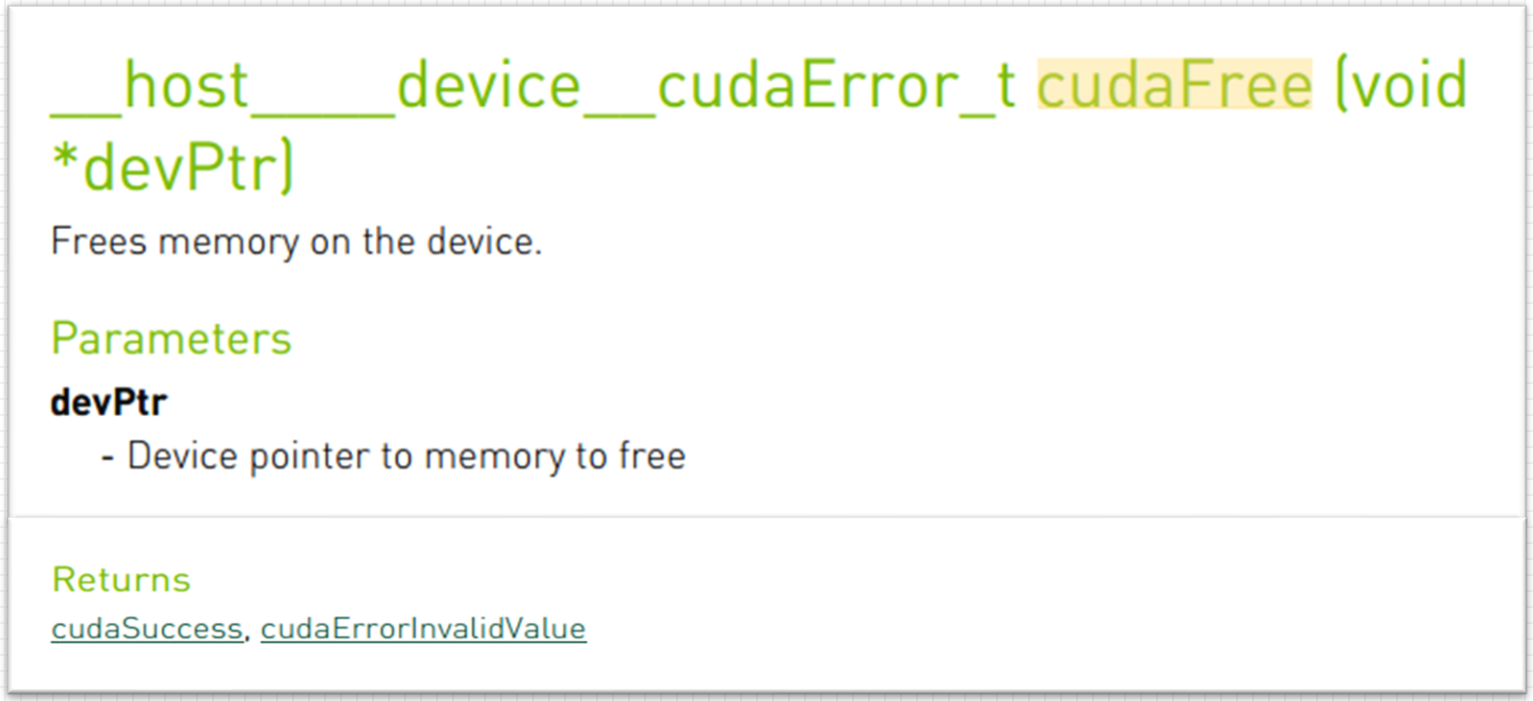
内存释放

★ 释放主机内存：
代码：

```
free(pHost_A);
```

★ 释放设备内存：
代码：

```
cudaFree(pDevice_A);
```



__host____device__cudaError_t cudaFree (void *devPtr)

Frees memory on the device.

Parameters

devPtr

- Device pointer to memory to free

Returns

cudaSuccess, cudaErrorInvalidValue

自定义设备函数

★ 1、设备函数（device function）

（1）定义只能执行在GPU设备上的函数为设备函数

（2）设备函数只能被核函数或其他设备函数调用

（3）设备函数用 __device__ 修饰

★ 2、核函数（kernel function）

（1）用 __global__ 修饰的函数称为核函数，一般由主机调用，在设备中执行

（2） __global__ 修饰符既不能和__host__同时使用，也不可与__device__ 同时使用

★ 3、主机函数（host function）

（1）主机端的普通 C++ 函数可用 __host__ 修饰

（2）对于主机端的函数， __host__修饰符可省略

（3）可以用 __host__ 和 __device__ 同时修饰一个函数减少冗余代码。编译器会针对主机和设备分别编译该函数。

THANKS

谢谢聆听