ECEN 5623
Wrap-Up of RAID / Lab-3
&
Introduction to Vector Instructions and Co-Processors and Lab-4

Lecture 8

RAID6 Review

- RAID-6 is a Double Parity, Double Fault in a Single-Set Safe Data Protection Scheme
  - Reed-Solomon (Galois Field) RAID-6
    - P,Q – P is Simple XOR Parity, Q is Galois Field Encoded
    - Q Encoding and Rebuilds Algorithmically Complex
  - Even/Odd RAID-6
    - Simpler, Patent Protected
  - RAID-6 DP
    - Simpler, Patent Protected
  - RAID-6 Liberation Codes
    - http://www.usenix.org/events/fast08/tech/full_papers/plank/plank.pdf
    - Patent Free, Similar Performance and Simplicity of DP and Even/Odd

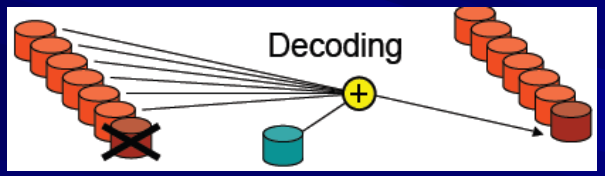
Industry Trends with RAID

- Solid-State Disk
  - RAM
  - Flash (SLC, MLC – Single/Multi Layer Cell)
  - Emergent Technologies (Phase-Change, Race-Track, Other)
    - http://www.numonyx.com/en-US/MemoryProducts/PCM/Pages/PCM.aspx
    - http://www.youtube.com/watch?v=dJf3z9AfiVM
- Why is Tape and Spinning Disk Still Around?
  - Cost: $/GB
  - Cost: $/IO /sec
  - Tiering Down or Up Often the Best Solution
  - Data Has A Life: On-Line, Near-Line, Off-Line, Deep Archive
  - http://download.intel.com/design/flash/nand/extreme/ITJ_03_SSDs_for_Embedded.pdf

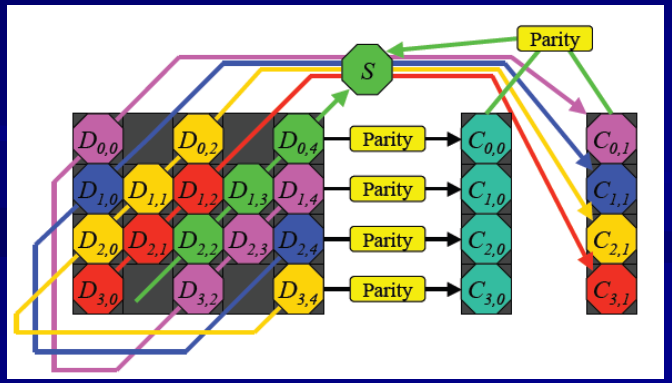
RAID6 and Erasure Codes

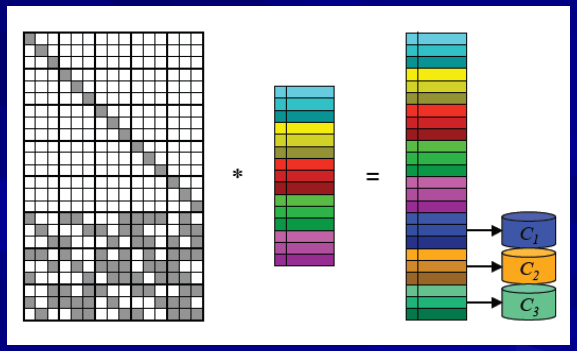
- **Numerous RAID6 Implementations Today**
- **Double Fault Protection**
- **General MDS and non-MDS Erasure Codes Provide Triple Fault and Beyond**

RAID5 (MDS=1)



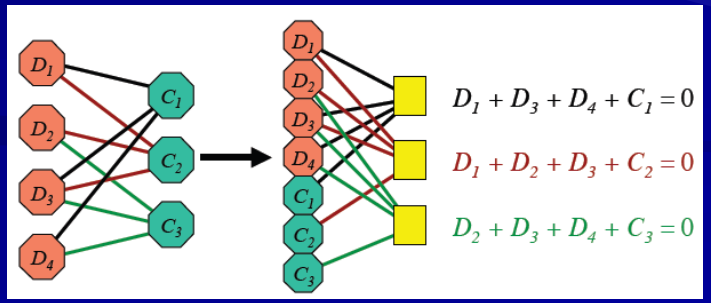
Dual XOR, Even/Odd (MDS=2)





Cauchy Reed-Solomon

LDPC (not MDS)



MTTDL for Erasure Codes

- **RAID1 MTTDL (MDS=1, R=0.5)**
  - **Exponential Distribution of Failure & Repair Time**
    - **One Drive failure per 1000 hours (month) for population of 1000 drives**
    - P(t)=(1-e^{-kt}), MTTF=1/k, so MTTF=10^6 means k=10^{-6}, and P(1000000)=0.63
  - **MTTDL = Double_Fault_Time / Total_Repair_Time**
    - N Possible First Faults
    - N-1 Possible Second Faults During Repair Window
    - 1 out of N-1 Second Faults Causes Data Loss
    - **Large Repair Times for a population N is a growing problem!**
  - **$MTTF^2$ is time window for double fault occurrence over population N**
  - **N*MTTR is total repair time for population N**

$$MTTDL_{RAID\,1} = \frac{MTTF^2}{N * MTTR}$$

- **RAID5 MTTDL (MDS=1, R=0.75 for 3+1)**
  - Where MTTF is time to failure for a member disk
  - MTTR = time to replace and rebuild data/parity
  - N = number of devices
  - Data loss for 2nd device failure before rebuild is complete for set, degraded operation requires restoration and data rebuild on the fly for on-going IO

$$MTTDL_{RAID\,5} = \frac{MTTF^2}{(N+1) * N * MTTR}$$

- **RAID6 MTTDL (MDS=2, R=0.6 for 3+2)**
  - Triple Fault window is huge ($MTTF^3$)
  - Repair window grows too

$$MTTDL_{RAID\,6} = \frac{MTTF^3}{(N+2) * (N+1) * N * MTTR^2}$$

The Annual Probability of Data Loss

$$P(t)_{failure} = (1 - e^{-kt}), k = 1/MTTF$$

$$P(t)_{data\_loss} = (1 - e^{-\left[\frac{1}{(MTTDL_{set})} * Lifetime\right]}) * N_{sets}, k = 1/MTTDL_{set}$$

Birth/Death Exponential Model

- P(t)=Probability of Failure over time period
- k=probability of failure, t=time
- MTTF = Mean Time To Failure (or Between)
- N=Population, Lifetime=e.g. Annual
- MTTDL=Mean Time to Data Loss in Population

How good is it?

- Optimistic – Only 2 states
- Pessimistic – Does not account for proactive data protection measures
- Does model resiliency to drive failures

MTTDL RAID Examples

- Potential for Data Loss

$$MTTDL = \frac{Time\_to\_Muiltple\_Concurrent\_Failures}{Number\_of\_Data\_Loss\_Combinations * (Exposure\_Window)}$$

- RAID1 Equation
  - Joint probability of 2 erasures
  - Coupling of mirror drives
  - Exposure window

$$MTTDL_{RAID1} = \frac{MTTF^2}{N * MTTR}$$

$$MTTDL_{RAID1,3-way} = \frac{MTTF^3}{(N-1) * N * MTTR^2}$$

- RAID5 Equation
  - (N+1)*N Double Fault Scenarios

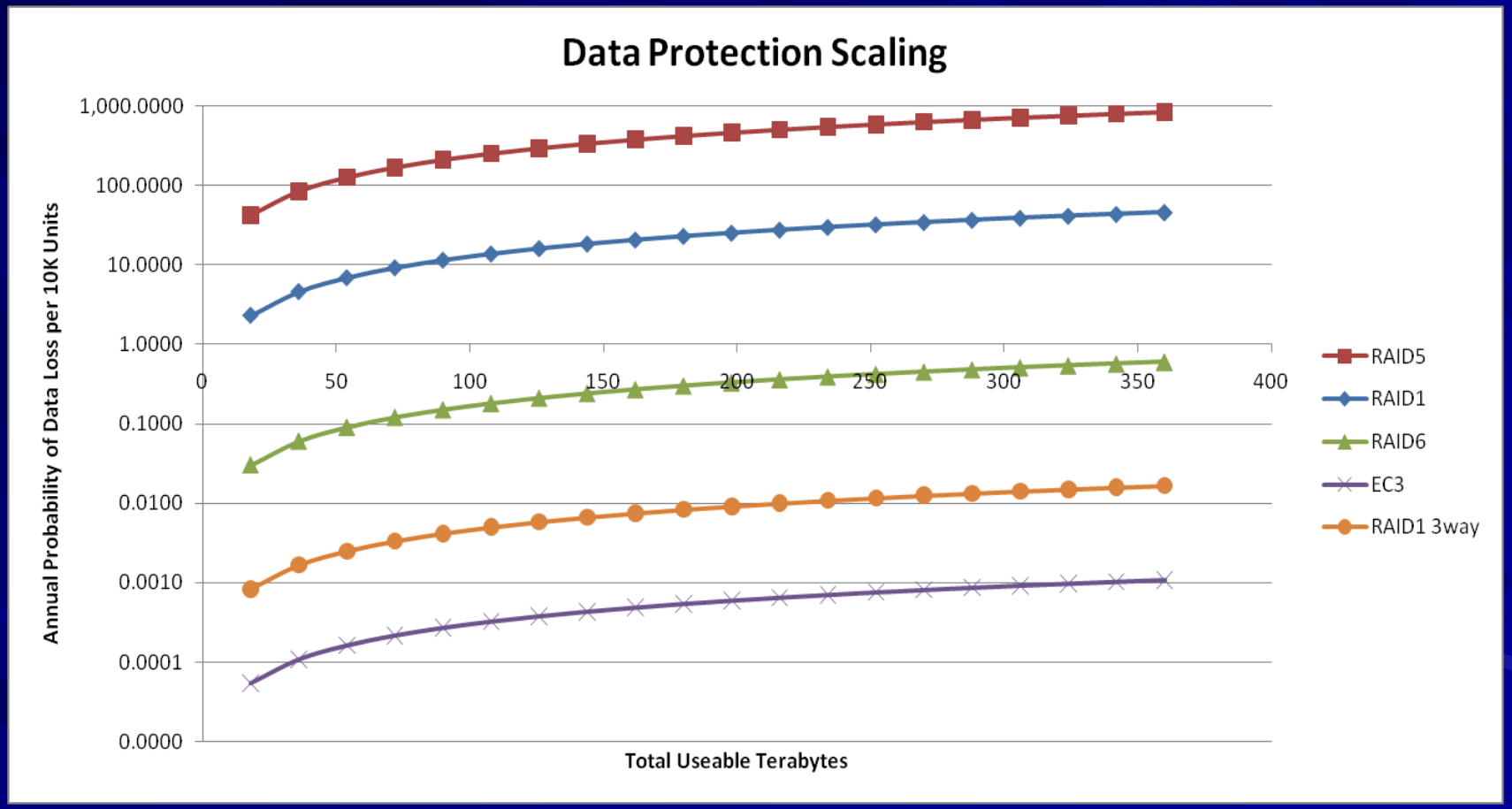
$$MTTDL_{RAID5} = \frac{MTTF^2}{(N+1) * N * MTTR}$$

- RAID6 Equation
  - Joint probability 3 erasures
  - (N+2)*(N+1)*N Triple Fault Scenarios

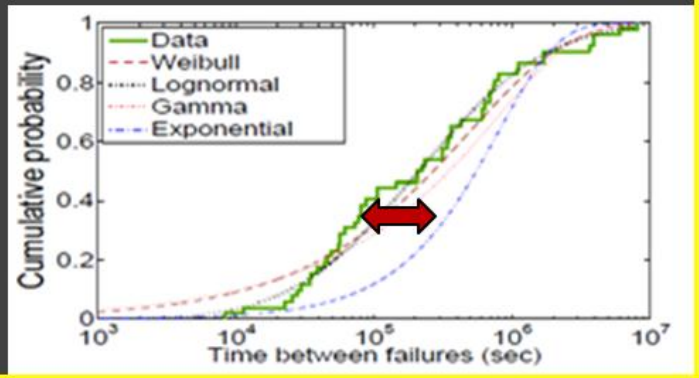
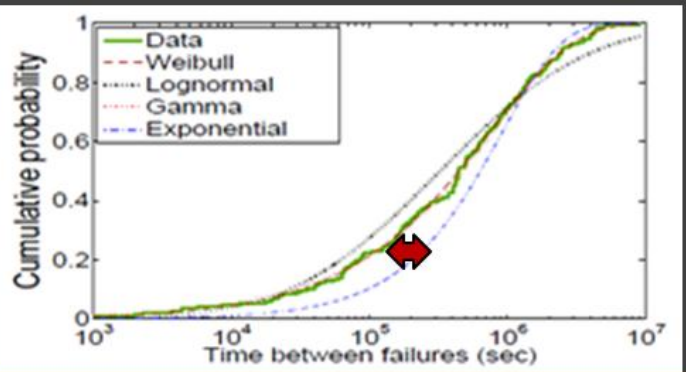
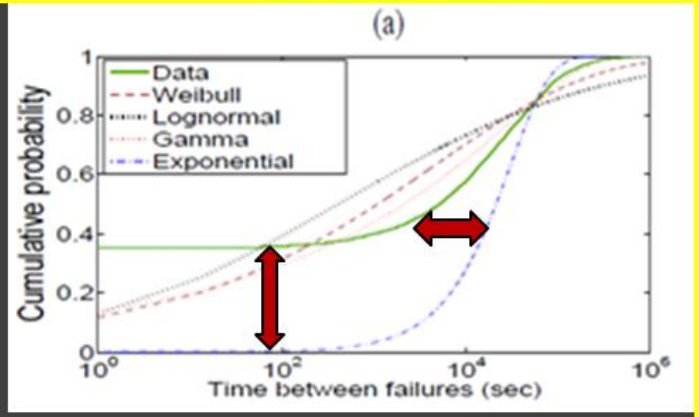
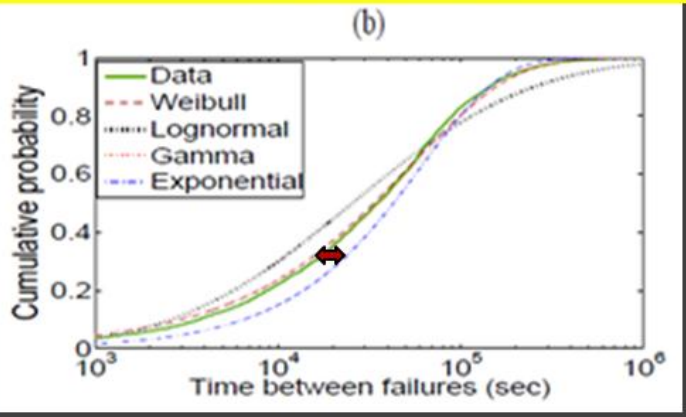
$$MTTDL_{RAID6} = \frac{MTTF^3}{(N+2) * (N+1) * N * MTTR^2}$$

Scaling With Virtualization

Keeps Sets Independent, Constant Overhead, Virtual Mapping of RAID Sets over Nodes/Drives



Comparison to Statistics

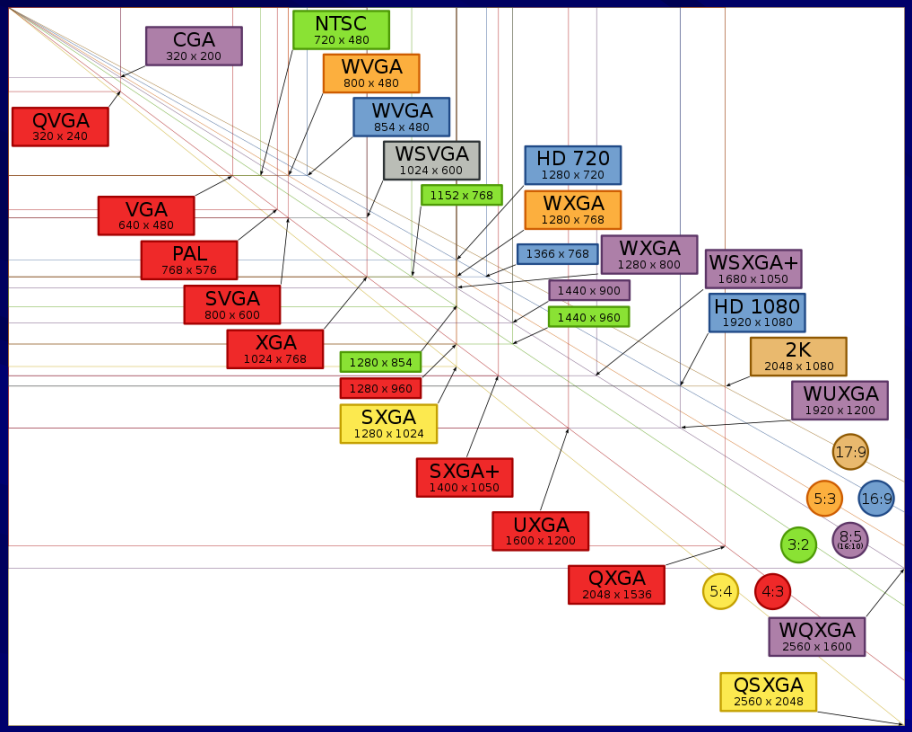
	Infant Mortality	Mid to End of Life
Node Level		
System Level		

A large-scale study of failures in high-performance computing systems", Bianca Schroeder, Garth A. Gibson

Using SIMD and GPU Vector Processing for Digital Media

Introduction

FRAME: XY Pixel Maps



http://en.wikipedia.org/wiki/Display_resolution

- FRAME Resolution
  - Computer Graphics Resolutions (Close in Viewing)
    - VGA = 640x480
    - SVGA=800x600
  - TV and Cinema Resolutions (Lean Back Viewing)
  - NTSC = Standard Defintion, 720x480 Interlaced (Odd, then Even Scan Lines)
  - High Definition (Progressive full frame or Interlaced)
    - 720i/p = 1280x720
    - 1080i/p = 1920x1080
- FRAME Aspect Ratios
  - X to Y Ratio
  - NTSC = 3:2, 720x480 (240)
  - HD 720 = 16:9, 1280x720 (80)
  - HD 1080 = 16:9, 1920x1080 (120)
  - 2K = 17:9, 2048x1080 (120)
- FRAME Rates
  - 60i, NTSC = 59.94 odd/even, or 29.97 (Basically 30 fps, 60x1000/1001 – RF Chroma/Audio Seperation)
  - 24p, Cinema = 24 fps
  - 60p, HDTV Progressive Modes

DM Frame Transformation Challenges

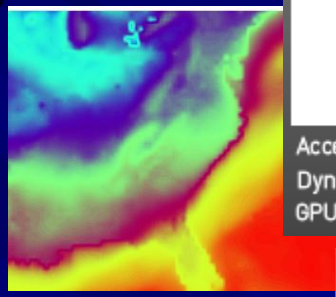
- Any Operation over HD Frame is Costly
  - At 60, 30, 24 Frames per Second
    - NTSC = 3:2, 720x480 = 345,600 Operations
    - HD 720 = 16:9, 1280x720 = 921,600 Operations
    - HD 1080 = 16:9, 1920x1080 = 2,073,600 Operations
    - 2K = 17:9, 2048x1080 = 2,097,152 Operations
- Parallel Operations on Pixel Slices (Vectors)
- Concurrent Operations on Macro Blocks

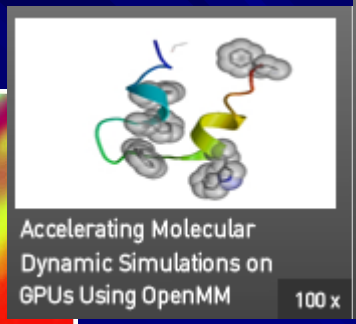
Potential Scaling and Speed-Up

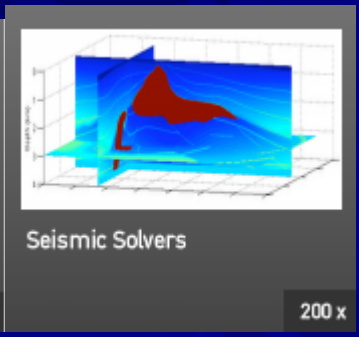
- Clusters (Infiniband, gigE/10GE)
- NUMA Many-Core Scaling (e.g. QPI and HyperTransport-3)
- Vector Instructions (e.g. Multiply and Accumulate, XOR, etc.)
- Co-Processors (Offload/Acceleration Engines)
- Vector Co-Processors (GPU)
- General Purpose Vector Co-Processors (GP-GPU)
- Purpose-Built FPGA Co-Processors

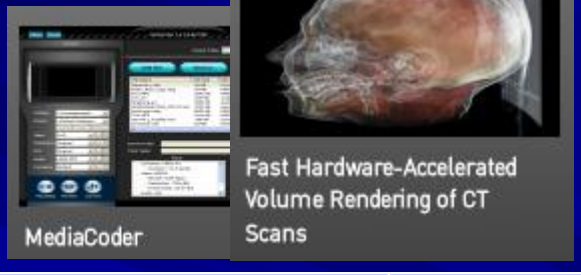
GP-GPU, What Is It?

- Ideal for Large Bitwise, Integer, and Floating Point Vector Math
- Flynn's Taxonomy
- SIMD Architecture often Co-Processor for MIMD









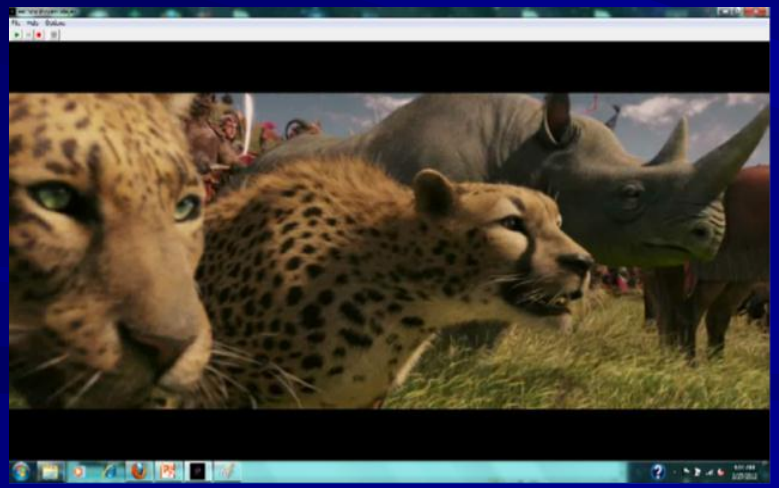
	Single Instruction/Program	Multiple Instruction
Single Data	**SISD** (Traditional Uni-processor)	**MISD** (Voting schemes and active-active controllers)
Multiple Data	**SPMD** (e.g. CUDA GP-GPU), **SIMD** (SSE 4x Vector Processing)	**MIMD** (Distributed systems (MPMD), Clusters with MPI/PVM (SPMD), AMP/SMP)

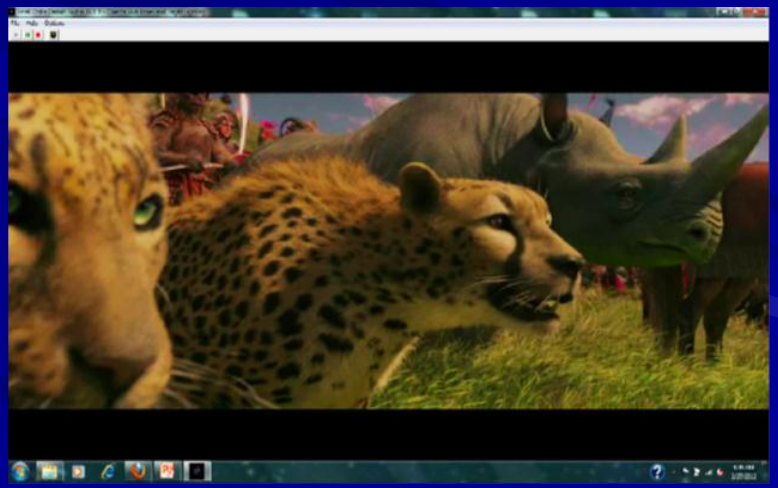
SIMD Vector Instructions

- Intel MMX, SSE 1, 2, 3, 4.x Code Generation
- Using SIMD Extensions to Accelerate Algorithms (Edge Enhancement)
  - http://software.intel.com/en-us/articles/using-intel-streaming-simd-extensions-and-intel-integrated-performance-primitives-to-accelerate-algorithms/

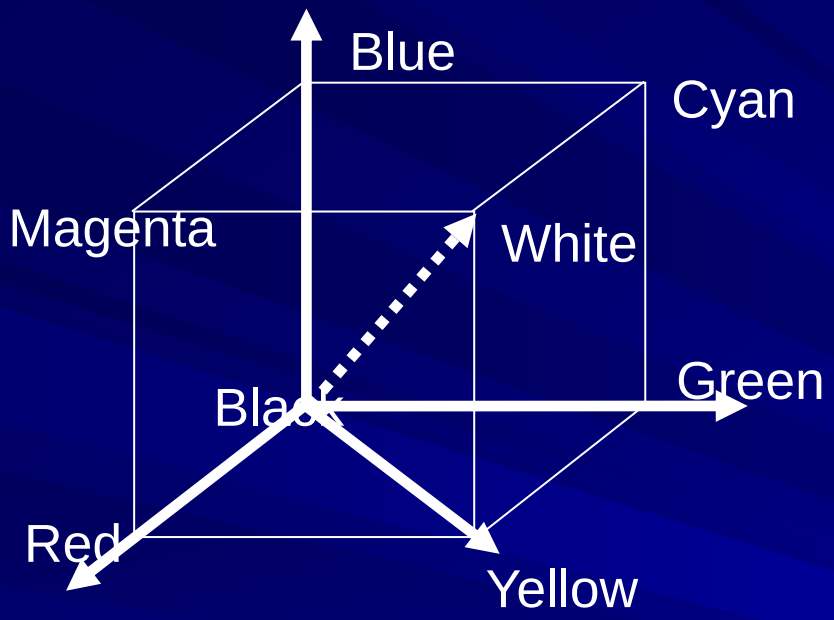
RT Frame Transformation Examples (Beyond Sharpen)

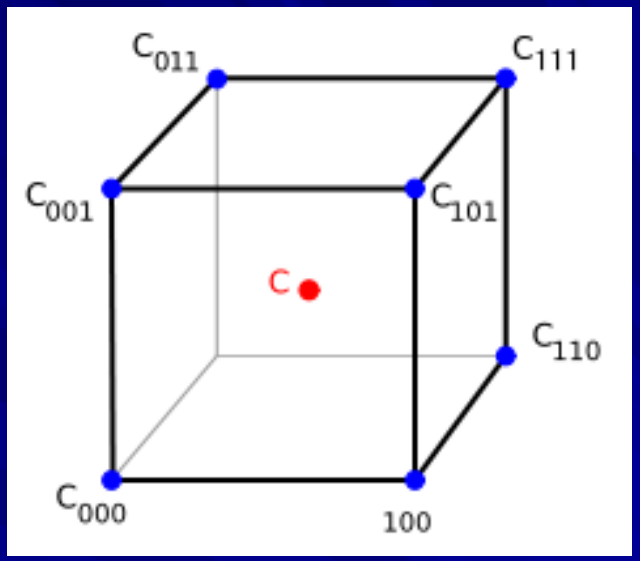
- Color Editing and Boosting
- Gamut Control for Range of Background Lighting Conditions
- Pixel Input (sRGB) ➔ New Tri-linear Interpolated Pixel (sRGB)





Tri-Linear Interpolation

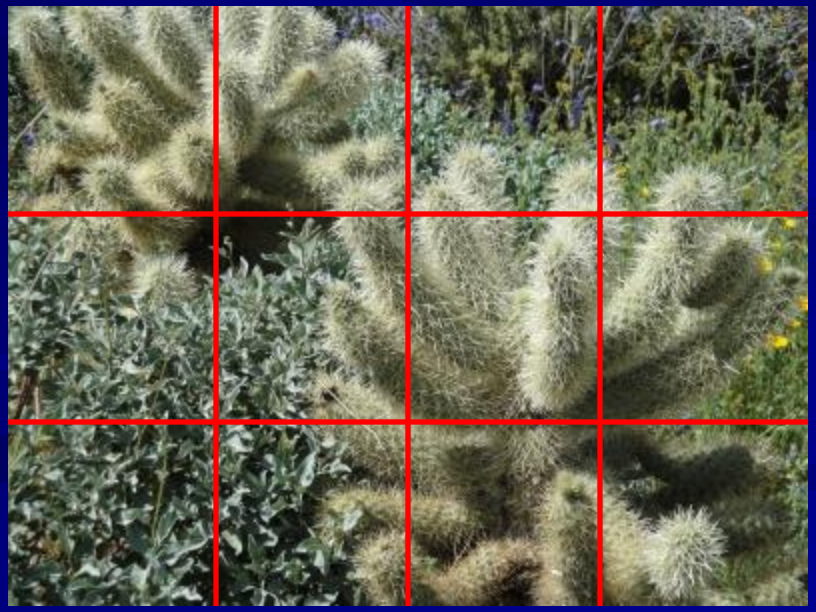




- sRGB LUT is 256x256x256 = 16MB of 24-bit Pixels, or a 50MB Table!
- Store say 16x16x16 and Interpolate Remaing Pixels
- http://www.grc.nasa.gov/WWW/winddocs/utilities/b4wind_guide/trilinear.html

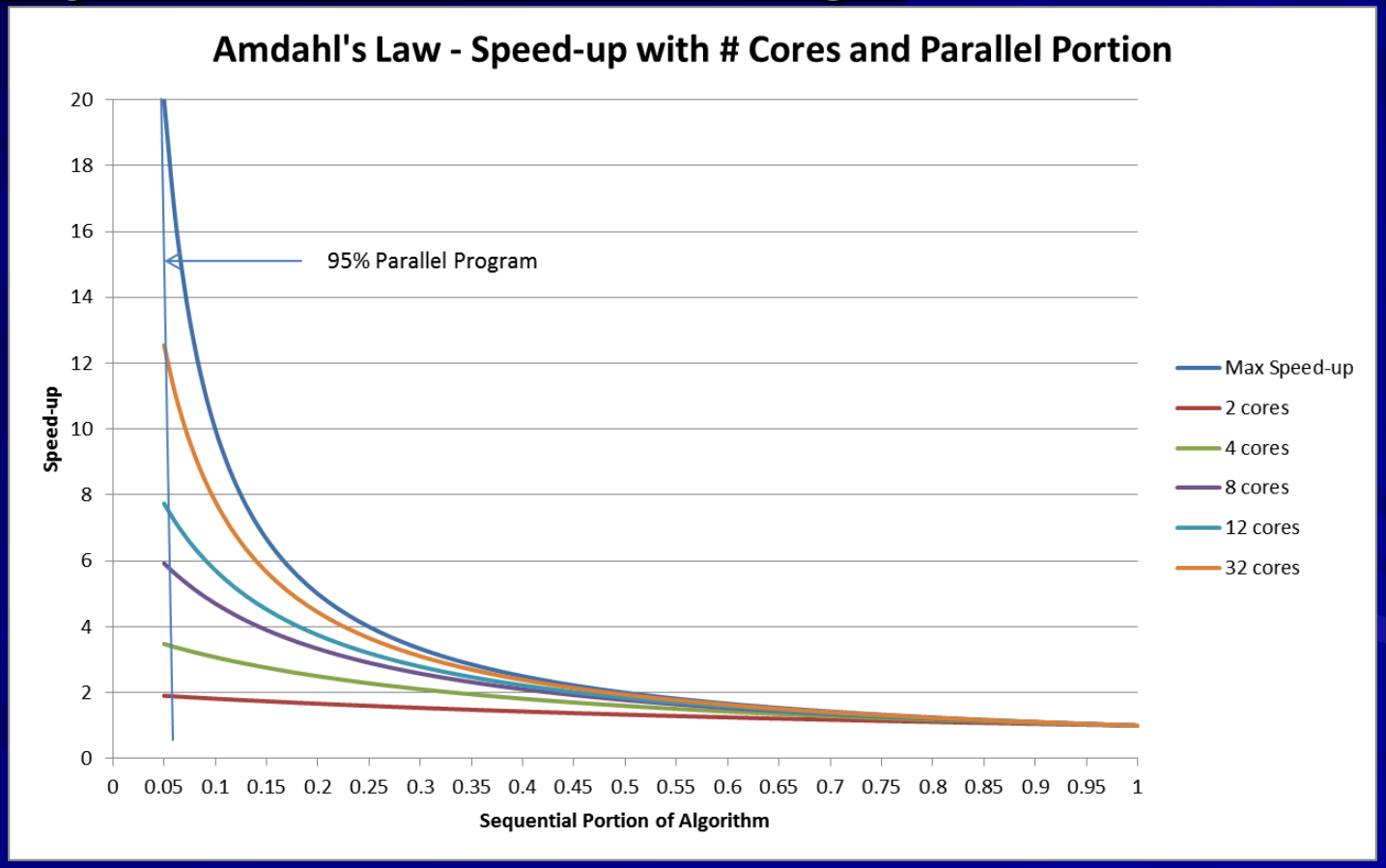
Let's Look at Some Code

- Threading and Image Segmentation for Many-Core and GPGPU
  - Example to be Published in IBM Paper on 1/24
  - Example Code Used in First Lab – Threading!!
  - Grids for Threading, Step One toward GPU Acceleration
- Examples of Video/Still-Frame Enhancement



Embarasingly Parallel

- www.ibm.com/developerworks/industry/library/ind-cloud-e-learning2



Far Less Simple to Speed-Up

- Any Algorithm with Data Dependencies Requiring Locks for Correctness
- E.g. FFT
- Prime Number Searches
- Schedule and Map to Minimize Locks Taken
- Will Serialize Far More Often

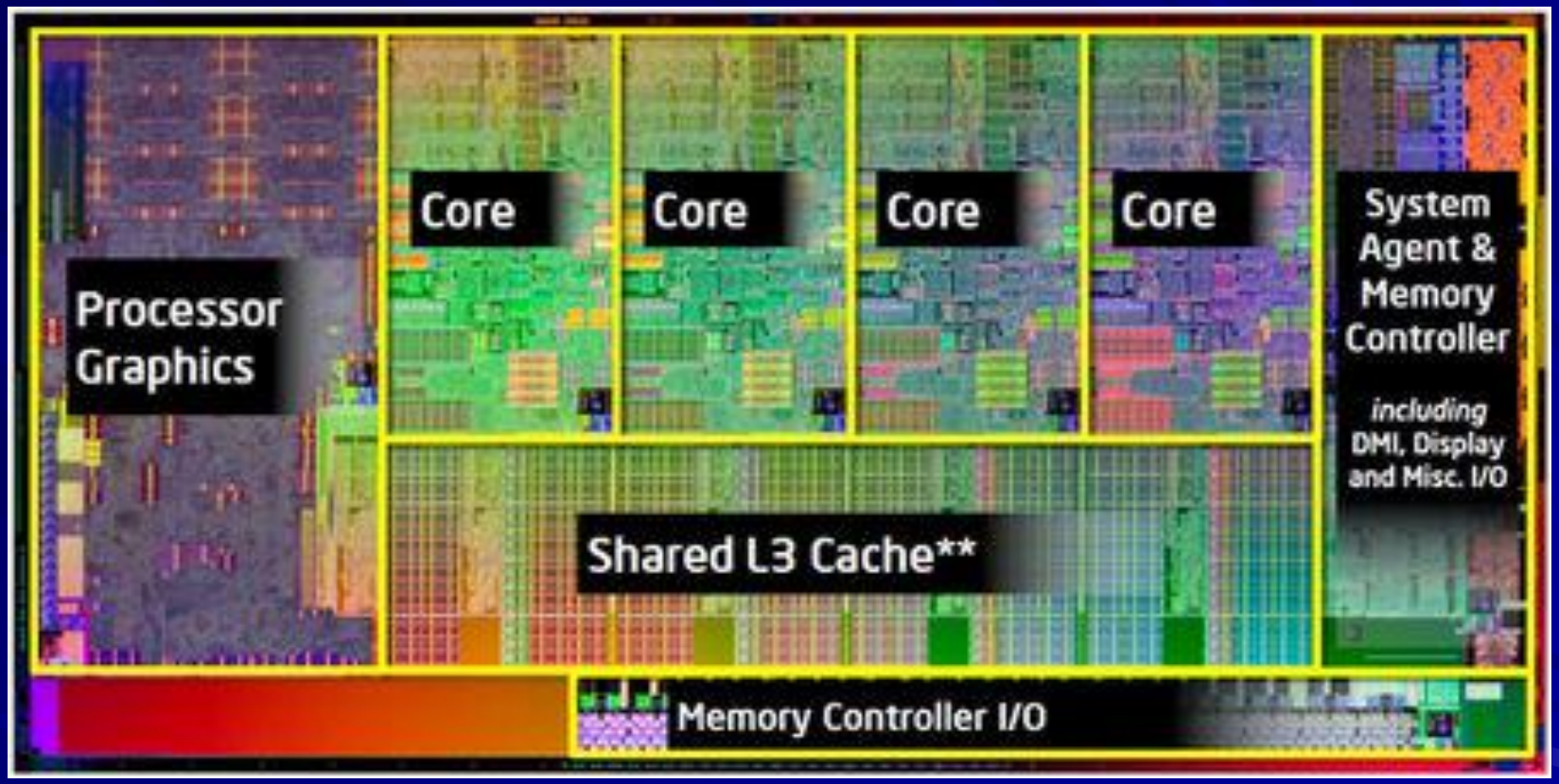
Other Algorithms

- Edge Detection
- Background Substitution (Green Screen)
  - Remove Green Screen Pixels
  - Take Arbitrary Background
  - Stitch in Foreground over New Background Rastering Image
- Resolution Change
- Almost Any Other Image Processing Algorithm (e.g.Segmentation, Adding Text Subtitles, etc.)

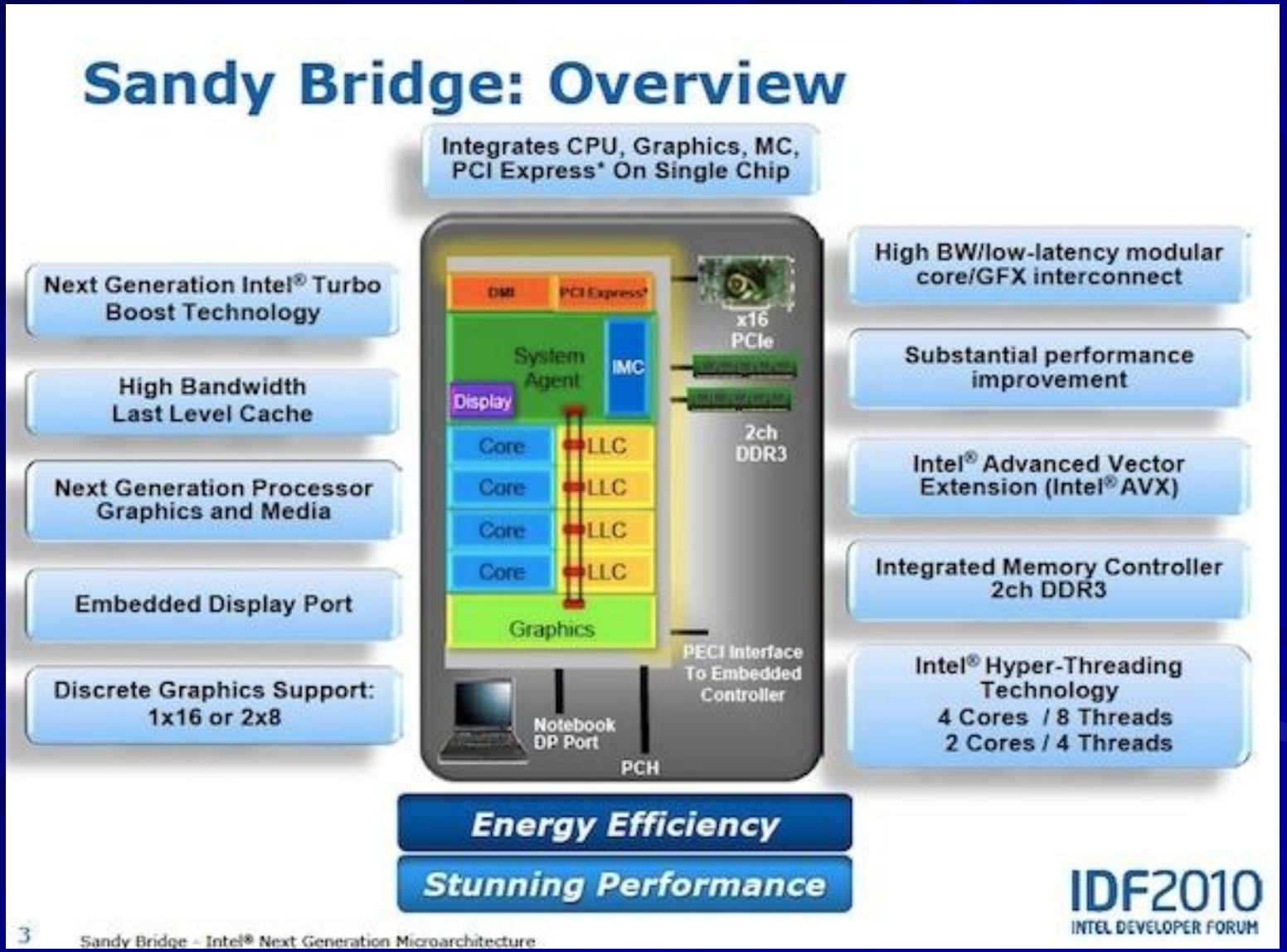
Scale-Up Architectures

- Many Core (MIMD), GP-GPU (SIMD, SPMD), Vector Processing ISEs (SIMD)
  - MIMD and SPMD
    - E.g. Cell BBE (1 SMT PowerPC core + 8 SPEs)
    - Intel x86_64 Plus GPU or GP-GPU
  - MIMD with SIMD
    - E.g. x86 with SSE 4.x Instruction Set Extension
- Advnatages of Co-Processor s vs. Instruction Set Extension or Integrated Graphics Engine
  - IO Bus Bandwidth Competition
  - Scaling (E.g. Tesla 240+ Cores)
  - Headless GP-GPU vs GPU

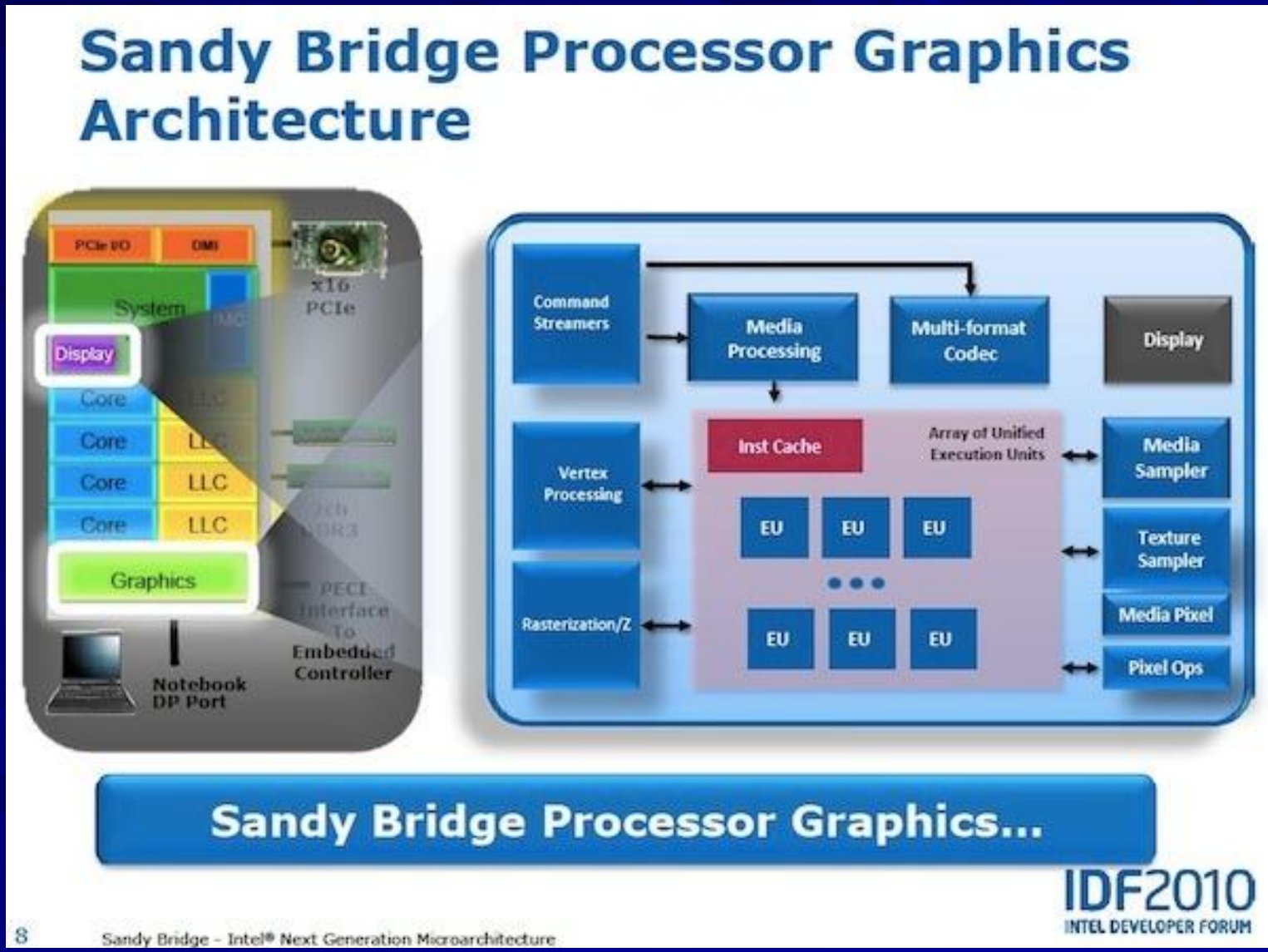
Sandy Bridge SoC



Sandy Bridge iGPU/dGPU HW



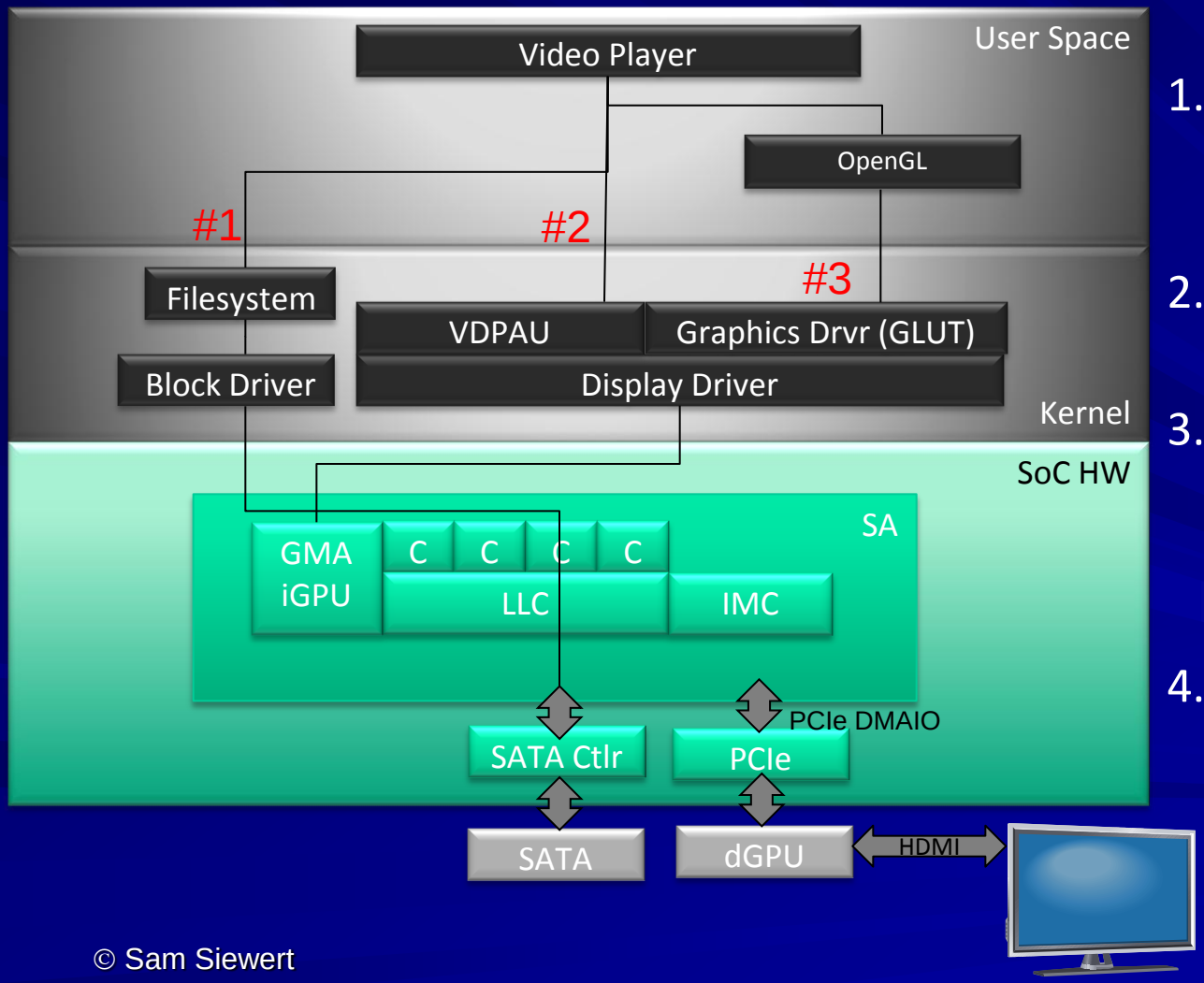
iGPU Detail



Offload, Co-Proc, Vector Proc

- GPU (Graphics Processing Units)
  - Evolved for Consumer CGI and Games
    - Physics Engines
    - 3D Rendering + Texture (4D Vector Operations)
    - Game Engines and Simulation
    - HD Output: HDMI, HD-SDI, Headless GP-GPU
  - Higher End Used for Digital Cinema / Post Production, Broadcast
    - PNY Quadro FX
    - NVIDIA CUDA for Post
- GP-GPU Being Used to Accelerate Encode, Transcode, Trans-rate, etc. - http://www.elementaltechnologies.com/
- Built-In SIMD Instruction Set Extensions – Intel SSE

Integrated vs Discrete GPU



Challenges

1. RT Transformation of Video Can Result in Lots of Bus I/O
2. Fetch Encoded Video from File or Network
3. Send to Decoder for Presentation, or Simple Decode to another Buffer
4. Transform and Render

Digital Media GP-GPU Applications (Visual Computing)

- Graphical Rendering and Video Morphing
- Video Game Physics Engines
- Encode and Trans-code MPEG Program and Transport Streams
- Image Processing
  - Digital Cinema Post-Production
  - Medical Imaging
  - Scientific Imaging
- Video Analytics (E.g. Facial Recognition)
- See NVIDIA Website Example Applications - http://www.nvidia.com/object/cuda_apps_flash_new.html

What We'll Do

- Investigate Speed-Up in Extended Lab
  - You Choose How
    - GPU
    - Intel SSE Instructions
    - Many-Core
    - Combination of the Above
  - Understand Where SISD, SPMD, and MIMD Will Help
  - Study Theory and Algorithmic Examples to Understand Speed-up
  - Understand Why it's Important

NVIDIA CUDA

Installation and Test
Simple Benchmarks

Installation Hurdles (Hardware and Driver)

- Install Your GeForce, Tesla, etc. and Verify on PCI Bus
- Download Driver (195.36.15), CUDA toolkit, and SDK
- Download “Getting Started”
- Install Driver, and If you See an ERROR
  - Exit X with /sbin/init 3
  - Use “sh driver-install-blah-blah.run”
  - Check /var/log/nvidia-installer.log
  - Use init 5 or startx to re-run X windows
- For Module Load Issue On FC12, First Blacklist Current X11 driver
  - See http://fedorasolved.org/video-solutions/nvidia-yum-kmod
  - Set up RPM Fusion Repository for YUM
  - Check Kernel Revision/Type with “uname –r”
  - Follow Detailed Instructions on Blacklisting nouveau for X11
  - Reboot
  - X Windows Should Show NVIDIA Splash – If not, See Troubleshooting on same page
- Re-install NVIDIA driver if needed, Check with:
  - /usr/bin/nvidia-settings – should come up without any errors
  - X-windows Should run on your GeForce

Installation Hurdles (Toolkit and SDK with GCC)

- Follow Downloaded “Getting Started”
- Install CUDA toolkit
- Install GPU SDK
- Set $PATH and $LD_LIBRARY_PATH
- Minor Changes to Work with GCC
  - Do “yum search glut” and then install glut development libraries
  - Update /usr/local/cuda/bin/nvcc.profile
    - Cures varargs compile bug
  - Update NVIDIA_GPU_Computing_SDK
    - Update common/common.mk NVCCFLAGS
    - Cures inline funciton compile errors
- Now Try NVIDIA_GPU_Computing_SDK/C and “make”
- Turn SELinux to DISABLED, PERMISSIVE

Play with CUDA Examples

- C/bin/linux/release - ./deviceQuery

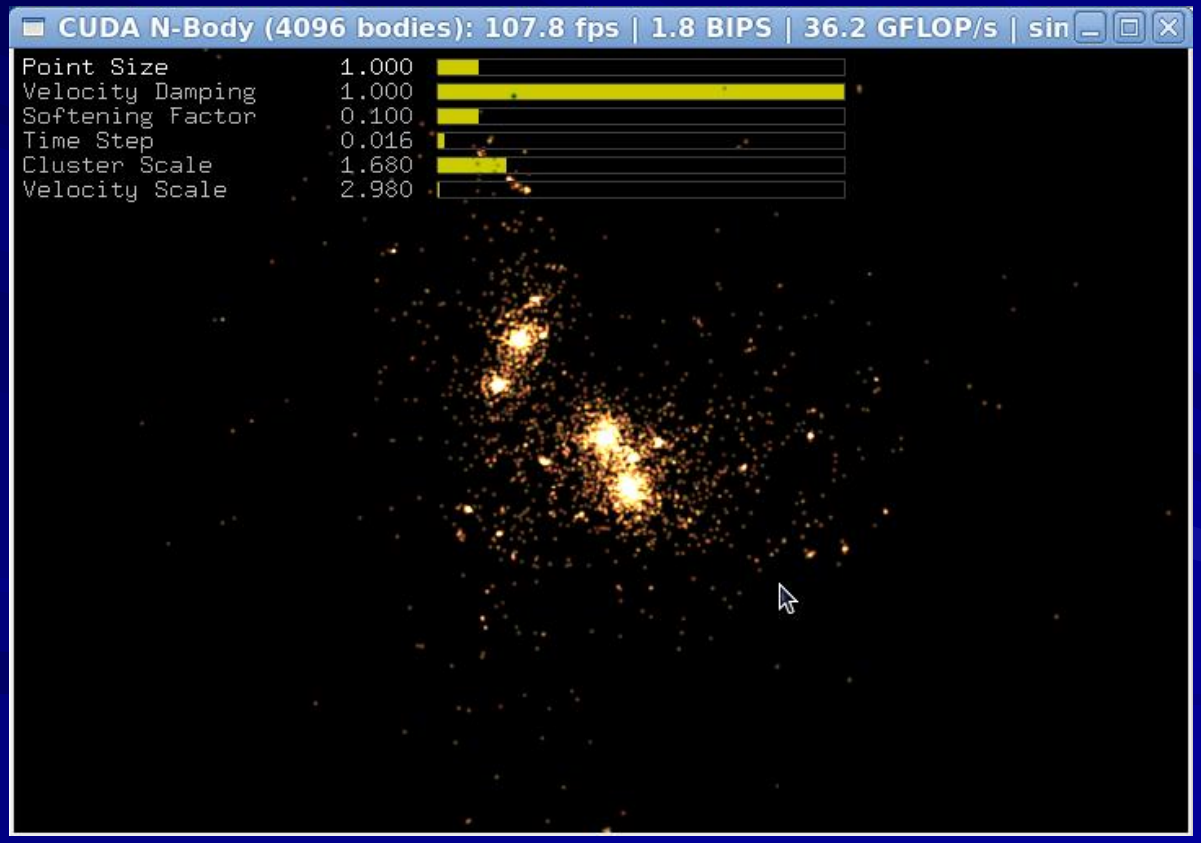
```
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8700M GT"
  CUDA Driver Version:                           3.0
  CUDA Runtime Version:                          3.0
  CUDA Capability Major revision number:         1
  CUDA Capability Minor revision number:         1
  Total amount of global memory:                 536150016 bytes
  Number of multiprocessors:                     4
  Number of cores:                               32
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
  Maximum sizes of each dimension of a block:    512 x 512 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             256 bytes
  Clock rate:                                    1.25 GHz
  Concurrent copy and execution:                 Yes
  Run time limit on kernels:                     Yes
  Integrated:                                    No
  Support host page-locked memory mapping:       No
  Compute mode:                                  Default (multiple host threads can use this device
    simultaneously)

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 134562363, CUDA Runtime Version = 3.0, NumDevs =
    1, Device = GeForce 8700M GT


PASSED
```

bandwidthTest

```
./bandwidthTest Starting...

Running on...

 Device 0: GeForce 8700M GT
 Quick Mode

 Host to Device Bandwidth, 1 Device(s), Paged memory
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     1645.3

 Device to Host Bandwidth, 1 Device(s), Paged memory
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     1123.6

 Device to Device Bandwidth, 1 Device(s)
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     15037.5


[bandwidthTest] - Test results:
PASSED
```

N-body, etc. …

- Test Environment, Measure Performance



Hands On Session and Practice

- Q&A on Prof's CUDA Laptop
- More CUDA Demo Code
- Read Programmer's Guide Next and Browse Example Source
- Read "Benchmarking GPU Devices with N-Body Simulations" – On our Web site
- Study SSE Intel Paper Example
- Disassemble Optimized Code and Step Through in Mixed Mode

GPU and GP-GPU Programming

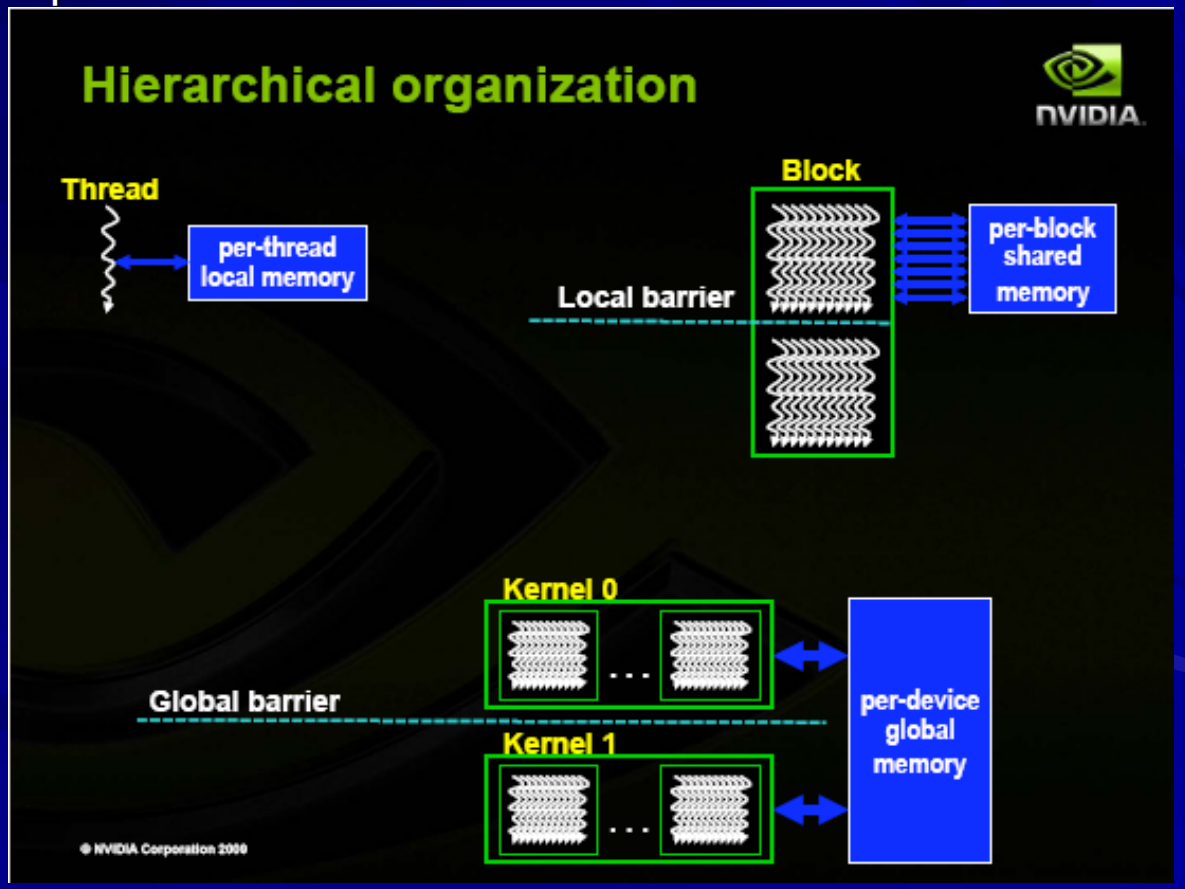
CUDA and Open-CL

How To Teach Yourself CUDA

- Install CUDA 3.0 Driver (Instructions from Lecture 11 and Install Guide)
  - http://ecee.colorado.edu/~ecen5033/ecen5033/lectures/Lecture11_files/frame.htm
  - http://developer.download.nvidia.com/compute/cuda/3_0/docs/GettingStartedLinux.pdf
- Install the Examples and Build
  - Follow NVCC updates in Lecture 11 for FC12
  - Run deviceQuery, bandwidthTest
- Download CUDA C Programer's Guide and API Doc
  - http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf
  - http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/CudaReferenceManual.pdf
- Study and Understand NVIDIA_GPU_Computing_SDK/C/src/vectorAdd.cu

Concurrent SPMD Concepts

- Parallel Kernels Composed of Many Threads
  - All Threads Execute the Same Program (“SP”)
- Threads are Grouped into Blocks
  - Same Block Can Cooperate
  - Unique ID



Simple C Code to Device Code

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

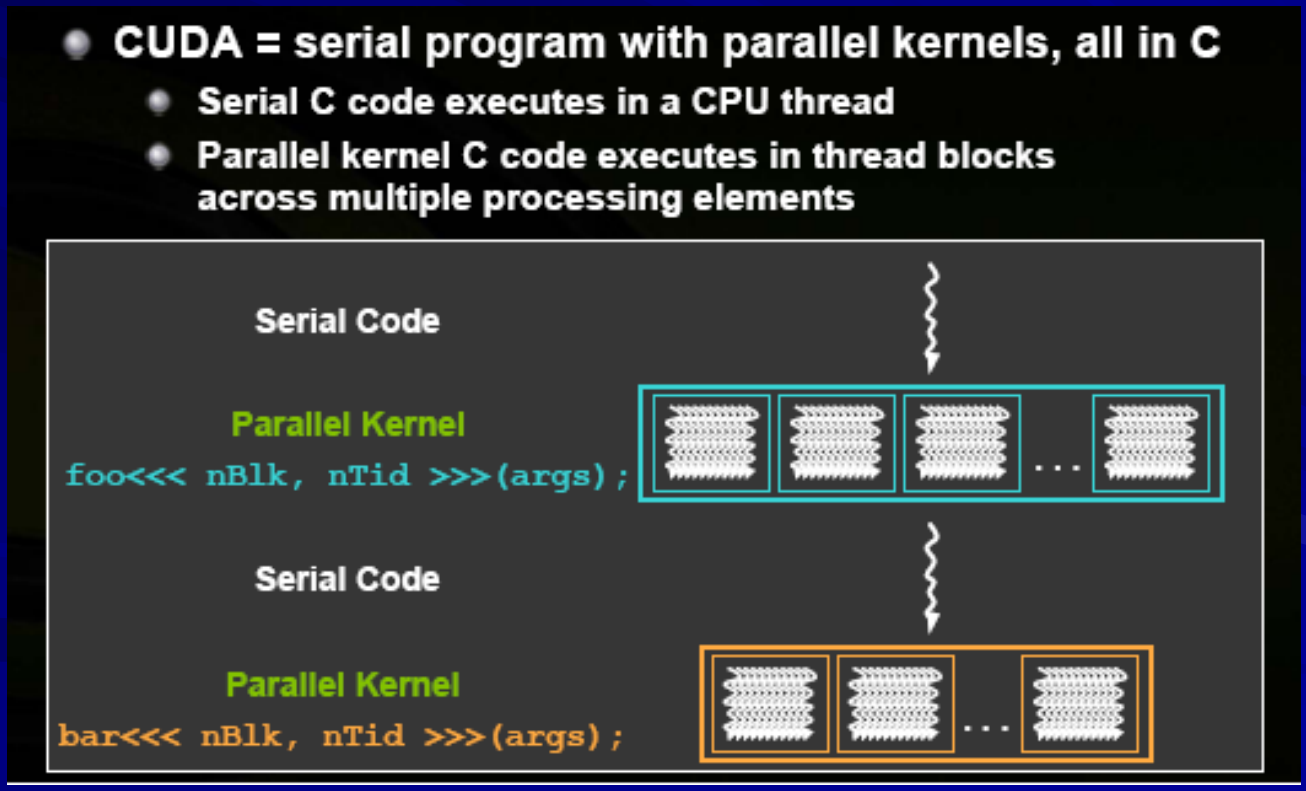
Standard C Code

```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}
// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Parallel C Code

Host/Device SPMD

- Sections of Serial and Parallel Co-Proc Execution

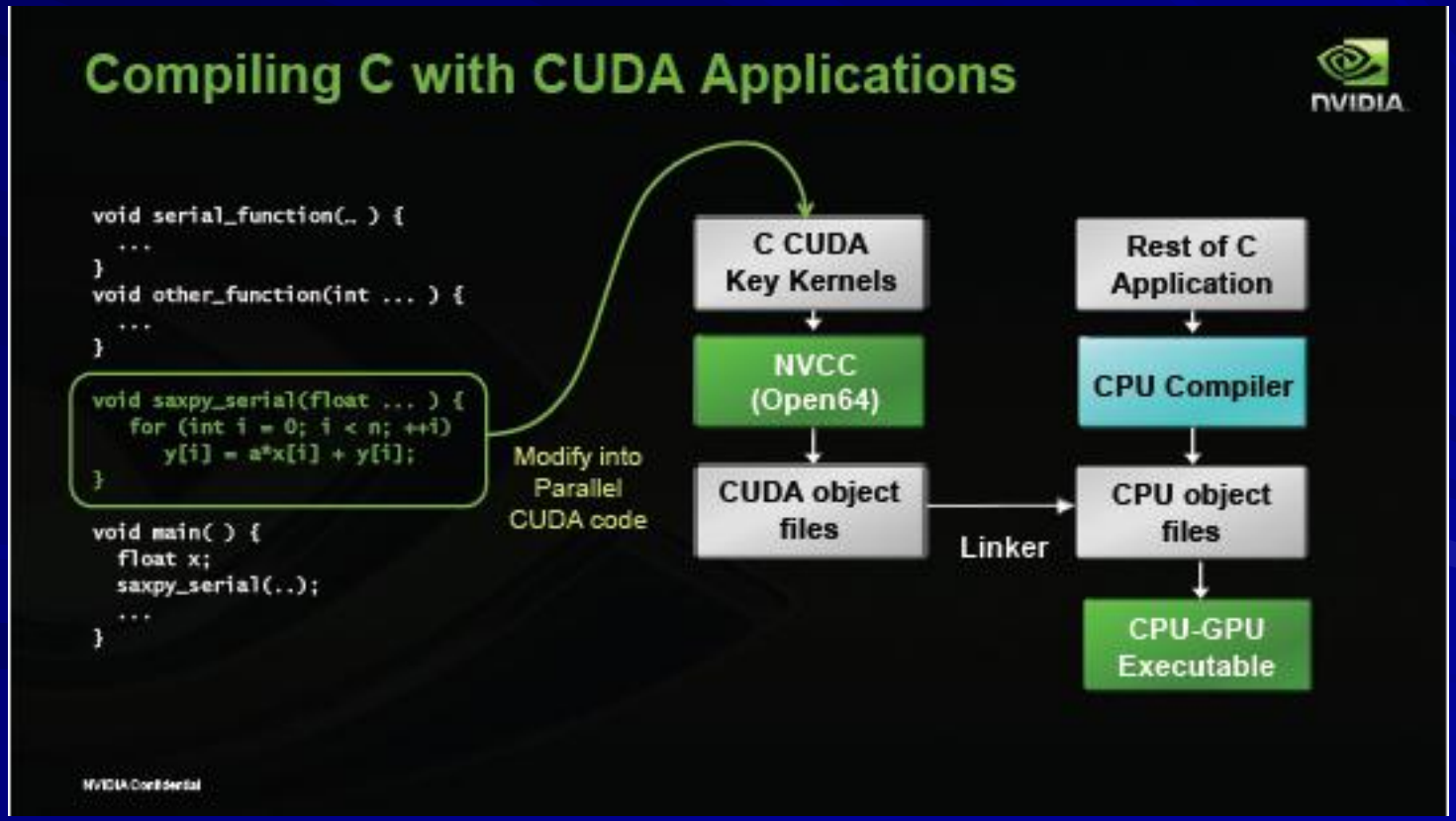


Basic CUDA Concepts – vectorAdd.cu

- Some includes - <cutil_inline.h>
- Declare Host “h_” and Device “d_” Pointers
- Device Code for GPU/GP-GPU
- Host Code (for x86 or AMD PCI-e Host)
  - Standard ANSI C with “cutil” Calls
  - E.g. cudaMalloc, cudaMemcpyHostToDevice
- Copy Data to Transform to the Device (cudaMemcpy)
- Invoke Kernel (Device Code)
  - threadsPerBlock
  - blocksPerGrid
  - Call Device Code Kernel
    - VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
  - Use “d_Variable” to denote device variable
- Copy Result Data Transformed from the Device
- Complexity Comes When Optimizing Memory Transfers and Asynchronous Operations Between Host and Device

NVCC Compiles CUDA Code, GCC Compiles C Code

- Linked Together for Hybrid SPMD Executable



More CUDA References

- GPU Technology Conference
  - http://www.nvidia.com/gtc
  - Hands-on, Application Examples, Architectural Directions, New Products from NVIDIA and Partners
- Programming Massively Parallel Processors (Morgan Kaufman – 2010)
  - David Kirk, Wen-Mei W. Hwu
  - 978-0-12-381472-2
- NVIDIA CUDA Zone Developer's Web Site
  - http://www.nvidia.com/object/cuda_home_new.html
  - http://www.nvidia.com/page/support.html
- CUDA On-Line Training
  - http://developer.nvidia.com/object/cuda_training.html
  - Many Partner Tools for Windows, Linux, Etc.