GPU Architcture

Balaji

IISc Bangalore

Building blocks

- Thread- program, line to be executed, data structures, values of variables
- Block- A collection of threads in the **grid**. Flexibility in the number of threads in a block.
- Warp - 32 threads in a block, executed in parallel.
  __syncthreads() to synchronize threads in a block.
  __syncwarp() to synchronize warps.
- x,y,z dimensions of thread, block ids
- SM - streaming multiprocessors, having cuda cores.
- SPMD vs SIMD - Single Program Multiple Data vs Single Instruction Multiple Data

Kernel Functions

- A function that is executed on the GPU
- Keywords: __global__, __device__, __host__, __shared__
- cudaMalloc / cudaFree ,cudaMemcpy to allocate/free memory on the GPU and transfer data between CPU and GPU respectively
- Called as kernelName¡¡¡blockspergrid, threadsperblock¿¿¿(args)
- Compiled by nvcc compiler to generate PTX code which is further compiled into an object file executed on GPU.
- Dynamically allocated arrays cant be multidimensional, have to be linearized.

Transparent Scalability

- Barrier to be synchronized so that all threads reach same execution point.
- Only for threads in a block, so multiple blocks can be parallelly executed.
- Threads waiting for each other to reach the same checkpoint- deadlock
- The ability to execute the same application code on different hardware with different amounts of execution resources is referred to as transparent scalability

Warps And SIMD

- Single instruction executed by a all threads in a warp.
- Cores in SM arranged into processing blocks.
- Each warp is assigned to the same processing block.
- Lesser area for control, more for arithmetic throughput.

Control Divergence

- If threads in a warp take different paths, SIMD mechanism takes multiple passes.
- These multiple passes are executed sequentially or parallely in different gpu architectures.
- Each thread follows its own control flow and in a pass corresponding to an instruction it does not follow, the thread becomes inactive.

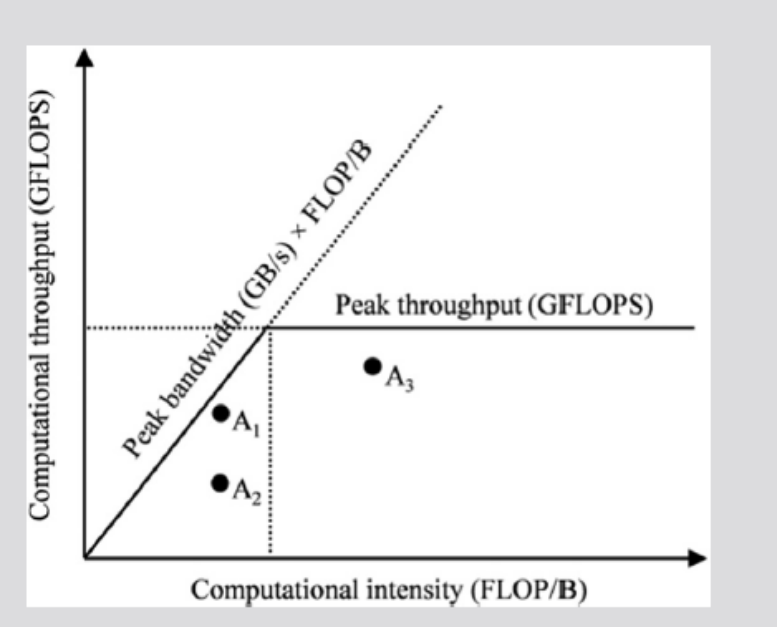
Reducing Latency In GPUs

- Zero-overhead Scheduling in GPUs- Puts long latency operation warps to sleep and finds another warp to execute.
- Subscribes to a lot more threads than executable using available resources(cores,registers) on SM, to find a warp that can be executed while a warp is executing a long latency operation.
- GPUs can dedicate more chip area to floating-point execution and memory access channel resources than cache memory and branch prediction mechanisms.
- The ratio of the number of warps assigned to an SM to the maximum number it supports is referred to as occupancy.

Limitations to occupancy

- Dynamic block partitioning(Variable number of threads) in GPUs to allocates exact amount of resources as per requirements
- Number of threads that can be supported by SM not being multiple of threads per block
- Maximum number of blocks that can be assigned to SM is limited.
- Blocks executing code with lot of automatic variables having register requirements can reduce occupancy. Limit in registers per SM.
- Register spilling to local memory may increase execution time.
- Shared Memory usage by blocks similarly.

Efficiency Barrier



- Compute to global memory access ratio is called computational intensity.
- Memory bandwidth limits throughput of memory-bound programs.
- Higher computational intense programs are compute-bound by the hardware resources, lower ones are memory-bound.

Memory types in GPU

- Global memory - accessible by all threads, slow, large, read and write
- Local Memory - portion of the global memory reserved by a thread, eg: statically allocated arrays, spilled registers, and other elements of the thread's call stack.
- Shared memory - accessible by threads in a block, fast, small
- Constant memory - read only, short-latency, high-bandwidth, read-only-access for the device, writable by host
- Register memory - fast, small, per thread
- Texture memory - Not discussed

Locations And Access

Device code can:
- R/W per-thread **registers**
- R/W per-thread **local memory**
- R/W per-block **shared memory**
- R/W per-grid **global memory**
- Read only per-grid **constant memory**

Host code can
- Transfer data to/from per grid **global** and **constant memories**

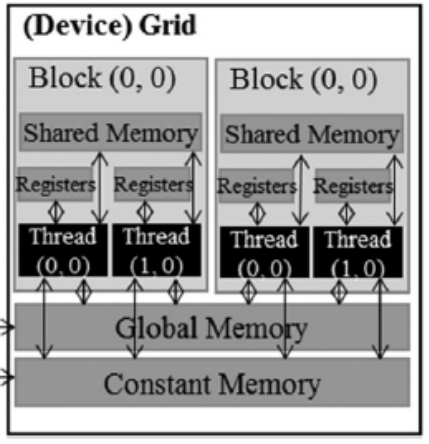


Table 5.1 CUDA variable declaration type qualifiers and the properties of each type.

Variable declaration	Memory	Scope	Lifetime
Automatic variables other than arrays	Register	Thread	Grid
Automatic array variables	Local	Thread	Grid
__device__ __shared__ int SharedVar;	Shared	Block	Grid
__device__ int GlobalVar;	Global	Grid	Application
__device__ __constant__ int ConstVar;	Constant	Grid	Application

Shared memory block optimisations

- Usage of locality(on-chip constant caching), tiling algorithms to utilize shared memory reduce latency.
- Strip Mining - breaking a large problem into smaller ones that fit into shared memory.
- Can compute appropriate shared memory per block using cuda. Declaring an extern array will dynamically allocate an array of a given siize without mentioning the same.
- Do boundary checking of tiling algorithms when matrices multiplied are rectangular and dimensions are not multiples of tile width.