HETEROGENEOUS COMPUTATION BASED ON GPU

2022

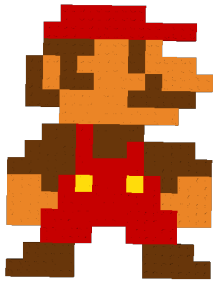
Harold Molina-Bulla DTSC-UC3M

INDEX

2022

HISTORY

- The game business needs mor complex images:



Harold Molina-Bulla DTSC-UC3M

HISTORY

- The CPUs did not have enough computational power to rendering the images.
- Not capable to render in “real-time” the images, and execute the game
- Solution:
  - Separate tasks
    - CPU for logic control of the game
    - GPU to render images and display
  - Render: create complex images and surfaces in 2 or more dimensions (4^{th} dimension: time)

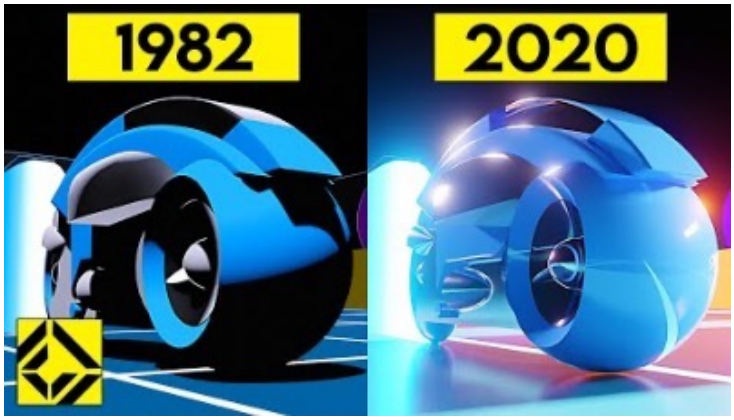
HISTORY

- To render:
  - Needs vectorial operations
  - Apply the same transformation to a big set of data (pixels in an image or boxels in a cube).
  - Single instruction (or single program) to multiple data (pixels in an image)

Harold Molina-Bulla DTSC-UC3M

HISTORY

- Movie Rendering:
  - TRON (Disney) First movie rendered by computer: CrayX-1 1982



2022

HISTORY

- More parallel simple cores, more computational power
- Someone asks:
  - All is vectorial algebra, what if….
    - Use to training Machine Learning
    - Complex Neural Networks -> Deep Learning
    - Complex Models -> Meteorology
    - Complex Analysis -> Montecarlo Analysis

Harold Molina-Bulla DTSC-UC3M

HISTORY

- CRYPTO-CURRENCY MINING
  - GPUs are very good for Crypto Currency
  - Consumes a lot of energy
    - Environment impact (not so green)
    - Farms of GPUs for Crypto Currency
    - No GPUs available in the commerce, or too expensive
      - Gamers Angry.
      - Scientist Angry
      - Greenpeace Angry

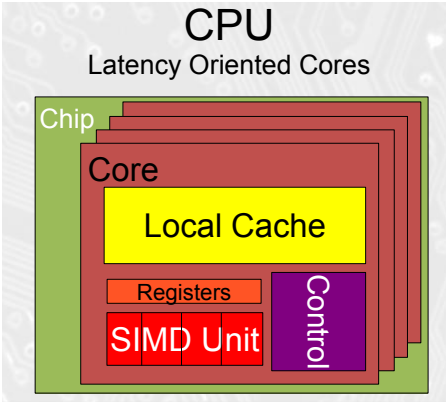
INTRODUCTION

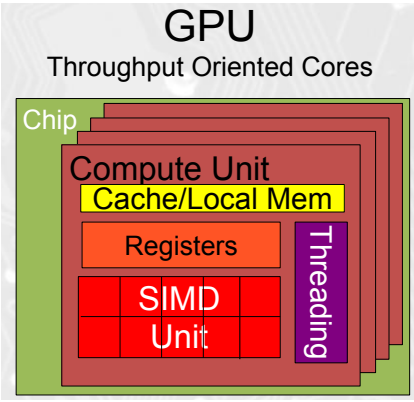
- Heterogeneous computation means programs which are executed partially in a general purpouse computer (**host**), and specific parts of the code (**kernels**) are executed in specific purpouse device (**device**)
- The host will execute the secuential code (or parallel threads in a MIMD)
- The device will execute parallel code (parallel threads in a SIMD or SPMD: Single Program/Multiple Data)
- Each architecture is very different, and oriented to different task kind.

GPU

- Graphics Processing Unit
  - Vectorial processor (?)
    - Oriented to processesing in parallel data vectors
    - Single Instruction Execution over Data Sets
    - Very efficient parallel execution in comparation with sequential instructions

DIFFERENCE BETWEEN CPU VS GPU

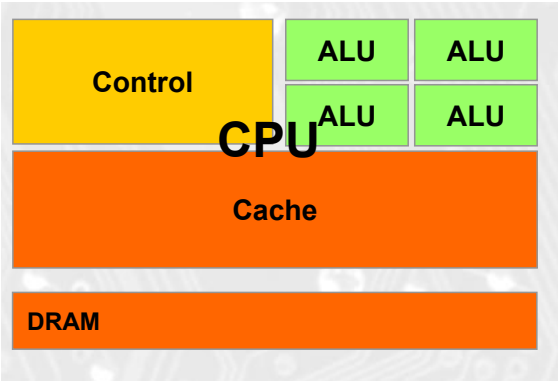




Harold Molina-Bulla DTSC-UC3M

CPU ARCHITECTURE

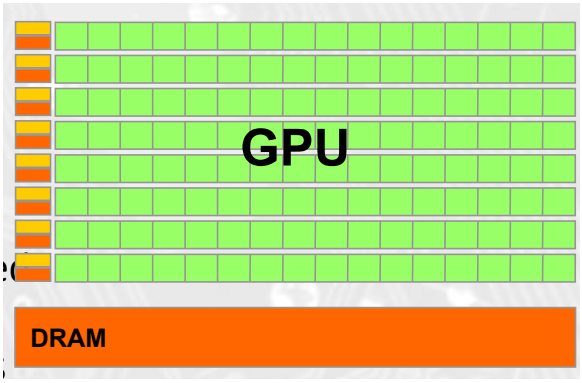
- Latency Oriented Design
  - Large caches
    - Convert long latency memory access to short latency cache access
  - Sophisticated control
    - Branch prediction for reduced branch latency
    - Data forwarding for reduced data latency
  - Powerful ALU
    - Reduced operation latency



2022

GPU ARCHITECTURE

- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Long latency but heavily piplined for high throughput
- Require massive number of threads to tolerate latencies



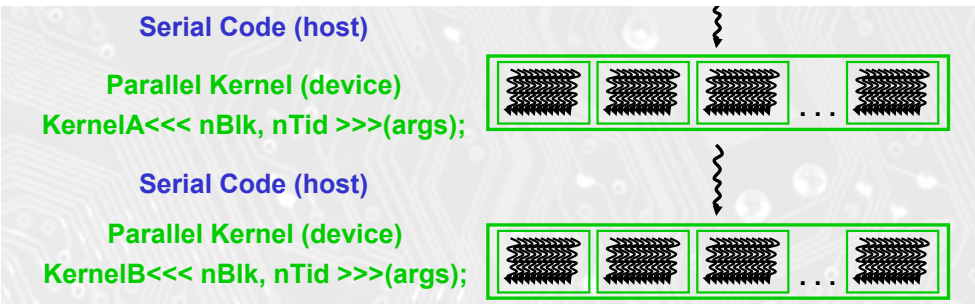
- CPUs for sequential parts where latency matters
  - CPUSs can be 10+X faster than GPUs for sequential code
- GPUSs for parallel parts where throughput wins
  - GPUs can be 10+X faster than CPUs for parallel code

PROGRAMMING LANGUAGES

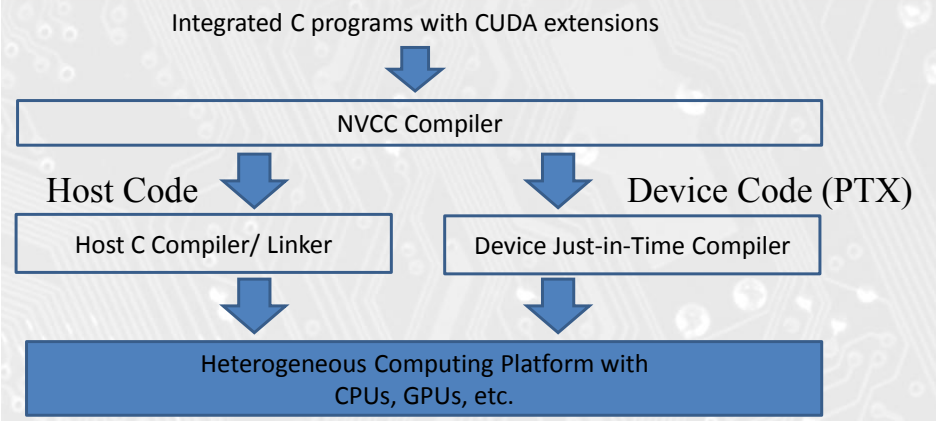
- CUDA C: Nvidia native programming language oriented to NVidia GPUs.
  - Hierarchical thread organization
  - Main interfaces for launching parallel execution
  - Thread index(es) to data index mapping
- OpenCL: Open Framework to programming across heterogeneous architectures
  - Same language, independent of hardware
  - Compiler creates specific binaries for specific hardware
  - Allows heterogeneous (mixed) architectures

CUDA/OPENCL EXECUTION MODEL

- Heterogeneous host+device application C program (we will use Python)
- Sequential parts in host C/Python code
- Parallel parts in device SPMD kernel C code

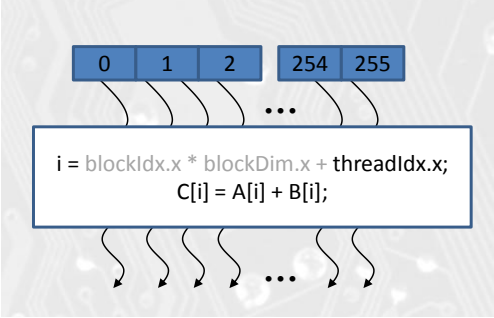


COMPILING CUDA PROGRAM



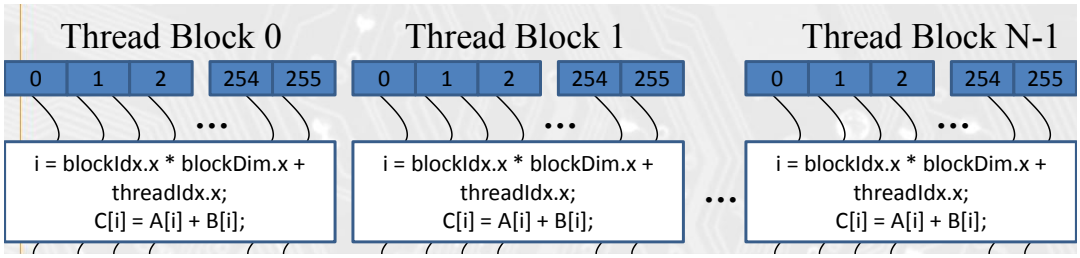
GPU EXECUTION

- A CUDA kernel is executed by an array of threads
  - All threads in a grid run the same kernel code (SIMD)
  - Each thread has indexes that it uses to compute memory address and make control decisions



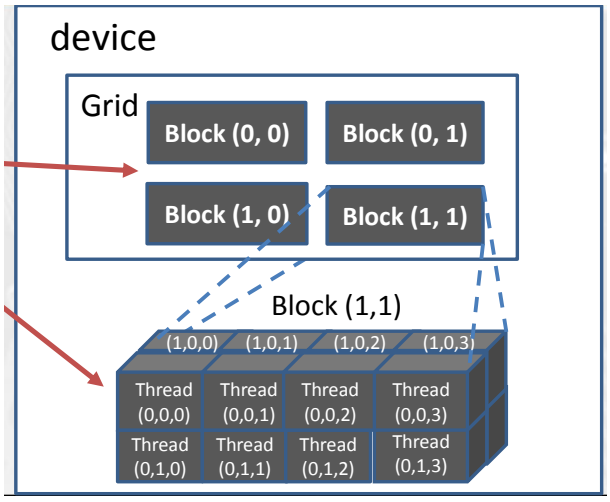
THREAD BLOCKS:SCALABLE COOPERATION

- Divide thread array into multiple blocks
  - Threads within a block cooperate via shared memory, atomic operations and barrier synchronization
  - Threas in different blocks do not interact



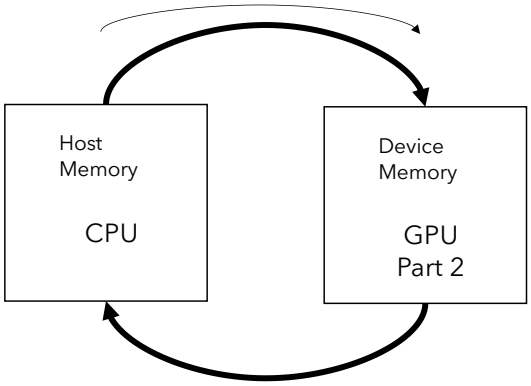
INDEXES

- Each thread uses indixes to decide what data to work on
  - blockIdx: block index which is executing
  - threadIdx: thread is executing
- Simplifies memory addressing when processing multidimensional data



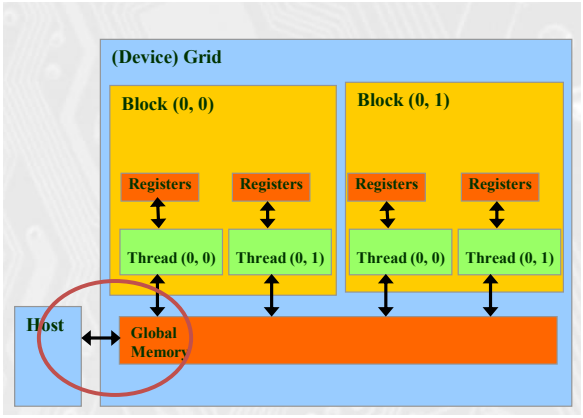
DATA ALLOCATION AND MOVEMENT

- Allocate device memory for data (data source and results)
- Copy data from source (host) to device memory allocated
- Launch kernel code in device
- Copy results to Host



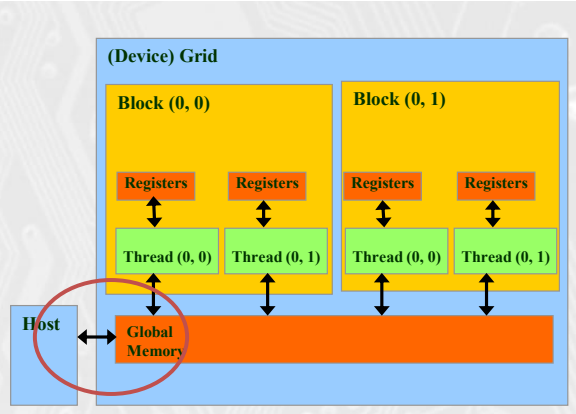
CUDA MEMORY MANAGMENT

- Device code can:
  - R/W per thread registers
  - R/W all-shared global memory
- Host code can
  - Transfer data to/from per grid global memory



CUDA MEMORY MANAGMENT

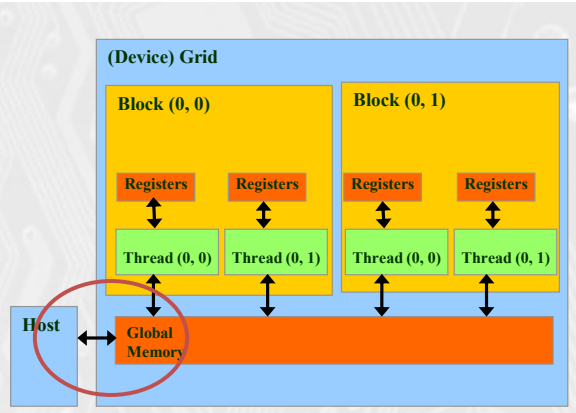
- cudaMalloc()
  - Allocates object in the device global memory
  - Address reference and Size of allocated in terms of bytes
- cudaFree()
  - Frees object from device global memory



Harold Molina-Bulla DTSC-UC3M

CUDA MEMORY MANAGMENT

- cudaMemcpy()
  - Memory data transfer
  - Bottleneck in communications
  - Transfer to device is asynchronous



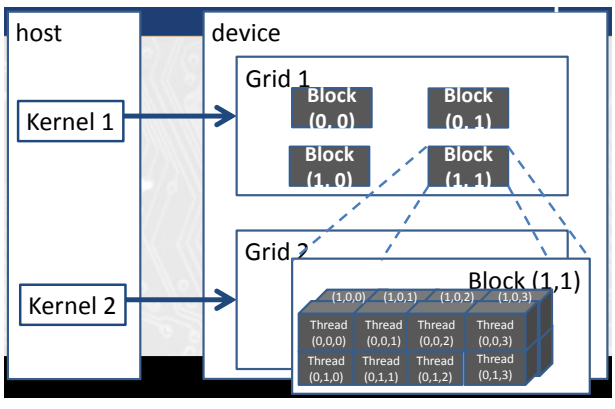
KERNEL-BASED SIMD PROGRAMMING

- Three types of functions, depending where they are executed

	Executed on the:	Only callable from the:
__device__ float DeviceFunc()	device	device
__global__ void KernelFunc()	device	host
__host__ float HostFunc()	host	host

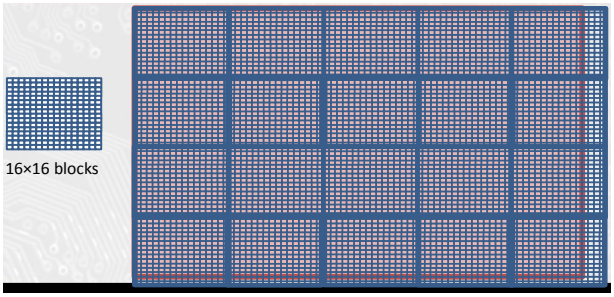
MULTIDIMENSIONAL GRIDS

- Multi-dimensional block and thread indices
- Mapping block/threads indices to data indices



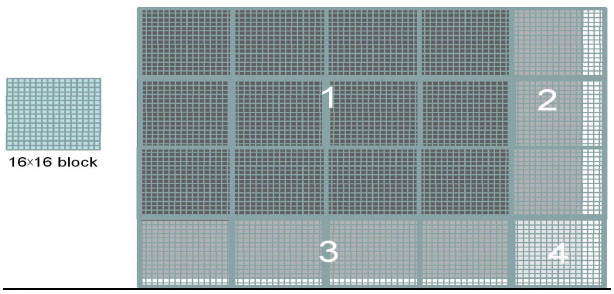
PROCESSING 2D MATRICES

- Example: Define blocks of 16x16 blocks (256 parallel threads)
- Define N blocks, where N x Block size must fit in number of cores available in device.
- Each position will be addressed by r/o index provided by the execution context.



PROCESSING 2D MATRICES

- If block not fits in data:
  - Check if indexes index a defined memory position
    - If yes, execute program
    - If not, return null
  - BIG PROBLEM HERE
    - The GPU does not have branch prediction, then double the execution time.



Harold Molina-Bulla DTSC-UC3M

MEMORY INDEXING

All the data is a vector, then you have to construct the memory position using the indexes