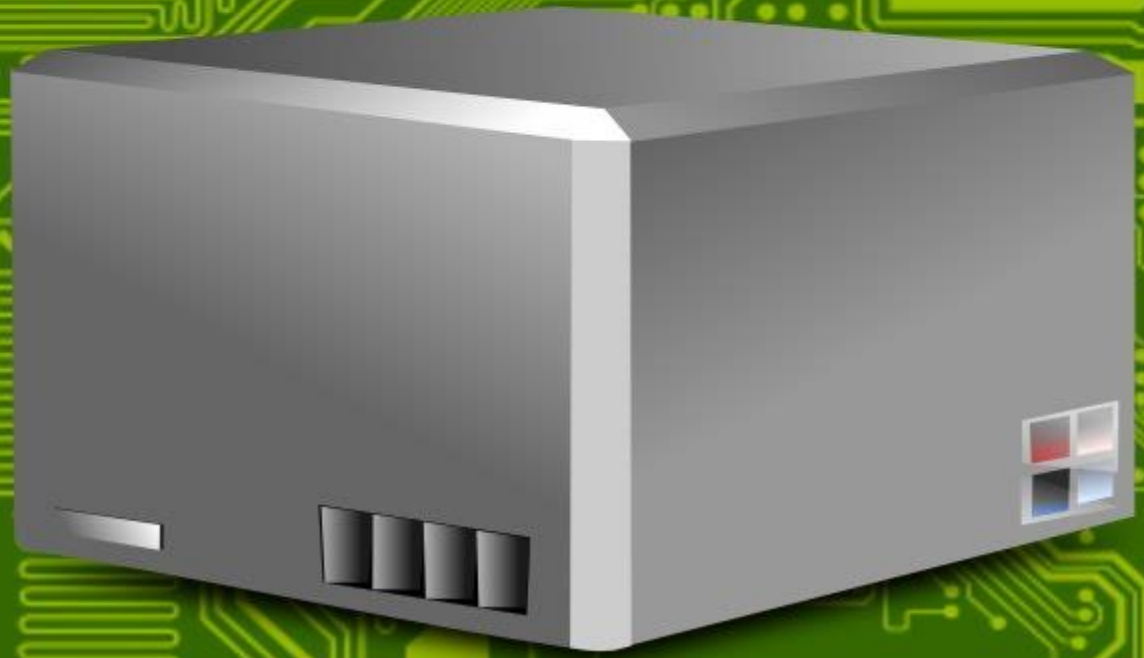
GPU Computing



GPU 内存如何管理

Hui Liu
Email: hui.sc.liu@gmail.com

内存使用

- CUDA 程序会使用GPU内存与CPU内存
- CPU内存的分配与释放是标准的, 例如 new 和 delete, malloc 与 free
- GPU 上内存涉及分配和释放, 使用CUDA提供的库函数实现
- CUDA/GPU内存与CPU内存的互相传输
- 这里主要讲全局内存与共享内存的管理

CPU 内存

- 栈: 有编译器自动分配释放
- 堆: 用户自己分配释放
- C: malloc, calloc, free
- C++: new, delete

GPU 内存

- GPU 内存有很多种类型, 例如全局, 纹理, 常量, 共享
- 每种的管理都不一样
- GPU 内存也有不同的属性, 例如1D, 2D, 3D, 锁页, 统一内存
- CUDA 会增加新的内存类型

GPU 全局内存分配释放

- 内存分配

```
cudaError_t cudaMalloc(void **devPtr, size_t  size);
```

- 内存释放

```
cudaError_t cudaFree(void *devPtr) ;
```

Host 内存分配释放

- Host 内存属于CPU内存, 传输速度比普通CPU内存快很多.
- 内存分配

```
cudaError_t cudaMallocHost(void **devPtr, size_t  size);
```

- 内存释放

```
cudaError_t cudaFreeHost(void *devPtr) ;
```

统一(Unified)内存分配释放

- Unified 内存可以同时被CPU与GPU访问.

- 内存分配

```
cudaError_t cudaMallocManaged(void **devPtr, size_t size,
                              unsigned int  flags = cudaMemAttachGlobal)
```

flags = cudaMemAttachGlobal: 内存可以被任意处理器访问 (CPU, GPU)
flags = cudaMemAttachGHost: 只可以被CPU访问

- 内存释放

```
cudaError_t cudaFree(void *devPtr) ;
```

CPU与GPU内存同步拷贝

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

kind: cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, cudaMemcpyDeviceToDevice, or cudaMemcpyDefault

CPU与GPU内存异步拷贝

cudaError_t cudaMemcpyAsync(void *dst, const void *src, size_t count,
cudaMemcpyKind kind, cudaStream_t stream = 0)

kind: cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, cudaMemcpyDeviceToDevice, or cudaMemcpyDefault

stream: 如果是非0, 可能与其他stream的操作有重叠

共享内存

- 定义在SM中
  - 访问延迟比全局内存低两个数量级, 访问速度比全局内存快一个数量级
  - 16 (48, 64) KB per SM
    - NVIDIA GPUs 最多可以有 30 SMs
- 共享内存是定义在线程块中
- 线程块中每个线程都可以访问, 但不可以被其他线程块访问
- 用途
  - 在一个线程块中共享数据
  - 用作用户自己管理的高速缓存

共享内存

Size known at compile time

```
__global__ void kernel(…)
{
  …
  __shared__ float sData[256];
  …
}

int main(void)
{
  …
  kernel<<<nBlocks,blockSize>>>(…);
  …
}
```

Size known at kernel launch

```
__global__ void kernel(…)
{
  …
  extern __shared__ float sData[];
  …
}

int main(void)
{
  …
  smBytes = blockSize*sizeof(float);
  kernel<<<nBlocks, blockSize,
     smBytes>>>(…);
  …
}
```

Slide Courtesy of NVIDA: Timo Stich

THANK YOU