All of the opinions expressed in this presentation are my own and do not reflect any held by NVIDIA.
OUTLINE

CPU versus GPU
  Why are they different?
CUDA
  Terminology
GPU
  Pascal
Opportunities
  What skills you should know

CPU
Problems Solved Over Time

Complex code with all sorts of different demands and use cases.
**CPU**

Priorities

Latency
- Event driven programming

Managing complex control flow
- Branch prediction & speculative execution

Reduction of memory access
- Large caches

Small collection of active threads

Results in less space devoted to math

---

**GPU**

Problems Overtime

Real-time graphics and simulations

Recent boom of AI
GPU

Priorities

Parallelization

Work-loads related to games are massively parallel

“4K" = 3840x2180 ~60fps -= 500M threads per sec!

“SOL” of VR for 2018 -> 4k @ 120hz -= 1B threads!

Latency Trade-off

Advanced batching and scheduling of data

Computation

Less complex control flow, more math

Most space devoted to math

With a smart thread scheduler to keep it busy

CPU

FUTURE

Simpler more power efficient (IOT)

Integration of components == cheaper

GPU

Perf/w

Low-latency (VR requirement)

HPC “co-processors”

Power efficiency! Phi, Tesla, Fire pro.

Fast interconnects (NV-Link, future PCI-E)

“Heterogeneous computing”
WHY CARE?

Understand what architectures are best for the problem you need to solve

- FEM/Graphics -> Co-proc/GPU
- GUI/OS -> CPU
- DNN -> GPU/co-proc/TPU?

It's a spectrum, don't force the problem on a poorly suited architecture

Example: Deep Neural Networks

CUDA

Compute Unified Device Architecture

NVIDIA’s parallel language

- OpenCL is similar

API/language focuses on compute features

- LD/ST to generic buffers
- Support for advanced scheduling features
CUDA Constructs

Thread
- Work item

Block/CTA
- Work group

Grid
- Index space

Thread
Per-thread local memory

Warp = 32 threads
Share through local memory

Block/CTA = Group of warps
Shared mem per CTA: inter-CTA sync/communication

Grid = Group of CTAs that share the same kernel
Global mem for cross-CTA communication
CUDA
Warp

Utilize existing libraries
- cuFFT (10x faster)
- cuBLAS (6-17x faster)
- cuSPARSE (8x faster)

Application integration
- Matlab, Mathematica
- DIY

CUDA, OpenCL, DirectX Compute

Source:
CUDA
Compute Integration

OpenACC

```c
#include <stdio.h>
#define N 1000000
int main(void)
double pi = 0.0f; long i;
#pragma acc region for
for (i=0; i<N; i++) {
    double t = (double)((i+0.5)/N);
    pi +=4.0/(1.0+t*t);
}
printf("pi=%16.15f\n",pi/N);
return 0;
```

cuFFT

```c
#define NX 64
#define NY 64
#define NZ 128

cufftHandle plan;
cufftComplex *data1, *data2;
cudaMalloc(void**)&data1, sizeof(cufftComplex)*NX*NY*NZ;
cudaMalloc(void**)&data2, sizeof(cufftComplex)*NX*NY*NZ;

/* Create a 3D FFT plan. */
cufftPlan3d(&plan, NX, NY, NZ, CUFFT_C2C);
/* Transform the first signal in place. */
cufftExecC2C(plan, data1, data1, CUFFT_FORWARD);
/* Transform the second signal using the same plan. */
cufftExecC2C(plan, data2, data2, CUFFT_FORWARD);
/* Destroy the cuFFT plan. */
cufftDestroy(plan);
cudaFree(data1); cudaFree(data2);
```

WHY CARE?

Constructs reflect:
- Communication boundaries
- Read/write coherency
- Warps
  - Low level primitive HW operates on
  - Blocks/CTAs are made up of integer # of warps (48 threads = 2 warps, not 1.5)
- AMD/Intel HPC chips have similar constructs

Use existing libraries
- Don't reinvent the wheel, better use of your time
GPU ARCHITECTURE

Evolution

Fixed function (Geforce 256, Radeon R100)
  Hardware T&L, Render Output Units/ Raster Operations Pipeline (ROPs)
Programmability (Geforce 2 -> 7)
  Vertex, Pixel shaders
Unification (Geforce 8)
  Generic “compute” units
Generalization (Fermi, Maxwell, AMD GCN)
  Queuing, batching, advanced scheduling
  Compute focused

GPU ARCHITECTURE

GPU Last Week - Geforce 3

Source: http://hexus.net/tech/reviews/graphics/251-hercules-geforce-3-ti500/?page=2
GPU ARCHITECTURE
GPU Yesterday - GTX 980 (GM204)

GPU ARCHITECTURE
GTX 1080 (GP104)
**PASCAL**

**GPC**

Similar to a CPU core
Stamp out for scaling
Contains multiple SMs

**Graphics Features**
- Polymorph Engine
- Raster Engine

**Compute Features**
- CTA (work group)

---

**PASCAL**

**SM**

Schedule threads efficiently
Maximize Utilization

4 Warp schedulers
- 2 dispatch units per scheduler
- Warp == 32 threads
- 1 SM can handle 64 warps
PASCAL Warp Scheduler

Avoid Divergence!
Shuffle your program to avoid divergence within a warp

Size of CTA matters!
Programmer thinks in CTAs, HW operates on warps
CTA size 48 threads
48/32 = 1.5 warps, second warp is "partially occupied"

Scheduler
2 dispatch units need two independent inst per warp
Series of dependent instructions = Poor utilization
Poor instruction mix (heavy on a certain type) == Poor utilization

PASCAL Utilization
**PASCAL**

*Utilization*

Latency

Memory access - ~10 cycles -> 800 cycles

“Cuda cores” take a few cycles to complete

# of warps on a SM tied to usage of register file

Complex program with high # of registers fill up RF

Decreases # of warps available on the SM for issue

Balance work-load

Tex versus FP versus SFU versus LD/ST

Avoid “peak” utilization of any one unit

---

**PERF #’S**

<table>
<thead>
<tr>
<th></th>
<th>PRICE ($)</th>
<th>TFLOPS (SP)</th>
<th>TFLOPS (DP)</th>
<th>GFLOPS/W (SP)</th>
<th>GFLOPS/W (DP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 1080</td>
<td>599.00</td>
<td>8.23</td>
<td>0.257</td>
<td>45.6</td>
<td>1.43</td>
</tr>
<tr>
<td>NVIDIA GV100</td>
<td>??? (Tesla was P100 5599.00)</td>
<td>15</td>
<td>7.5</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>Intel Xeon PHI 7290</td>
<td>6254.00</td>
<td>6</td>
<td>3.5</td>
<td>24.5</td>
<td>14.3</td>
</tr>
<tr>
<td>Proj. Scorpio</td>
<td>???</td>
<td>6</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>PS4 Pro</td>
<td>399.99</td>
<td>4.12</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
</tbody>
</table>
WHAT’S GOOD ENOUGH?

GPU demands are never ending...
  Physically based materials for lighting
  Indirect illumination
  Physically based materials for interaction
  4k? 5k? 8k?
  VR doubles required work... needs to hit 120fps+
API demands are never ending...
  New low-level APIs (Vulkan, DX12) create their own problems...

OPPORTUNITIES

NVIDIA/AMD/Intel/Google/Qualcomm/Apple
  Very Strong C/C++ fundamentals
  Very Strong OS/Algorithms fundamentals
  Very Strong Graphics fundamentals
  Parallel Programming w/ emphasis on Performance
  Compiler experience a plus

Other industries
  Games Industry
  Biochemistry simulations
  Weather/climate modeling
  CAD
QUESTIONS?

cschultz@nvidia.com