Multicores, Manycores, and Accelerators

18-613: Foundations of Computer Systems
7th Lecture, March 21, 2019

Instructor:
Franz Franchetti
Outline

- Multicores, Manycores, accelerators
- Experimental accelerators
- Latest Nvidia GPUs
- CUDA
- Summary
The Future is Parallel and Heterogeneous

Programmability? Performance portability?
The Future Is Here: Accelerators

Google TPU 2.0
600 GB/s, 45 Tflop/s

Nvidia V100 GPU
100 Tflop/s tensor operations
7.5 Tflop/s FP64 SIMT

Intel Nervana Neural Network Processor
aka Lake Crest, release date 2019

Intel Arria 10 GX FPGA
15Gbps SERDES transceivers
1,150K logic elements, 53 Mb
8 GB DDR4 ECC
Intel Multicores and Manycores

Desktop and Server Multicores

- **2005**: Pentium D
  - 2 cores, 3.6 GHz

- **2007**: Core 2 Quad
  - 4 cores, 2.67 GHz

- **2010**: Core i7 980X
  - 6 cores, 3.3 GHz

- **2014**: Xeon E5-4624Lv2
  - 10 cores, 2.5 GHz

- **2015**: Xeon E7-8890 v3
  - 18 cores, 2.5 GHz

- **2019**: Xeon Platinum 8176F
  - 28 cores, 3.8 GHz

Experimental Chips and Server Manycores

- **2008**: Larrabee GPU
  - 32 cores, 2 GHz

- **2009**: Intel SCC
  - 48 cores, 1 GHz

- **2013**: Xeon Phi 5120D
  - 60 cores, 1.05 GHz

- **2016**: Xeon Phi 7290F
  - 72 cores, 1.5 GHz
Other CPU-Style Multicores/Manycores

- **2006**
  - IBM Cell BE
  - 8+1 cores, 3.2 GHz

- **2007**
  - Tilera Tile64
  - 64 cores, 900 MHz

- **2008**
  - ClearSpeed CSX700
  - 192 cores, 250 MHz

- **2012**
  - Parallella
  - 16 cores, 1 GHz

- **2013**
  - IBM POWER9
  - 24 cores, 4 GHz

- **2013**
  - Oracle SPARC T5
  - 16 cores, 3.6 GHz

- **2014**
  - Fujitsu SPARC64 XIfx
  - 32+2 cores, 2.2 GHz

- **2017**
  - NEC SX-Aurora
  - 8 cores, 1.6 GHz
Math Co-Processors and Sound Cards

x87 Co-Processors

1980
Intel 8087
x87 FPU, 10 MHz

1987
Intel 80387
IEEE754 x87 FPU, 33 MHz

1989
Intel 80486
x87 FPU integrated

1990
Weitek 4167
x87 FPU, 33 MHz

Audio Co-Processors

1989
Sound Blaster 1.0
11 voice FM synthesizer

2012
SoundBlaster X-Fi
Advanced DSP, 48kHz

2008
Platform Controller Hub
Sound, clock, display

2014
On chip PCH
SoC design
Graphics Processors (GPUs)

Nvidia

1995
Nvidia NV1
2D/3D GPU

2006
Nvidia G80
CUDA

2009
Nvidia Tesla C1060
double GPU computing

2019
Nvidia V100
Tensor cores, 125 Tflop/s

ATI/AMD

1992
ATI Mach32
GUI acceleration

2000
ATI Radeon
DirectX 7.0 3D, MPEG-1

2014
Radeon R9 295X2
1.43 Tflop/s double

2017
Opteron X3000 APU
4 cores + 3rd gen GCN GPU
Intel GPUs

1998
Intel 740
VGA card

2008
Larrabee
Discrete GPU

2008
GMA X4500HD
G45 chipset

2018
UHD 630
Coffe Lake, on-die
Embedded and System on a Chip (SoC)

Raspberry Pi
Broadcom BCM2837B0
1.4 GHz, ARM Cortex A53

Qualcomm Snapdragon 855
ARM CPU, GPU, DSP, CV modem

Apple A12
6 cores, ARM, 2.5 GHz

Intel Quark
1 core, 32 MHz

**Visual Subsystem**
- Adreno 680 GPU
- Vulkan 1.1 API support
- HDR gaming 10-bit color depth, Rec. 2020 color gamut
- Physically Based Rendering
- API Support: OpenGL ES 3.2, OpenGLES 2.0, FP, Vulkan
- Hardware-accelerated H.265 and VP9 decoder
- HDR Playback Codec support for HDR10+, HDR10, HLG and Dolby Vision
- Volume VR audio playback
- 8K, 360 VR video playback

**RF Front-End**
- Comprehensive 5G/4G modem-to-antenna solution
- QPMSD 5G mmWave antennas module
- Qualcomm Adaptive Antenna Tuning
- Qualcomm Envelope Tracking
- High power transmits (3.0x)

**Security**
- Secure Processing Unit featuring mobile payments and Qualcomm® 3D Sonic Sensor
- Biometric Authentications: Fingerprint, Iris, Voice, Face

**Memory**
- Memory Speed: 2333 MHz
- Memory Type: 4/6-bit, LPDDR4x
- Memory Density: up to 16 GB

**Connectivity Specifications**
- Bluetooth Version: 5.0
- Bluetooth Speed: 2 Mbps
- Satellite Systems Support: Dual frequency GNSS, GPS, GLONASS, Beidou, Galileo, QZSS, SBAS
- Near Field Communications (NFC) Support
- USB Version: 3.1, USB Type-C Support

**Charging**
- Qualcomm® Quick Charge™ 4+ technology

**5G Modern**
- Snapdragon™ X50 5G Modem
- Supported Technologies: 5G NR
- Support for sub-6 GHz and mmWave
- Support for Transmission: mmWave, 800 MHz bandwidth, 8 carriers, 2x2 MIMO
- Sub-6 GHz 100 MHz bandwidth, 4x4 MIMO

**6G Modern**
- Snapdragon™ X65 LTE modem
- Cellular Technologies: LTE FDD, LTE TDD with CEPS support, LAA, LTE broadcast, WCDMA, TD-SCDMA, CDMA, EVDO, GSM/EDGE
- Download LTE Cat. 20 up to 20 Gbps, 7x20 MHz carrier aggregation, 2x26-GAM, 4x2 MIMO on five carriers
- Uplink LTE Cat 13 up to 383 Mbps, 3x20 MHz carrier aggregation, 2x26-GAM, Uplink Coa Compression
- Multi SIM with support for Dual SIM Dual VoLTE (DSV), and Dual SIM Dual Standby (DSDS) +AA
- Next-generation Calling Services including Voice with VRRC to 3G and 2G, Ultra HD Voice (UHS), and CSFB to 3G and 2G

**WiFi**
- Qualcomm® Wi-Fi 6-ready mobile platform
- WiFi Standards: 802.11a/b/g/n/ac, 802.11ax
- Wi-Fi 6E Standards: 6 GHz band, 8x8 MIMO, 4x4 MIMO
- Wi-Fi 6E Spectral Bands: 2.4 GHz, 5 GHz
- Channel Utilization: 20/40/80 MHz
- MIMO Configuration: 2x2to2x4 monitors
- MU-MIMO
- Dual band simultaneous (DBS)
- Key Features: 802.11ax (80 MHz) 8x8 MU-MIMO
- 802.11ax (2x2 MIMO) 4x4 MU-MIMO
- Key Features: 802.11ax (80 MHz) 8x8 MU-MIMO
- 802.11ax (2x2 MIMO) 4x4 MU-MIMO
- Key Features: 802.11ax (80 MHz) 8x8 MU-MIMO
- 802.11ax (2x2 MIMO) 4x4 MU-MIMO
- Key Features: 802.11ax (80 MHz) 8x8 MU-MIMO
- 802.11ax (2x2 MIMO) 4x4 MU-MIMO

**CPU**
- 8x Qualcomm® Kryo™ 485
- Up to 2.84 GHz
- 64-bit Architecture
- 7nm Process Technology

**Camera**
- Qualcomm® Spectra™ 360 Image Sensor Processor
- Dual 4-bit ISPs
- Hardware accelerator for computer vision (CV/ISP)
- Up to 22 MP dual camera
- Up to 48 MP single camera
- Rec. 2020 color gamma, video capture
- Up to 10-bit color depth video capture
- Slow motion video capture at 720p HDR, 48 fps
- HEVC video capture
- Video Capture Formats: HDR10+, HDR10, HLG
- 4K HDR Video Capture with Portrait Mode (Blake)
- Multi-frame Noise Reduction (MNR)
- Real-time object classification, segmentation, and replacement

**Audio**
- Low-power Audio Subsystem with AI
- Hexagon Voice Assistant Accelerator for advanced voice use cases
- Qualcomm® AptX™ Audio Technology
- Native ESD support, PCM up to 384 kHz/32-bit
- Supports two audio words simultaneously: Xperience Assistant, Amazon Alexa, Baidu, Cortana
- Always-on echo cancellation and noise suppression; up to 4 mic for Android mic support for better voice recognition in tough conditions
- Qualcomm® aptX™ Adaptive audio technology
- aptX Adaptive adjusts performance for optimum quality, latency, and efficiency
- Qualcomm TrueWireless™ Stereo Plus technology

**Display**
- Maximum On-Device Display Support: Up to 4K HDR
- Maximum External Display Support: Up to two 4K HDR displays
- 10-bit color depth, Rec. 2020 color gamut

**Storage**
- 64 GB eMMC 5.0 (Qualcomm®)
- 128 GB eMMC 5.0 (Qualcomm®)
- 256 GB eMMC 5.0 (Qualcomm®)
- 512 GB eMMC 5.0 (Qualcomm®)
- 1 TB eMMC 5.0 (Qualcomm®)

**Miscellaneous**
- Qualcomm® Quick Charge™ 4+ technology
- Qualcomm® Quick Charge™ 5 technology
- Qualcomm® Quick Charge™ 3.0 technology
- Qualcomm® Quick Charge™ 2.0 technology
- Qualcomm® Quick Charge™ 1.0 technology
- Qualcomm® Quick Charge™ technology

**References**
ARM

Processor classes

Cortex-M
Targets SoC

Cortex-R
Real-time

Cortex-A
Application processors

Accelerators and special features

Mali GPU
ARM GPU

Trillium
Machine learning

SecurCore
Tamper proof, security
Field Programmable Gate Arrays (FPGAs)

- PicoZed
  - Xilinx Zynq-7000 SoC

- Xilinx Virtex
  - UltraScale+ HBM VCU128-ES1

- BittWare TeraBox
  - 8 FPGA cards, 2 TB DDR4

- Intel Arria 10 GX
  - 1,150K logic elements

- 24 TeraFLOPS processing: 16x Altera Arria 10 or Stratix V FPGAs
  - Up to 18 million logic elements (Arria 10 GX)
  - Up to 62,000 multipliers (Stratix V GS)

- 1.28 Terabits/sec I/O
- 128x 10GigE, 32x 40GigE, 32x 100 GigE, or 32x QDR Infiniband

- 6.5 Terabits/sec memory bandwidth
  - Up to 64 banks DDR3-1600 (512 GBytes)
  - DDR4, QDRII+, and RLDRAM3 memory options

- 4U or 5U Rackmount PCIe system (server, industrial, or expansion)
  - Dual socket Intel Ivy Bridge with up to 12 cores
  - Up to 768 GBytes of system memory
  - 8 Gen3 x16 PCIe slots

Complete software support
- Windows and Linux 64 drivers, interface libraries, and hardware management
- FPGA development kit for Arria 10 and Stratix V
Memory and Packaging

Memory chips

- DDR4 DIMM
  - Standard memory
- SO-DIMM
  - Laptops
- GDDR5
  - For GPUs
- HMC
  - 3D memory on PCB
- HBM2
  - For SoC/SiP

Integration

- Printed circuit board (PCB)
  - Traditional circuit board
- Multi chip packaging (MCP)
  - Tighter integration
- Silicon interposer
  - Chip-on-chip integration
Bulk Storage and Busses

Bulk storage (disks)

- Hard drive
  - Standard disk
- SSD
  - Flash as disk drive
- NVMe
  - HDD on PCIe
- SD card
  - Memory for cell phone etc

Extension busses

- PCIe x1, x4, x8, x16
  - Desktop/laptop standard
- NVLink
  - Fast GPU interconnect
- Coherent Accelerator Processor Interface (CAPI)
  - IBM/server class open standard
Outline

- Multicores, Manycores, accelerators
- Experimental accelerators
- Latest Nvidia GPUs
- CUDA
- Summary

*Based on “15-740 Computer Architecture” by Nathan Beckmann*
Example 1: DianNao

Figure 11. Accelerator.

Figure 1. Neural network hierarchy containing convolutional, pooling and classifier layers.

DianNao [Chen et al, ASPLOS’14, Best paper]
DaDianNao [Chen et al, MICRO’14, Best paper]
PuDianNao [Liu et al, ASPLOS’15]
ShiDianNao [Du et al, ISCA’15]
Example 2: EIE: Sparse Neural Networks

EIE: Efficient Inference Engine

[Han et al, ISCA’16]
Example 3: Graphicianado

GraphMat Processing Model

1. For each Vertex V
2. For each incoming edge E(U,V) from active vertex U
3. \( \text{Res} \leftarrow \text{Process} \_ \text{Edge} \left( E_{\text{weight}}, U_{\text{prop}}, \{ \text{optional} \} V_{\text{prop}} \right) \)
4. \( V_{\text{temp}} \leftarrow \text{Reduce} \left( V_{\text{temp}}, \text{Res} \right) \)
5. End
6. End
7. For each Vertex V,
8. \( V_{\text{prop}} \leftarrow \text{Apply} \left( V_{\text{temp}}, V_{\text{prop}}, V_{\text{const}} \right) \)
9. End

[Ham et al, MICRO’16, Best paper]
Example 4: Plasticine

Pattern based reconfigurable computing

[Prabhakar et al, ISCA’17]
Outline

- Multicores, Manycores, accelerators
- Experimental accelerators
- Latest Nvidia GPUs
- CUDA
- Summary
Volta GV100 with 84 SM Units
Volta GV100 Streaming Multiprocessor (SM)
Tensor Cores

\[ D = \]

FP16 or FP32

\[ \begin{align*}
A_{0,0} & & A_{0,1} & & A_{0,2} & & A_{0,3} \\
A_{1,0} & & A_{1,1} & & A_{1,2} & & A_{1,3} \\
A_{2,0} & & A_{2,1} & & A_{2,2} & & A_{2,3} \\
A_{3,0} & & A_{3,1} & & A_{3,2} & & A_{3,3}
\end{align*} \]

FP16

\[ \begin{align*}
B_{0,0} & & B_{0,1} & & B_{0,2} & & B_{0,3} \\
B_{1,0} & & B_{1,1} & & B_{1,2} & & B_{1,3} \\
B_{2,0} & & B_{2,1} & & B_{2,2} & & B_{2,3} \\
B_{3,0} & & B_{3,1} & & B_{3,2} & & B_{3,3}
\end{align*} \]

FP16

\[ \begin{align*}
C_{0,0} & & C_{0,1} & & C_{0,2} & & C_{0,3} \\
C_{1,0} & & C_{1,1} & & C_{1,2} & & C_{1,3} \\
C_{2,0} & & C_{2,1} & & C_{2,2} & & C_{2,3} \\
C_{3,0} & & C_{3,1} & & C_{3,2} & & C_{3,3}
\end{align*} \]

FP16 or FP32

Full precision product

Sum with FP32 accumulator

Convert to FP32 result

FP16 storage/input

\[ \times \]

more products

\[ + \]

F32

F32
## Volta GV100 with 84 SM Units

<table>
<thead>
<tr>
<th>Tesla Product</th>
<th>Tesla K40</th>
<th>Tesla M40</th>
<th>Tesla P100</th>
<th>Tesla V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>GK180 (Kepler)</td>
<td>GM200 (Maxwell)</td>
<td>GP100 (Pascal)</td>
<td>GV100 (Volta)</td>
</tr>
<tr>
<td>SMs</td>
<td>15</td>
<td>24</td>
<td>56</td>
<td>80</td>
</tr>
<tr>
<td>TPCs</td>
<td>15</td>
<td>24</td>
<td>28</td>
<td>40</td>
</tr>
<tr>
<td>FP32 Cores / SM</td>
<td>192</td>
<td>128</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>FP32 Cores / GPU</td>
<td>2880</td>
<td>3072</td>
<td>3584</td>
<td>5120</td>
</tr>
<tr>
<td>FP64 Cores / SM</td>
<td>64</td>
<td>4</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>FP64 Cores / GPU</td>
<td>960</td>
<td>96</td>
<td>1792</td>
<td>2560</td>
</tr>
<tr>
<td>Tensor Cores / SM</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>8</td>
</tr>
<tr>
<td>Tensor Cores / GPU</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>640</td>
</tr>
<tr>
<td>GPU Boost Clock</td>
<td>810/875 MHz</td>
<td>1114 MHz</td>
<td>1480 MHz</td>
<td>1530 MHz</td>
</tr>
<tr>
<td>Peak FP32 TFLOPS(^1)</td>
<td>5</td>
<td>6.8</td>
<td>10.6</td>
<td>15.7</td>
</tr>
<tr>
<td>Peak FP64 TFLOPS(^1)</td>
<td>1.7</td>
<td>.21</td>
<td>5.3</td>
<td>7.8</td>
</tr>
<tr>
<td>Peak Tensor TFLOPS(^1)</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>125</td>
</tr>
<tr>
<td>Texture Units</td>
<td>240</td>
<td>192</td>
<td>224</td>
<td>320</td>
</tr>
<tr>
<td>Memory Interface</td>
<td>384-bit GDDR5</td>
<td>384-bit GDDR5</td>
<td>4096-bit HBM2</td>
<td>4096-bit HBM2</td>
</tr>
<tr>
<td>Memory Size</td>
<td>Up to 12 GB</td>
<td>Up to 24 GB</td>
<td>16 GB</td>
<td>16 GB</td>
</tr>
<tr>
<td>L2 Cache Size</td>
<td>1536 KB</td>
<td>3072 KB</td>
<td>4096 KB</td>
<td>6144 KB</td>
</tr>
<tr>
<td>Shared Memory Size / SM</td>
<td>16 KB/32 KB/48 KB</td>
<td>96 KB</td>
<td>64 KB</td>
<td>Configurable up to 96 KB</td>
</tr>
<tr>
<td>Register File Size / SM</td>
<td>256 KB</td>
<td>256 KB</td>
<td>256 KB</td>
<td>256 KB</td>
</tr>
<tr>
<td>Register File Size / GPU</td>
<td>3840 KB</td>
<td>6144 KB</td>
<td>14336 KB</td>
<td>20480 KB</td>
</tr>
<tr>
<td>TDP</td>
<td>235 Watts</td>
<td>250 Watts</td>
<td>300 Watts</td>
<td>300 Watts</td>
</tr>
<tr>
<td>Transistors</td>
<td>7.1 billion</td>
<td>8 billion</td>
<td>15.3 billion</td>
<td>21.1 billion</td>
</tr>
<tr>
<td>GPU Die Size</td>
<td>551 mm(^2)</td>
<td>601 mm(^2)</td>
<td>610 mm(^2)</td>
<td>815 mm(^2)</td>
</tr>
<tr>
<td>Manufacturing Process</td>
<td>28 nm</td>
<td>28 nm</td>
<td>16 nm FinFET(^+)</td>
<td>12 nm FFN</td>
</tr>
</tbody>
</table>

\(^1\) Peak TFLOPS rates are based on GPU Boost Clock
Multi-GPU Systems: NVLink
Thread Divergence

Pre-Volta

Program Counter (PC) and Stack (S)

32 thread warp

Volta

Convergence Optimizer

32 thread warp with independent scheduling

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
__syncwarp()
```
Outline

- Multicores, Manycores, accelerators
- Experimental accelerators
- Latest Nvidia GPUs
- CUDA
- Summary

Based on “15-418/15-618: Parallel Computer Architecture and Programming” by Randy Bryant and Nathan Beckmann
GPU Architecture

- Multi-core chip
- SIMD execution within a single core (many execution units performing the same instruction)
- Multi-threaded execution on a single core (multiple threads executed concurrently by a core)
NVIDIA Tesla architecture (2007)

- (GeForce 8xxx series GPUs)
  First alternative, non-graphics-specific (“compute mode”) interface to GPU hardware

- Lets say a user wants to run a non-graphics program on the GPU’s programmable cores...
  - Application can allocate buffers in GPU memory and copy data to/from buffers
  - Application (via graphics driver) provides GPU a single kernel program binary
  - Application tells GPU to run the kernel in an SPMD fashion (“run N instances”)
  - Go! (launch(myKernel, N))
CUDA Programming Language

- Introduced in 2007 with NVIDIA Tesla architecture
- “C-like” language to express programs that run on GPUs using the compute-mode hardware interface
- Relatively low-level: CUDA’s abstractions closely match the capabilities/performance characteristics of modern GPUs (design goal: maintain low abstraction distance)

Note: OpenCL is an open standards version of CUDA

- CUDA only runs on NVIDIA GPUs
- OpenCL runs on CPUs and GPUs from many vendors
- Almost everything we say about CUDA also holds for OpenCL
Basic CUDA Syntax

- **“Host” code:** serial execution  
  Running as part of normal C/C++ application on CPU

- **Bulk launch of many CUDA threads:**  
  “launch a grid of CUDA thread blocks”  
  Call returns when all threads have terminated

- **SPMD execution of device kernel function:**

- **“CUDA device” code:** kernel function  
  (`__global__` denotes a CUDA kernel function) runs on GPU

- Each thread computes its overall grid thread id from its position in its block (`threadIdx`) and its block’s position in the grid (`blockIdx`)
Clear Separation of Host and Device Code

- Separation of execution into host and device code is performed statically by the programmer

```
const int Nx = 12;
const int Ny = 6;
dim3 threadsPerBlock(4, 3, 1);
dim3 numBlocks(Nx/threadsPerBlock.x,
                Ny/threadsPerBlock.y, 1);

// assume A, B, C are allocated Nx x Ny float arrays
// this call will cause execution of 72 threads
// 6 blocks of 12 threads each
matrixAddDoubleB<<<numBlocks, threadsPerBlock>>>(A, B, C);
```

```
/device_ float doubleValue(float x)
{
    return 2 * x;
}

// kernel definition
global_ void matrixAddDoubleB(float A[Ny][Nx],
                               float B[Ny][Nx],
                               float C[Ny][Nx])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;

    C[j][i] = A[j][i] + doubleValue(B[j][i]);
}
```

“Host” code: serial execution on CPU

“Device” code (SPMD execution on GPU)
Number of SPMD Threads is Explicit in Program

- Number of kernel invocations is not determined by size of data collection (a kernel launch is not map(kernel, collection) as was the case with graphics shader programming)

Regular application thread running on CPU (the "host")

```c
const int Nx = 11; // not a multiple of threadsPerBlock.x
const int Ny = 5;  // not a multiple of threadsPerBlock.y

dim3 threadsPerBlock(4, 3, 1);
dim3 numBlocks((Nx+threadsPerBlock.x-1)/threadsPerBlock.x,
                (Ny+threadsPerBlock.y-1)/threadsPerBlock.y, 1);

// assume A, B, C are allocated Nx x Ny float arrays

// this call will cause execution of 72 threads
// 6 blocks of 12 threads each
matrixAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
```

CUDA kernel definition

```c
__global__ void matrixAdd(float A[Ny][Nx],
                          float B[Ny][Nx],
                          float C[Ny][Nx])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;

    // guard against out of bounds array access
    if (i < Nx && j < Ny)
        C[j][i] = A[j][i] + B[j][i];
}
```
CUDA Memory Model

- Distinct host and device address spaces
memcpy Primitive

- Move data between address spaces

```c
float* A = new float[N];  // allocate buffer in host mem

// populate host address space pointer A
for (int i=0; i<N; i++)
    A[i] = (float)i;

int bytes = sizeof(float) * N
float* deviceA;  // allocate buffer in
cudaMalloc(&deviceA, bytes);  // device address space

// populate deviceA
cudaMemcpy(deviceA, A, bytes, cudaMemcpyHostToDevice);

// note: deviceA[i] is an invalid operation here (cannot
// manipulate contents of deviceA directly from host.
// Only from device code.)
```
CUDA device Memory Model

- Three distinct types of memory visible to kernels

Programmer has direct control over memory hierarchy
CUDA Example: 1D Convolution

\[
\text{output}[i] = (\text{input}[i] + \text{input}[i+1] + \text{input}[i+2]) / 3.f;
\]
1D Convolution in CUDA

One thread per output element

![Diagram showing convolution process with threads]

CUDA Kernel

```c
#define THREADS_PER_BLK 128
__global__ void convolve(int N, float* input, float* output) {
    int index = blockIdx.x * blockDim.x + threadIdx.x; // thread local variable

    float result = 0.0f; // thread-local variable
    for (int i=0; i<3; i++)
        result += input[index + i];

    output[index] = result / 3.f;
}
```

Host code

```c
int N = 1024 * 1024
cudaMalloc(&devInput, sizeof(float) * (N+2)); // allocate array in device memory
cudaMalloc(&devOutput, sizeof(float) * N); // allocate array in device memory

// Initialize contents of devInput here...
convolve<<<N/THREADS_PER_BLK, THREADS_PER_BLK>>>(N, devInput, devOutput);
```
CUDA Synchronization Constructs

- `__syncthreads()`
  Barrier: wait for all threads in the block to arrive at this point

- Atomic operations
  e.g., `float atomicAdd(float* addr, float amount)`
  Atomic operations on both global memory and shared memory variables

- Host/device synchronization
  Implicit barrier across all threads at return of kernel
CUDA Abstractions

- **Execution: thread hierarchy**
  - Bulk launch of many threads
  - Two-level hierarchy: threads are grouped into thread blocks

- **Distributed address space**
  - Built-in memcpy primitives to copy between host and device address spaces
  - Three different types of device address spaces
  - Per thread, per block (“shared”), or per program (“global”)

- **Barrier synchronization primitive for threads in thread block**

- **Atomic primitives for additional synchronization**
  shared and global variables
Summary

- Multicores, Manycores, accelerators
- Experimental accelerators
- Latest Nvidia GPUs
- CUDA