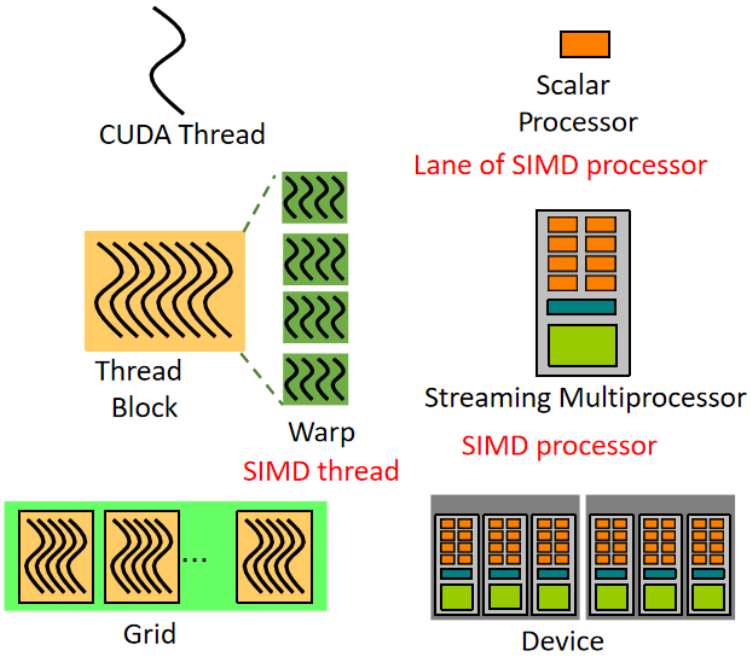
CUDA

- 基本概念
  - 什么是CUDA（**C**ompute **U**nified **D**evice **A**rchitecture，**统一计算架构**）：一种集成技术，通过CUDA可以利用GPU进行并行计算
    - CUDA 架构
      - 为通用计算公开 GPU 并行性
      - 公开/启用性能
    - CUDA C++
      - 基于行业标准 C++
      - 支持异构编程的扩展集
    - 用于管理设备、内存等的简单 API。
      - 本节介绍 CUDA C++
      - 其他可用的语言/绑定：Fortran、 Python、 Matlab 等
  - 简单的处理流程
    - 从CPU内存中拷贝输入数据到GPU内存中
    - 加载 GPU 程序并执行，在芯片上缓存数据以提高性能。
    - 将结果从GPU内存中拷贝回CPU内存
  - 核函数：设备上
    - CUDA C++的关键字 __global__：表示定义一个函数
      - 在设备上执行
      - 在主机代码中被调用（也可以从其他设备代码调用）
    - nvcc 将源代码分离为主机和设备组件
      - 由 NVIDIA 编译器处理的设备函数
      - 由标准主机编译器处理的主机函数
    - 运行一个核函数，内核启动<<<block数， thread 数>>>（）
      - 执行内核的每个线程都被赋予一个唯一的线程 ID (threadIdx)，可通过内置变量在内核中访问该 ID。
  - 并行运行代码
    - add<<<N, 1>>>()：并行运行N次add
- CUDA 共享内存
  - 共享内存相当于用户管理的缓存：应用程序显式分配和访问它。
  - add<<<N, 1>>>：启动N个add副本
  - 在一个block内，线程通过共享内存共享数据
  - 极快的片上存储

    - __shared__声明，为每一个快分配一个变量
        - 驻留在线程块的共享内存空间中
        - 具有块的生命周期
        - 每个块有一个不同的对象
        - 只能被块内的所有线程访问
        - 没有固定地址
    - 典型的编程模式：
        - 将数据从设备内存加载到共享内存
        - 与块的所有其他线程同步，以便每个线程可以安全地读取由不同线程填充的共享内存位置，
        - 处理共享内存中的数据
        - 如有必要，再次同步以确保共享内存已更新结果，
        - 将结果写回设备内存。
- CUDA 基本的优化——执行模型



    - CUDA 线程由标量处理器（SIMD 处理器的lane）执行
    - thread block被拆分为warp并在 SIMT 中的多处理器（SIMD 处理器）上执行。
    - 多个并发thread block可以驻留在一个 SIMD 处理器上（当线程块的数量大于多处理器时） - 受 SIMD 处理器资源（共享内存和寄存器文件）的限制
    - 内核作为线程块网格启动
- warp
    - thread block 由warps构成
    - warp被并行的在多处理器上执行
- GPU 内存操作
    - loads

        - 缓存
            - 默认模式
            - 尝试击中 L1，然后是 L2，然后是 GMEM
            - 加载粒度为 128 字节行
    - store：
        - L1 无效，L2 回写
    - load 操作
        - 每个 warp 发出内存操作（32 个 CUDA 线程，一个 SIMD 线程）
        - warp中的 CUDA 线程提供内存地址
        - 确定需要哪些lines/segments
        - 请求所需的lines/segments
    - GPU内存优化
        - 力求完美结合
            - （对齐起始地址 - 可能需要填充）
            - warp应该在一个连续的区域内访问
        - 有足够的并发访问使总线饱和
            - 每个线程处理多个元素
                - 多个负载流水线化
                - 索引计算通常可以重复使用
            - 启动足够的 warp 以最大化吞吐量
                - 通过切换 warp 隐藏延迟
        - 使用所有缓存！
- 共享内存
    - 用途：
        - 块内的线程间通信
        - 缓存数据以减少冗余的全局内存访问，如 CPU 缓存
        - 使用它来改进全局内存访问模式
        - 共享内存预计是每个处理器内核附近的低延迟内存（很像 L1 缓存）
        - 可以通过编程 API 进行管理（分配和释放）
    - 组织：
        - 32 个banks，4 字节宽banks
        - 连续的 4 字节word属于不同的bank--------轮询分配
    - 性能
        - 通常：每个多处理器每 1 或 2 个时钟每组 4 字节
        - 每 32 个线程（warp）发出共享访问

        - 串行化：如果32的N个线程访问同一个bank中不同的4字节字（bank冲突），N次访问串行执行
    - 避免bank conflict：避免同一个warp中的不同线程访问同一个bank
- unified memory
    - 更简单的编程和内存模型
        - 单一分配，单一指针，可在任何地方访问
        - 无需显式复制 简化代码移植
    - 通过数据局部性保持性能
        - 将数据迁移到访问处理器 保证全局一致性
        - 仍然允许显式手动调整
    - 启用大数据模型
        - 超额订阅 GPU 内存 分配到系统内存大小
    - 更简单的数据访问
        - CPU/GPU 数据一致性
        - 统一内存原子操作
    - Tune——统一内存性能
        - 通过 cudaMemAdvise API 显式预取 API 的使用提示
    - code

```
__global__
void setValue(int *ptr, int index,  int val)
{
  ptr[index] = val;
}


void foo(int size)
  { char *data;
  cudaMallocManaged(&data, size);

  memset(data, 0, size);

  setValue<<<...>>>(data, size/2, 5);
  cudaDeviceSynchronize();

  useData(data);

  cudaFree(data);
}
```

Unified Memory allocation

Access all values on CPU

Access one value on GPU

以上内容整理于 幕布文档