Threads and Blocks

- A thread is associated with each data element
- Threads are organized into blocks
- Blocks are organized into a grid

GPU hardware handles thread management, not applications or OS

Programming GPU

- CUDA programming model
  - Single Instruction, Multiple Thread (SIMT)
  - Threads are blocked to “Thread Block” and executed in groups of 32 threads. Multithreaded SIMD processor executes a whole block of threads.

```c
// C version of DAXPY loop.
void daxpy(int n, double a, double*x, double*y)
{
  for (int i=0; i<n; i++)
    y[i] = a*x[i] + y[i];
}

// CUDA version.
__host__ // Piece run on host processor.
int nblocks = (n+255)/256; // 256 CUDA threads/block
daxpy<<<nblocks,256>>>(n,2.0,x,y);

__device__ // Piece run on G-P-GPU.
void daxpy(int n, double a, double *x, double *y)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i<n) y[i]=a*x[i]+y[i];
}
```

Programmer’s View of Execution

Create enough blocks to cover input vector (Nvidia calls this ensemble of blocks a Grid, can be 2-dimensional). Conditional `(i<n)` turns off unused threads in last block.
NVIDIA GPU Architecture

- Similarities to vector machines:
  - Works well with data-level parallel problems
  - Scatter-gather transfers
  - Mask registers
  - Large register files

- Differences:
  - No scalar processor
  - Uses multithreading to hide memory latency
  - Has many functional units, as opposed to a few deeply pipelined units like a vector processor

Example

- Multiply two vectors of length 8192
  - Code that works over all elements is the Grid
  - Thread blocks break this down into manageable sizes
  - 512 elements per block
  - Thus, Grid size = 16 thread blocks
  - SIMD instruction executes 32 elements at a time
  - Thread Block is analogous to a strip-mined vector loop with vector length of 32
  - Block is assigned to a multithreaded SIMD processor by the thread block scheduler
  - Current-generation GPUs (Fermi) have 7-15 multithreaded SIMD processors

NVIDIA GeForce GTX 480 GPU
(Fermi architecture) (2010)

- A multithreaded SIMD processor: each has its own SIMD Thread scheduler
- Thread Block Scheduler

NVIDIA GeForce GTX 480 GPU
(Fermi architecture) (2010)

- 2 warp schedulers
- 3.2 billion transistors
- 1345 GFLOPS
Example

- NVIDIA GPU core has 32,768 registers
  - Divided into lanes
  - Each SIMD thread is limited to 64 registers (vector)
  - A SIMD thread can have up to:
    - 64 vector registers of 32 32-bit elements (= 2K 32-bit elements)
    - 32 vector registers of 32 64-bit elements
  - Fermi has 16 physical SIMD lanes, each containing 2048 registers ($2^{15}/2^4$)
  - CUDA Threads of a Thread Block can collectively use up to 1024 registers in order to handle 32 elements of each thread of SIMD instructions with 16 SIMD Lanes.

Terminology

- Threads of SIMD instructions
  - Each has its own PC
  - Thread scheduler uses scoreboard to dispatch
  - No data dependencies between SIMD threads!
  - Keeps track of up to 48 threads of SIMD instructions
    - Hides memory latency
  - Thread block scheduler schedules blocks to SIMD processors
  - Within each SIMD processor:
    - 32 SIMD lanes
    - Wide and shallow compared to vector processors

Hardware Execution Model

- GPU is built from multiple parallel cores, each core contains a multithreaded SIMD processor with multiple lanes but with no scalar processor
- CPU sends whole “Grid” over to GPU, which distributes thread blocks among cores (each thread block executes on one core)
  - Programmer unaware of number of cores

Block Diagram of a Multithreaded SIMD Proc.
Scheduling of threads of SIMD instructions

- Two level scheduling
  - Thread Block Scheduler assigns Thread Blocks to multithreaded SIMD processors
  - The SIMD Thread Scheduler within a SIMD processor schedules when threads of SIMD instructions should run
- NVIDIA GPU
  - One warp of 32 threads
  - Warp threads are interleaved in execution on a single core to hide latencies
  - A single thread block can contain multiple warps, all mapped to single core
  - Can have multiple blocks running on a core

NVIDIA Instruction Set Arch.

- ISA is an abstraction of the hardware instruction set
  - “Parallel Thread Execution (PTX)”
  - Uses virtual registers
  - All instructions can be predicated
  - Translation to machine code is performed in software
- Example:
  ```
  shl.s32 R8, blockIdx, 9 ; Thread Block ID * Block size (512 or 256)
  add.s32 R8, R8, threadIdx; R8 = i = my CUDA thread ID
  ld.global.f64 RD0, [X+R8] ; RD0 = X[i]
  ld.global.f64 RD2, [Y+R8] ; RD2 = Y[i]
  mul.f64 RD0, RD0, RD4 ; Product in RD0 = RD0 * RD4 (scalar a)
  add.f64 RD0, RD0, RD2 ; Sum in RD0 = RD0 + RD2 (Y[i])
  st.global.f64 [Y+R8], RD0 ; Y[i] = sum (X[i]*a + Y[i])
  ```

Implications of SIMT Model

- All data transfers are scatter-gather!, as individual threads perform scalar loads and stores
  - GPU adds hardware to dynamically coalesce individual thread loads and stores to mimic vector loads and stores
  - Every thread has to perform strip-mining calculations redundantly (“am I active?”) as there is no scalar processor equivalent
  - GPU programmer must ensure that adjacent CUDA threads access nearby addresses at the same time that can be coalesced into a few blocks.

Conditionals in SIMT model

- Simple if-then-else are compiled into predicated execution, equivalent to vector masking
- More complex control flow compiled into branches
  - In NVIDIA terms, a branch diverges.
- How to execute a vector of branches?
Branch divergence

- Hardware tracks which threads take or don’t take branch
- If all go the same way, then keep going in SIMD fashion
- If not, create mask vector indicating taken/not-taken
- Keep executing not-taken path under mask, push taken branch PC+mask onto a hardware stack and execute later
- When can execution of threads in warp reconverge?

Conditional Branching

- GPU branch hardware uses internal masks (like vector)
- Also uses
  - Branch synchronization stack
  - Entries consist of masks for each SIMD lane
  - i.e. which threads commit their results (all threads execute)
  - Instruction markers
  - to manage when a branch diverges into multiple execution paths
  - Push on divergent branch
  - …and when paths converge
  - Act as barriers, pops stack
- Per-thread-lane 1-bit predicate register, specified by programmer

Conditional Branching Example

```c
if (X[i] != 0)
    X[i] = X[i] – Y[i];
else X[i] = Z[i];
```

```c
ld.global.f64 RD0, [X+R8] ; RD0 = X[i]
setp.neq.s32 P1, RD0, #0 ; P1 is predicate register 1
if P1 bra ELSE1, *Push ; if P1 false, go to ELSE1
    ld.global.f64 RD2, [Y+R8] ; RD2 = Y[i]
    sub.f64 RD0, RD0, RD2 ; Difference in RD0
    st.global.f64 [X+R8], RD0 ; X[i] = RD0
ELSE1:    ld.global.f64 RD0, [Z+R8]
    st.global.f64 [X+R8], RD0 ; X[i] = RD0
ENDIF1:  <next instruction>, *Pop ; pop to restore old mask
```

NVIDIA GPU Memory Structures

- Each SIMD Lane has private section of off-chip DRAM
  - “Private memory”
  - Contains stack frame, spilling registers, and private variables
- Each multithreaded SIMD processor also has local memory
  - Shared by SIMD lanes / threads within a block
- Memory shared by SIMD processors is GPU Memory
  - Host can read and write GPU memory
**GPU Memory Structure**

- **CUDA Thread**
- **Thread Block**
- **Grid 0**
- **Grid 1**
- **Inter-Grid Synchronization**
- **Main Memory**
- **CPU registers**
- **GPU registers**

**Memory Hierarchy**

- Cache size hierarchy is backwards from that of CPUs
- Caches serve to conserve precious memory bandwidth by intelligently prefetching

**Memory Prefetching**

- Graphics pipelines are inherently high-latency
- Cache misses simply push another thread into the core
- Hit rates of ~90%, as opposed to ~100%

**Memory Access**

- GPUs are all about 2D spatial locality, not linear locality
- GPU caches read-only (uses registers)
- Growing body of research optimizing algorithms for 2D cache model
**Fermi Architecture Innovations**

- Each SIMD processor has
  - Two SIMD thread schedulers, two instruction dispatch units
  - 16 SIMD lanes (SIMD width=32, chime=2 cycles), 16 load-store units, 4 special function units
  - Thus, two threads of SIMD instructions are scheduled every 2 clock cycles (Analogous to a multithreaded vector processor)

**GPU Limitations**

- Relatively small amount of memory, < 4GB in current GPUs
- I/O directly to GPU memory has complications
  - Must transfer to host memory, and then back
  - If 10% of instructions are LD/ST and other instructions are...
    - 10 times faster $1/(1 + .9/10) = \text{speedup of 5}$
    - 100 times faster $1/(1 + .9/100) = \text{speedup of 9}$

- Fast double precision
- Caches for GPU memory
  - L1 can be bigger than L2
- 64-bit addressing and unified address space
- Error correcting codes
  - For memory and registers
- Faster context switching
- Faster atomic instructions