GPU Architectures and Programming

Soumyajit Dey, Assistant Professor,
CSE, IIT Kharagpur

December 5, 2019

Course Organization

Topic	Week	Hours
Review of basic COA w.r.t. performance	1	2
Intro to GPU architectures	2	3
Intro to CUDA programming	3	2
Multi-dimensional data and synchronization	4	2
Warp Scheduling and Divergence	5	2
Memory Access Coalescing	6	2
Optimizing Reduction Kernels	7	3
Kernel Fusion, Thread and Block Coarsening	8	3
OpenCL - runtime system	9	3
OpenCL - heterogeneous computing	10	2
Efficient Neural Network Training/Inferencing	11-12	6

Handling Data Level Parallelism

Data parallel algorithms handle multiple data points in each basic step (single thread of control)

- Vector Processors : early style of data parallel compute
- Single Instruction Multiple Data (SIMD) in x86 : MMX (Multimedia Extensions), AVX (Advanced Vector Extensions)
- GPUs : have their own distinguishing characteristics

Vector Processors

- Vector registers : Each vector register is a fixed-length bank holding a single vector,
- Functional units are also vectorized,
- Original Scalar registers are also present.
- VMIPS has eight vector registers, and each vector register holds 64 elements, each 64 bits wide.

Vector Processors : Consider a simple $Y = a * X + Y$ operation

```
        L.D      F0,a          ;load scalar a
        DADDIU   R4,Rx,#512    ;last address to load
Loop:   L.D      F2,0(Rx)      ;load X[i]
        MUL.D    F2,F2,F0      ;a × X[i]
        L.D      F4,0(Ry)      ;load Y[i]
        ADD.D    F4,F4,F2      ;a × X[i] + Y[i]
        S.D      F4,9(Ry)      ;store into Y[i]
        DADDIU   Rx,Rx,#8      ;increment index to X
        DADDIU   Ry,Ry,#8      ;increment index to Y
        DSUBU    R20,R4,Rx     ;compute bound
        BNEZ     R20,Loop      ;check if done
```

(a) MIPS

```
L.D       F0,a        ;load scalar a
LV        V1,Rx       ;load vector X
MULVS.D   V2,V1,F0    ;vector-scalar multiply
LV        V3,Ry       ;load vector Y
ADDVV.D   V4,V2,V3    ;add
SV        V4,Ry       ;store the result
```

(b) VMIPS

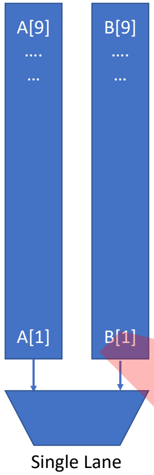
Figure: Assuming the data size < vector storage (Ref: CoA: a quantitative approach (Hennessy & Patterson))

In non-vectorized code, every ADD.D must wait for a MUL.D, and every S.D must wait for the ADD.D

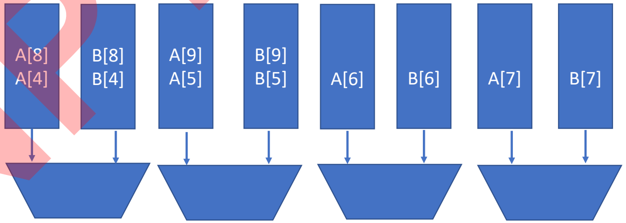
Vector Processors

- A vector instruction passes lot of parallel work to the hardware
- The FUs can be : fully parallel, or a combination of parallel and pipelined units
- If the clock rate of a vector processor is halved, doubling the number of lanes will retain the same potential performance.
- Work for compilers - loop vectorization, dependency handling

Vector Processors



Four add pipelines can complete four additions per cycle
Elements are interleaved



GPUs

Ideas from parallel instruction handling by vector architectures, ILP techniques etc were borrowed to accelerate graphics processing

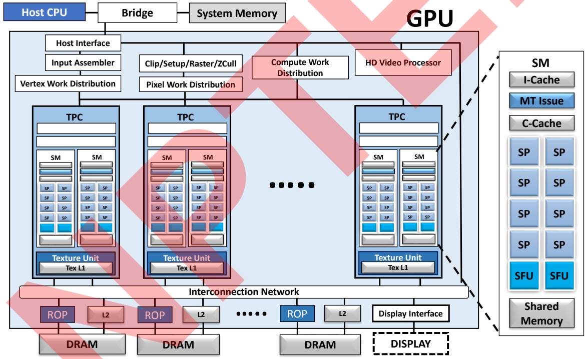


Figure: GPU systems (GeForce 8800) - Hennessy, Patterson (reproduced)

GPU Architecture (Tesla)

- Earlier figure depicts a GPU with an array of 128 streaming/scalar processor (SP) cores, organized as 16 multithreaded streaming multiprocessors (SM),
- Each SM has 8 SPs,
- 2 SMs together are arranged as independent processing units called texture/processor clusters (TPCs).

Early GPUs

Early GPUs accelerated the logical graphics pipeline

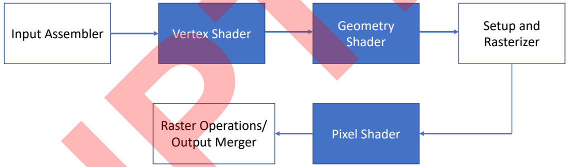


Figure: Graphics logical pipeline

Shader Programs

Graphics application sends the GPU a sequence of vertices grouped into geometric primitives—points, lines, triangles, and polygons.

- The input assembler collects vertices and primitives.
- Vertex shader programs map the position of vertices onto the screen, altering their position, color, or orientation.
- Geometry shader programs operate on geometric primitives (such as lines and triangles) defined by multiple vertices, changing them or generating additional primitives.

Shader Programs

Usually dataflow style, model how light interacts with different materials and to render complex lighting and shadows.

- The setup and rasterizer unit generates pixel fragments (which are potential contributions to pixels) that are covered by a geometric primitive.
- The pixel shader program fills the interior of primitives, including interpolating per-fragment parameters, texturing, and coloring.
- The raster operations processing (or output merger) stage : depth testing and stencil testing, color blending operation etc

Ref : "Computer Organization and Architecture" - Hennessy, Patterson (Appendix A on GPUs)

GPUs : massive multi-threading

Design goals

- Cover the latency of memory loads and texture fetches from DRAM
- Support fine-grained parallel graphics shader (and general parallel compute) programming models
- Virtualize the physical processors as threads and thread blocks to provide transparent scalability
- Simplify the parallel programming model to writing a serial program for one thread

First generation GPUs

- GeForce 256, introduced in 1999
- Contained fixed function vertex, pixel shaders programmed with OpenGL and the Microsoft DX7 API
- GeForce 3 - the first programmable vertex processor executing vertex shaders

- Ref for contents and here and subsequent places : "NVIDIA Tesla: A Unified Graphics and Computing Architecture" by Erik Lindholm, John Nickolls, Stuart Oberman, John Montrym, (NVIDIA) IEEE Micro, Volume 28, Issue 2, March 2008

Trade-off

- Vertex processors were designed for low-latency, high-precision math operations
- pixel-fragment processors were optimized for high-latency, lower-precision texture filtering - typically more busy (considering large triangulation)
- if these are fixed function blocks - difficult to select a fixed processor ratio
- Primary design objective for Tesla architecture - execute vertex and pixel-fragment shader programs on the same unified processor.
- Unification helps in 1) dynamic load balancing of varying vertex- and pixel-processing workloads, 2) introducing other shaders

Tesla architecture

We come back to GeForce 8800 GPU with 128 SPs organized as 16 SMs

- external DRAM control and fixed-function raster operation processors (ROPs) perform color and depth frame buffer operations directly on memory
- The interconnection network carries computed pixel-fragment colors and depth values from SPs to the ROPs
- The network also routes texture memory read requests from the SP to DRAM and read data from DRAM through a level-2 cache back to the SPs

Graphics in Tesla

- The input assembler collects vertex work
- Vertex work distributor distributes vertex work packets to the various TPCs
- The TPCs execute vertex/geometry shader programs
- output data is written to on-chip buffers
- buffers then pass their results to the viewport/clip/setup/raster/zcull block

We continue from here to general purpose processing

GPGPU

Each TPC has two SMs, each SM has

- eight streaming/scalar processor (SP) cores,
- two special function units (SFUs),
- a multi-threaded instruction fetch and issue unit (MT Issue),
- an instruction cache, a read-only constant cache,
- a 16-Kbyte read/write shared memory.

GPGPU

- Each SP core contains a scalar multiply-add (MAD) unit, giving the SM eight MAD units
- The SM uses its two SFU units for transcendental functions
- Each SFU also contains four floating-point multipliers
- In total an SM has eight MAD and floating-point multipliers

SIMT

GPU execution model

- SIMT architecture is similar to SIMD design, which applies one instruction to multiple data lanes.
- The difference is that SIMT applies one instruction to multiple independent threads in parallel, not just multiple data lanes.
- A SIMD instruction controls a vector of multiple data lanes together, a SIMT instruction controls the execution and branching behavior of one thread.

SIMT

- In contrast to SIMD vector architectures, SIMT enables programmers to write thread level parallel code for independent threads as well as data-parallel code for coordinated threads
- SIMT - essentially a single thread of SIMD instructions
- Each SM's multithreaded instruction unit creates, manages, schedules, and executes threads in groups of 32 parallel threads called *warps*
- Each SM manages a pool of 24 warps, with a total of 768 threads
- Each SM maps warp threads to the SP cores

Warp execution

- In each operation cycle, the SM warp scheduler selects one of the 24 warps
- An issued warp executes over four processor cycles
- The SP cores and SFU units execute instructions independently

ISA

- Support for floating-point, integer, bit, conversion, transcendental, flow control, memory load/store
- Floating-point and integer operations include add, multiply, multiply-add, minimum, maximum, compare, set predicate, and conversions between integer and floating-point numbers
- Transcendental function instructions include cosine, sine, binary exponential, binary logarithm, reciprocal, and reciprocal square root.
- Bitwise operators include shift left, shift right, logic operators, and move
- Control flow includes branch, call, return, trap, and barrier synchronization

Register File

Each SIMD processor (SM)

- has a large vector register file
- like a vector processor, these registers are divided logically across the SIMD Lanes, i.e. the SPs
- These numbers vary across across architecture families.

Fermi GTX 480 GPU

Has

- 16 SMs, total 512 CUDA cores
- Each SM has 32 SPs, 32,768 32-bit registers divided logically across executing threads
- Each SIMD Thread is limited to no more than 64 registers
- A warp has access to 64×32 registers which are 32 bit,
- Alternatively, considering double-precision floating-point operands, a warp has access to 32 vector registers of 32 elements, each of which is 64 bits wide.

Fermi Streaming Multiprocessor (SM)

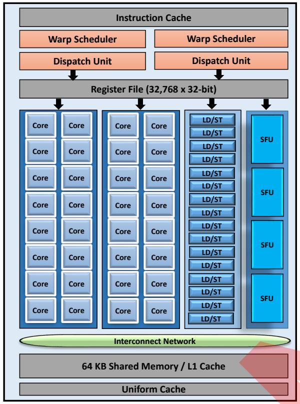


Figure: Fermi Streaming Multiprocessor (SM)

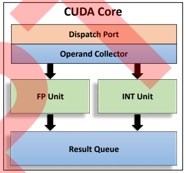


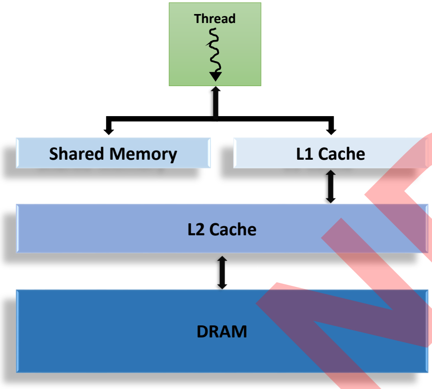
Figure: Single SP

- Each SM has 16 Load/store units (load/store data at each address to cache or DRAM.) - 16 SIMD lanes
- Each lane has 2048 registers
- Each SM has 4 SFUs, Each SP has one FP, one Integer ALU.
- ALUs also support Boolean, shift, move, compare, convert, bit-field extract, ...

Memory Hierarchy

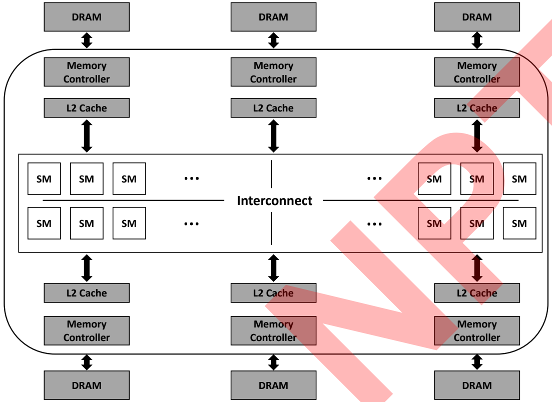
- Local memory for per-thread, private, temporary data (implemented in external DRAM)
- Shared memory for low-latency access to data shared by threads in the same SM
- Global memory for data shared by all threads of a computing application (implemented in external DRAM)

Fermi Memory Hierarchy



- Shared memory enables threads to cooperate, facilitates reuse of on-chip data, and reduces off-chip traffic.
- Each SM has 64 KB of on-chip memory that can be configured as 48 KB of Shared memory with 16 KB of L1 cache or as 16 KB of Shared memory with 48 KB of L1 cache.
- Source : NVIDIA Whitepaper on Fermi

Fermi Memory Hierarchy



- L1 (Data) cache + Shared memory is private to SMs along with read-only texture and constant caches
- L2 is unified for all SMs, 6 high-bandwidth DRAM channels
- Compared to CPU, GPUs have larger register file, smaller L1/L2 cache with higher bandwidth
- Ref : "The Architecture and Evolution of CPU-GPU Systems for General Purpose Computing" - Manish Arora

GPU ISA

- The instruction set target of the NVIDIA compilers is an abstraction of the hardware instruction set
- PTX (Parallel Thread Execution) provides a instruction set for compilers that remains same for different generations of GPUs
- PTX code gets translated to target hardware instructions while being loaded to GPU

PTX instructions

- format : opcode.type d, a, b, c;
- a,b,c, are source; d is destination operand
- Source operands are 32-bit or 64-bit registers or a constant value
- All instructions can be predicated by 1-bit predicate registers, which can be set by a set predicate instruction (setp)

GPUs becoming ubiquitous

GPUs have started finding wide usage in several domains where workloads have become intensive

- Mobile GPUs : ARM Mali, Adreno GPUs (Qualcom) - accelerate graphics as well as compute tasks
- NVIDIA in embedded space : Jetson TX/ Nano / AGX Xavier $\Rightarrow$ Multi core ARM CPU + 128-512 core GPU targeting AI / Deep Learning tasks
- NVIDIA Drive : for implementing autonomous car and ADAS functionality powered by deep learning (Tesla cars !!)

GPUs as mobile workload accelerators

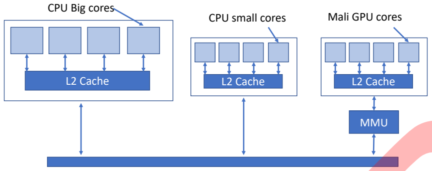


Figure: Typical architecture of an ARM based Mobile SoC

- Objective : Maximize performance and reduce power consumption
- Developers need to map the workload across the whole CPU + GPU system
- RenderScript for Android SDK, OpenCL - language support for data parallel computation on Mobile devices

Integrated GPUs in Desktop Systems

With the release of AMD's Fusion and Intel's Ivy Bridge architecture (i3, i5, i7) in 2011, the trend of fused CPU-GPU architectures started

- CPU and GPU access the same physical memory such that zero-copy transfers can be employed
- Zero- copy transfers ensure coherency; translate pointers to memory buffers for the common CPU and GPU address space, but do not actually transfer data.
- Bad effect - CPU and GPU compete for memory bandwidth of the shared physical memory

Integrated GPUs in Desktop Systems

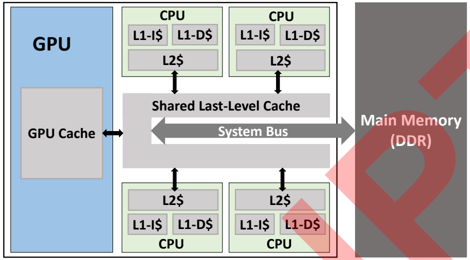


Figure: Fused CPU-GPU with shared LLC

- In more recent architectures, Intel Broadwell and beyond, CPU and GPU were further integrated
- They access the shared last level cache (LLC)
- This helps in CPU and GPU executing computational kernels on the same data in parallel collaboratively (LLC enables cache coherence between CPU and GPU)
- "Co-Scheduling on Fused CPU-GPU Architectures with Shared Last Level Caches" - Henkel et. al.

Jetson Series from NVIDIA

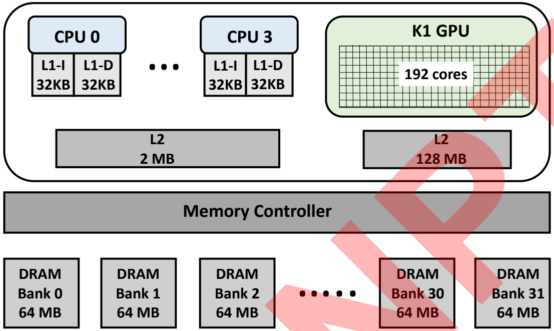


Figure: Jetson TK1

- TK1 SOC incorporates a quad-core 2.32 GHz 32-bit ARM machine and an integrated Kepler GK20a GPU
- The CPUs share a 2-MB L2 cache
- The GPU has 192 cores and a 128-KB L2 cache
- The CPU also has 'little' ARM cores (not shown) - low power, low performance

NVIDIA Drive series of systems

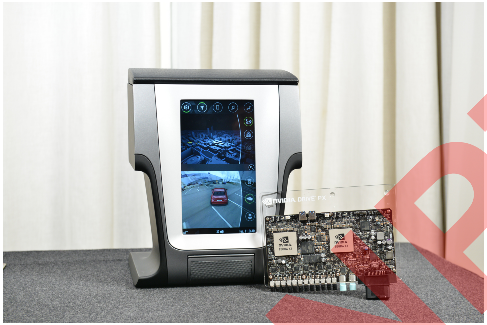


Figure: Source- Wiki, NVIDIA Drive PX Platform

- The Nvidia Drive PX 2 is based on 1/2 Tegra SoCs where each SoC contains 2 Denver cores, 4 ARM A57 cores and a GPU from the Pascal generation
- Useful for implementing high throughput real time neural net processing - self driving / drive assist systems