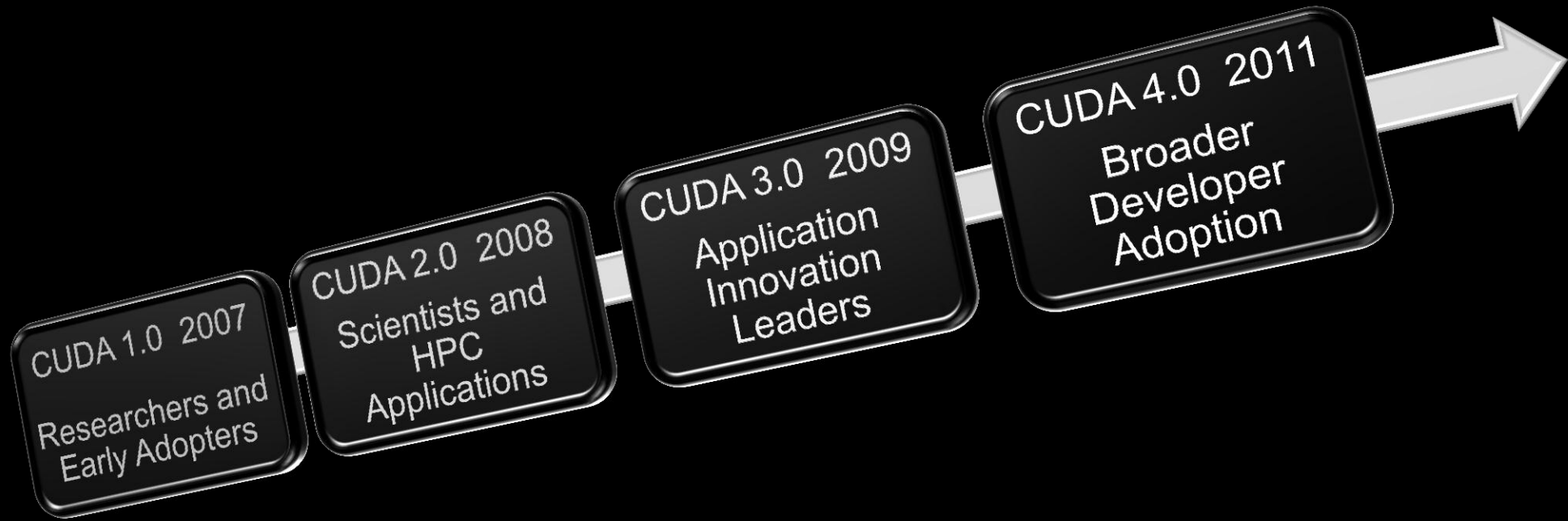CUDA 4.0

The 'Super' Computing Company
From Super Phones to Super Computers

CUDA 4.0 for Broader Developer Adoption

CUDA 1.0 2007
Researchers and Early Adopters

CUDA 2.0 2008
Scientists and HPC Applications

CUDA 3.0 2009
Application Innovation Leaders

CUDA 4.0 2011
Broader Developer Adoption

CUDA 4.0
Application Porting Made Simpler

Rapid Application Porting
Unified Virtual Addressing

Faster Multi-GPU Programming
NVIDIA GPUDirect™ 2.0

Easier Parallel Programming in C++
Thrust

CUDA 4.0: Highlights

Easier Parallel Application Porting

- Share GPUs across multiple threads
- Single thread access to all GPUs
- No-copy pinning of system memory
- New CUDA C/C++ features
- Thrust templated primitives library
- NPP image/video processing library
- Layered Textures

Faster Multi-GPU Programming

- NVIDIA GPUDirect™ v2.0
  - Peer-to-Peer Access
  - Peer-to-Peer Transfers
- Unified Virtual Addressing

New & Improved Developer Tools

- Auto Performance Analysis
- C++ Debugging
- GPU Binary Disassembler
- cuda-gdb for MacOS

Easier Porting of Existing Applications

Share GPUs across multiple threads

- **Easier porting of multi-threaded apps**

  pthreads / OpenMP threads share a GPU

- **Launch concurrent kernels from different host threads**

  Eliminates context switching overhead

- **New, simple context management APIs**

  Old context migration APIs still supported

Single thread access to all GPUs

- **Each host thread can now access all GPUs in the system**

  One thread per GPU limitation removed

- **Easier than ever for applications to take advantage of multi-GPU**

  Single-threaded applications can now benefit from multiple GPUs

  Easily coordinate work across multiple GPUs (e.g. halo exchange)

No-copy Pinning of System Memory

- **Reduce system memory usage and CPU memcpy() overhead**
  - **Easier to add CUDA acceleration to existing applications**
  - **Just register malloc'd system memory for async operations and then call cudaMemcpy() as usual**

Before No-copy Pinning	With No-copy Pinning
Extra allocation and extra copy required	Just register and go!
malloc(a)	
cudaMallocHost(b) memcpy(b, a)	cudaHostRegister(a)
cudaMemcpy() to GPU, launch kernels, cudaMemcpy() from GPU	
memcpy(a, b) cudaFreeHost(b)	cudaHostUnregister(a)

- **All CUDA-capable GPUs on Linux or Windows**
  - **Requires Linux kernel 2.6.15+ (RHEL 5)**

New CUDA C/C++ Language Features

- **C++ new/delete**
  - **Dynamic memory management**

- **C++ virtual functions**
  - **Easier porting of existing applications**

- **Inline PTX**
  - **Enables assembly-level optimization**

C++ Templatized Algorithms & Data Structures (Thrust)

- **Powerful open source C++ parallel algorithms & data structures**
  - **Similar to C++ Standard Template Library (STL)**
- **Automatically chooses the fastest code path at compile time**
  - **Divides work between GPUs and multi-core CPUs**
  - **Parallel sorting @ 5x to 100x faster**

Data Structures
• thrust::device_vector
• thrust::host_vector
• thrust::device_ptr
• Etc.

Algorithms
• thrust::sort
• thrust::reduce
• thrust::exclusive_scan
• Etc.

GPU-Accelerated Image Processing

NVIDIA Performance Primitives (NPP) library

- **10x to 36x faster image processing**
- **Initial focus on imaging and video related primitives**

Data exchange & initialization
Set, Convert, CopyConstBorder, Copy, Transpose, SwapChannels

Color Conversion
RGB To YCbCr (& vice versa), ColorTwist, LUT_Linear

Threshold & Compare Ops
Threshold, Compare

Statistics
Mean, StdDev, NormDiff, MinMax, Histogram, SqrIntegral, RectStdDev

Filter Functions
FilterBox, Row, Column, Max, Min, Median, Dilate, Erode, SumWindowColumn/Row

Geometry Transforms
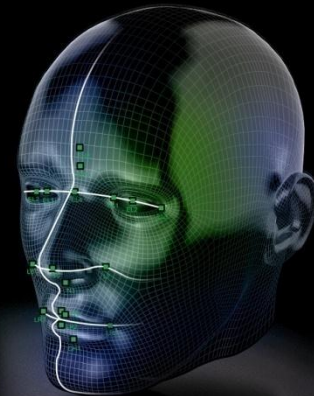Mirror, WarpAffine / Back/ Quad, WarpPerspective / Back / Quad, Resize

Arithmetic & Logical Ops
Add, Sub, Mul, Div, AbsDiff

JPEG
DCTQuantInv/Fwd, QuantizationTable

NVIDIA TESLA | Imaging

Layered Textures – Faster Image Processing

- **Ideal for processing multiple textures with same size/format**
  - **Large sizes supported on Tesla T20 (Fermi) GPUs (up to 16k x 16k x 2k)**
  - **e.g. Medical Imaging, Terrain Rendering (flight simulators), etc.**

- **Faster Performance**
  - **Reduced CPU overhead: single binding for entire texture array**
  - **Faster than 3D Textures: more efficient filter caching**
  - **Fast interop with OpenGL / Direct3D for each layer**
  - **No need to create/manage a texture atlas**

- **No sampling artifacts**
  - **Linear/Bilinear filtering applied only within a layer**

CUDA 4.0: Highlights

Easier Parallel Application Porting

Faster Multi-GPU Programming

New & Improved Developer Tools

- Share GPUs across multiple threads
- Single thread access to all GPUs
- No-copy pinning of system memory
- New CUDA C/C++ features
- Thrust templated primitives library
- NPP image/video processing library
- Layered Textures

- NVIDIA GPUDirect™ v2.0
  - Peer-to-Peer Access
  - Peer-to-Peer Transfers
- Unified Virtual Addressing

- Auto Performance Analysis
- C++ Debugging
- GPU Binary Disassembler
- cuda-gdb for MacOS

NVIDIA GPUDirect™: *Towards Eliminating the CPU Bottleneck*

Version 1.0

for applications that communicate over a network

Version 2.0

for applications that communicate within a node

- Direct access to GPU memory for 3rd party devices
- Eliminates unnecessary sys mem copies & CPU overhead
- Supported by Mellanox and Qlogic
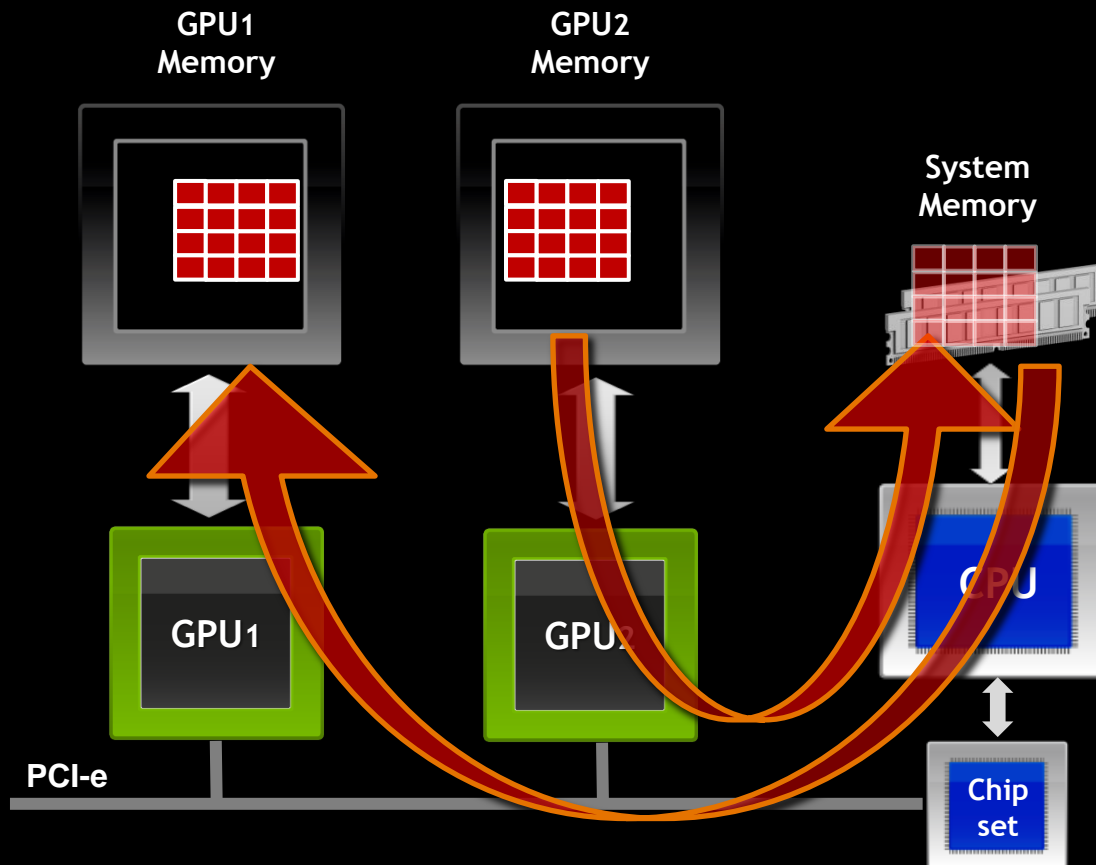- Up to 30% improvement in communication performance

- Peer-to-Peer memory access, transfers & synchronization

- Less code, higher programmer productivity

Details @ http://www.nvidia.com/object/software-for-tesla-products.html

Before NVIDIA GPUDirect™ v2.0

Required Copy into Main Memory

GPU1 Memory

GPU2 Memory

System Memory

Two copies required:
1. cudaMemcpy(GPU2, sysmem)
2. cudaMemcpy(sysmem, GPU1)
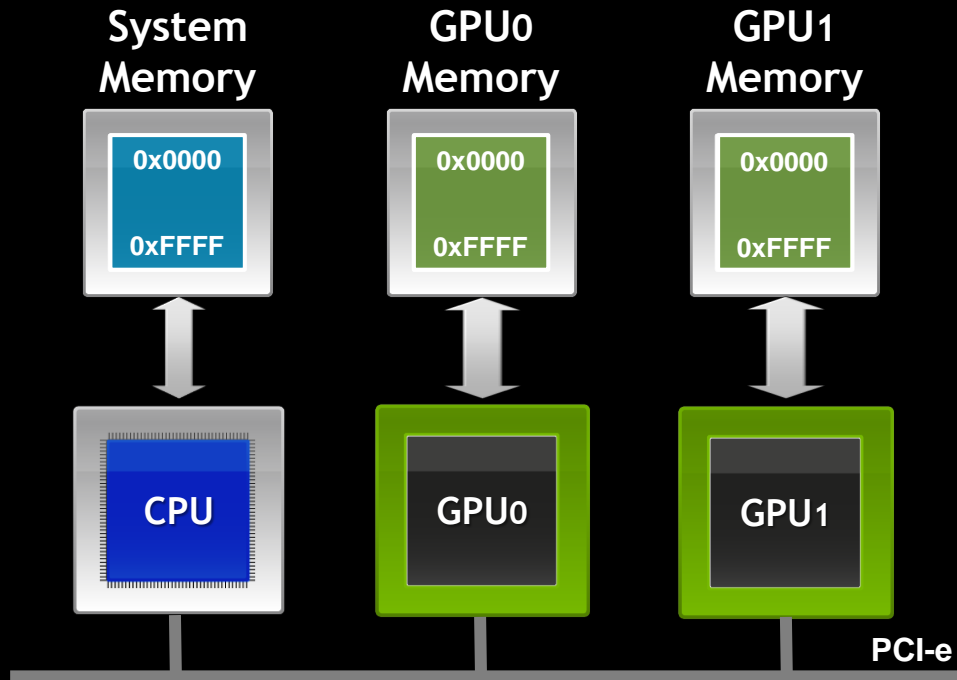
GPU1

GPU2

CPU

Chip set

PCI-e

GPUDirect v2.0: Peer-to-Peer Communication

- **Direct communication between GPUs**
  - **Faster - no system memory copy overhead**
  - **More convenient multi-GPU programming**

- **Direct Transfers**
  - **Copy from GPU_0 memory to GPU_1 memory**
  - **Works transparently with UVA**

- **Direct Access**
  - **GPU_0 reads or writes GPU_1 memory (load/store)**

- **Supported on Tesla 20-series and other Fermi GPUs**
  - **64-bit applications on Linux and Windows TCC**

Unified Virtual Addressing
Easier to Program with Single Address Space

No UVA: Multiple Memory Spaces

UVA : Single Address Space

System Memory	GPU0 Memory	GPU1 Memory
0x0000 0xFFFF	0x0000 0xFFFF	0x0000 0xFFFF

System Memory · GPU0 Memory · GPU1 Memory

0x0000 .... 0xFFFF

CPU	GPU0	GPU1

PCI-e

CPU	GPU0	GPU1

PCI-e
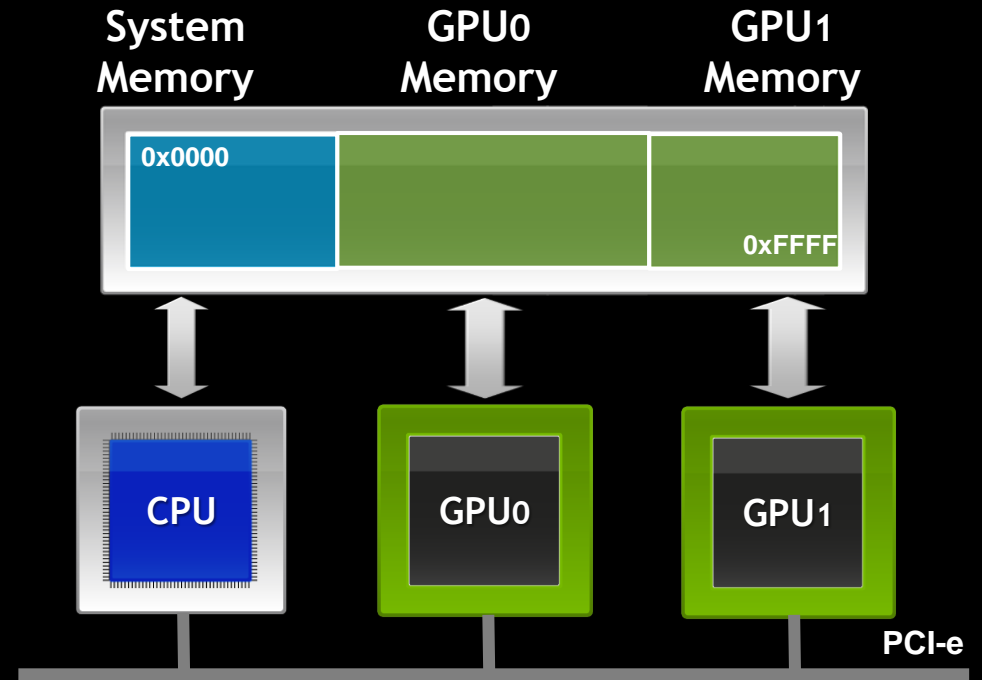
Unified Virtual Addressing

- **One address space for all CPU and GPU memory**
  - **Determine physical memory location from pointer value**
  - **Enables libraries to simplify their interfaces (e.g. cudaMemcpy)**

Before UVA	With UVA
Separate options for each permutation	One function handles all cases
cudaMemcpyHostToHost cudaMemcpyHostToDevice cudaMemcpyDeviceToHost cudaMemcpyDeviceToDevice	cudaMemcpyDefault (data location becomes an implementation detail)

- **Supported on Tesla 20-series and other Fermi GPUs**
  - **64-bit applications on Linux and Windows TCC**

CUDA 4.0: Highlights

Easier Parallel Application Porting

Faster Multi-GPU Programming

New & Improved Developer Tools

- Share GPUs across multiple threads
- Single thread access to all GPUs
- No-copy pinning of system memory
- New CUDA C/C++ features
- Thrust templated primitives library
- NPP image/video processing library
- Layered Textures

- NVIDIA GPUDirect™ v2.0
  - Peer-to-Peer Access
  - Peer-to-Peer Transfers
- Unified Virtual Addressing

- Auto Performance Analysis
- C++ Debugging
- GPU Binary Disassembler
- cuda-gdb for MacOS

Automated Performance Analysis in Visual Profiler

Summary analysis & hints

- Session
- Device
- Context
- Kernel

New UI for kernel analysis

- Identify limiting factor
- Analyze instruction throughput
- Analyze memory throughput
- Analyze kernel occupancy

convolutionColumnsKernel analysis - [Session4 - Device_0 - Context_0]

File   View

Analysis

Instruction Throughput Analysis for kernel convolutionColumnsKernel on device GeForce GTX 480

- IPC: 1.56
- Maximum IPC: 2
- Divergent branches(%): 0.00
- Control flow divergence(%): 0.03
- Replayed Instructions(%): 29.65
    - Global memory replay(%): 0.00
    - Local memory replays(%): 0.00
    - Shared bank conflict replay(%): 26.38
- Shared memory bank conflict per shared memory instruction(%): 99.90

Hint(s)

- **The kernel is compute bound**, to reduce instruction count
    - Understand the instruction mix, as single precision floating point, double precision floating point, int, mem, transcendentals, etc. have different throughputs. Use double precision arithmetic only when required (E.g. floating point literals without an f suffix (34.7) are interpreted as double precision as per C standard);
    - Try using arithmetic intrinsic functions.
    - Try using compiler flags(-ftz=true, -prec-div=false, -prec-sqrt=false etc) to get higher performance, but may result in some precision loss;
    Refer to the "Arithmetic Instructions" section in the "Performance Guidelines" chapter of the CUDA C Programming Guide for more details.
- **Shared memory bank conflicts are high** which causes serialization of threads within a warp. Shared memory bank conflicts can be reduced by

    - Using appropriate padding for data stored in shared memory so that each thread in a warp accesses data from a different bank;
    - Rearranging data in shared memory, thus changing access pattern;
    Refer to the "Shared Memory" section in the "Performance Guidelines" chapter of the CUDA C Programming Guide for more details.

Factors that may affect analysis

		Show all columns			
Limiting Factor Identification		GPU Timestamp (us)	GPU Time (us)	shared load Type:SM Run:4	shared store Type:SM Run:4
Memory Throughput Analysis	1	38718	1652.96	334560	24600
	2	41989.6	1652.86	334560	24600
Instruction throughput Analysis	3	44507.4	1652.93	334560	24600
	4	47024.9	1652.96	334560	24600
	5	49541.9	1653.09	334560	24600
Occupancy Analysis	6	52060.2	1652.15	224560	24600

New Features in cuda-gdb

Now available for both Linux and MacOS

info cuda threads
automatically updated in DDD

Breakpoints on all instances
of templated functions

Fermi
disassembly

(cuobjdump)

C++ symbols shown
in stack trace view

File   Edit   View   Program   Commands   Status   Source   Data   Help

(): `info cuda threads`

```
Cuda threads
BlockIdx ThreadIdx To BlockIdx ThreadIdx Count        Virtual PC        Filename   Line
Kernel 1
* (0,0,0)  (0,0,0)      (0,0,0)   (0,0,0)       1 0x000000001cea9880 templates.cu    12
  (0,0,0)  (1,0,0)      (0,0,0)   (1,0,0)       1 0x000000001cea98e0 templates.cu    15
  (1,0,0)  (0,0,0)      (1,0,0)   (0,0,0)       1 0x000000001cea9880 templates.cu    12
  (1,0,0)  (1,0,0)      (1,0,0)   (1,0,0)       1 0x000000001cea98e0 templates.cu    15
```

```
10        T incr = this->b;
11        if (threadIdx.x == 0)
          t += 4 + incr;
13        else
14          t += 3;
15        return t + this->a;
```

```
0x000000001cea9880 <_ZN8my_classIfE11my_functionEf+192>:       MOV R0, R0
0x000000001cea9888 <_ZN8my_classIfE11my_functionEf+200>:       MOV R3, R3
0x000000001cea9890 <_ZN8my_classIfE11my_functionEf+208>:       MOV32I R4, 0x40800000
0x000000001cea9898 <_ZN8my_classIfE11my_functionEf+216>:       FADD R3, R3, R4
0x000000001cea98a0 <_ZN8my_classIfE11my_functionEf+224>:       FADD R0, R0, R3
```

```
Breakpoint on CUDA kernel launch at my_kernel<int, float><<<(2,1,1),(2,1,1)>>> (out1=0x200100000, out2=0x200100200) at templates.cu:21
(gdb) break templates.cu:12
Breakpoint 1 at 0x1cea96f8: file templates.cu, line 12.
Breakpoint 2 at 0x1cea9880: file templates.cu, line 12.
warning: Multiple breakpoints were set.
They may be automatically deleted at the end of the run.
Use the "delete" command to delete unwanted breakpoints.
(gdb) info breakpoints
Num Type          Disp Enb Address            What
1   breakpoint     keep y   0x000000001cea96f8 in my_class<int>::my_function(int) at templates.cu:12
2   breakpoint     keep y   0x000000001cea9880 in my_class<float>::my_function(float) at templates.cu:12
(gdb) continue

Breakpoint 1, my_class<int>::my_function (this=0x3fffc30, t=3) at templates.cu:12
(gdb) continue

Breakpoint 2, my_class<float>::my_function (this=0x3fffc38, t=2) at templates.cu:12
(gdb) where
#0  my_class<float>::my_function (this=0x3fffc38, t=2) at templates.cu:12
#1  0x000000001cea95a0 in my_kernel<int, float><<<(2,1,1),(2,1,1)>>> (out1=0x200100000, out2=0x200100200) at templates.cu:29
(gdb)
```
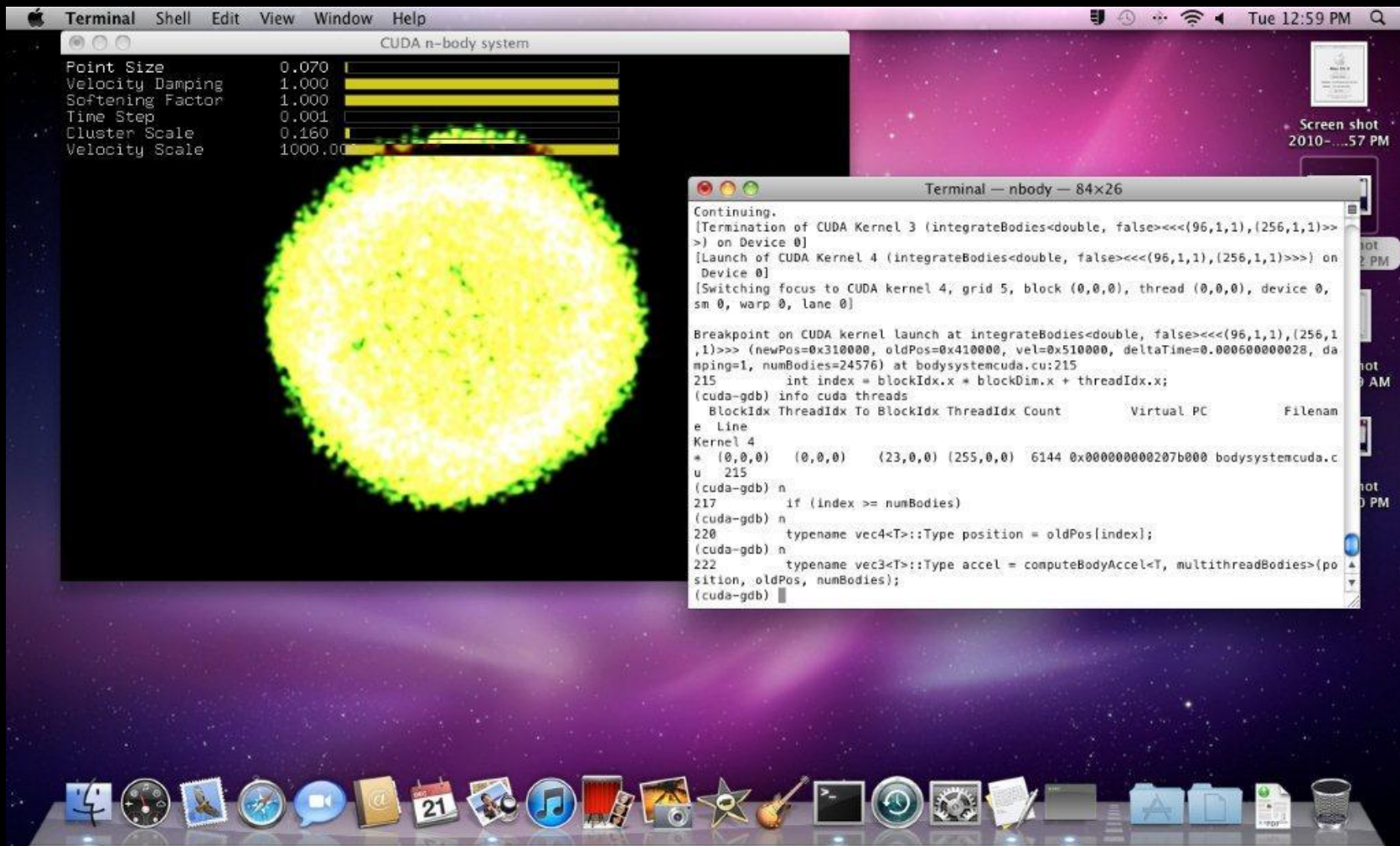
△ Display-1: `info cuda threads` (enabled)

Details @ http://developer.nvidia.com/object/cuda-gdb.html

cuda-gdb Now Available for MacOS

Details @ http://developer.nvidia.com/object/cuda-gdb.html

NVIDIA Parallel Nsight™ Pro 1.5

	Professional
CUDA Debugging	✓
Compute Analyzer	✓
CUDA / OpenCL Profiling	✓
Tesla Compute Cluster (TCC) Debugging	✓
Tesla Support: C1050/S1070 or higher	✓
Quadro Support: G9x or higher	✓
Windows 7, Vista and HPC Server 2008	✓
Visual Studio 2008 SP1 and Visual Studio 2010	✓
OpenGL and OpenCL Analyzer	✓
DirectX 10 & 11 Analyzer, Debugger & Graphics inspector	✓
GeForce Support: 9 series or higher	✓
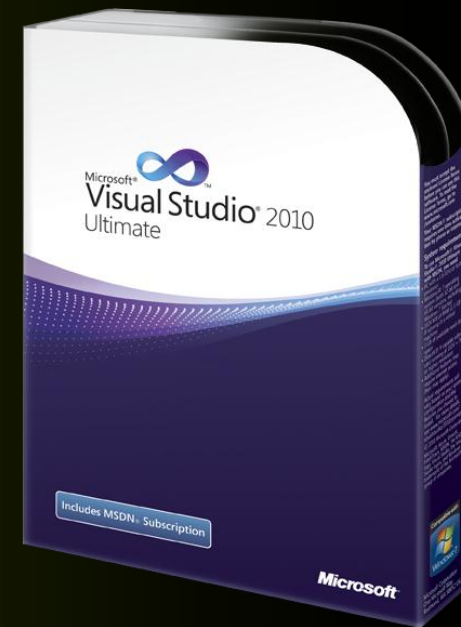
CUDA Registered Developer Program

All GPGPU developers should become NVIDIA Registered Developers

Benefits include:

- **Early Access to Pre-Release Software**
    - **Beta software and libraries**
    - **CUDA 4.0 Release Candidate available now**
- **Submit & Track Issues and Bugs**
    - **Interact directly with NVIDIA QA engineers**
- **New benefits in 2011**
    - **Exclusive Q&A Webinars with NVIDIA Engineering**
    - **Exclusive deep dive CUDA training webinars**
    - **In-depth engineering presentations on pre-release software**

Sign up Now: www.nvidia.com/ParallelDeveloper

Additional Information…

- CUDA Features Overview
- CUDA Developer Resources from NVIDIA
- CUDA 3rd Party Ecosystem
- PGI CUDA x86
- GPU Computing Research & Education
- NVIDIA Parallel Developer Program
- GPU Technology Conference 2011

CUDA Features Overview

	Platform	Programming Model	Parallel Libraries	Development Tools
New in CUDA 4.0	**GPUDirect^{tm} (v 2.0)** Peer-Peer Communication	**Unified Virtual Addressing** **C++ new/delete** **C++ Virtual Functions**	**Thrust C++ Library** Templated Performance Primitives Library	**Parallel Nsight Pro 1.5**

GPUDirect^{tm} (v 2.0)
  Peer-Peer Communication

Unified Virtual Addressing
C++ new/delete
C++ Virtual Functions

Thrust C++ Library
Templated Performance
Primitives Library

Parallel Nsight Pro 1.5

Hardware Features
ECC Memory
Double Precision
Native 64-bit Architecture
Concurrent Kernel Execution
Dual Copy Engines
6GB per GPU supported

Operating System Support
MS Windows 32/64
Linux 32/64
Mac OS X 32/64

Designed for HPC
Cluster Management
GPUDirect
Tesla Compute Cluster (TCC)
Multi-GPU support

C support
NVIDIA C Compiler
CUDA C Parallel Extensions
Function Pointers
Recursion
Atomics
malloc/free

C++ support
Classes/Objects
Class Inheritance
Polymorphism
Operator Overloading
Class Templates
Function Templates
Virtual Base Classes
Namespaces

Fortran support
CUDA Fortran (PGI)

NVIDIA Library Support
Complete math.h
Complete BLAS Library (1, 2 and 3)
Sparse Matrix Math Library
RNG Library
FFT Library (1D, 2D and 3D)
Video Decoding Library (NVCUVID)
Video Encoding Library (NVCUVENC)
Image Processing Library (NPP)
Video Processing Library (NPP)

3rd Party Math Libraries
CULA Tools (EM Photonics)
MAGMA Heterogeneous LAPACK
IMSL (Rogue Wave)
VSIPL (GPU VSIPL)

NVIDIA Developer Tools
Parallel Nsight
    for MS Visual Studio
cuda-gdb Debugger
    with multi-GPU support
CUDA/OpenCL Visual Profiler
CUDA Memory Checker
CUDA Disassembler
GPU Computing SDK
NVML
CUPTI

3rd Party Developer Tools
Allinea DDT
RogueWave /Totalview
Vampir
Tau
CAPS HMPP

CUDA Developer Resources from NVIDIA

Development Tools

CUDA Toolkit
Complete GPU computing development kit

cuda-gdb
GPU hardware debugging

cuda-memcheck
Identifies memory errors

cuobjdump
CUDA binary disassembler

Visual Profiler
GPU hardware profiler for CUDA C and OpenCL

Parallel Nsight Pro
Integrated development environment for Visual Studio

SDKs and Code Samples

GPU Computing SDK
CUDA C/C++, DirectCompute, OpenCL code samples and documentation

Books
CUDA by Example
GPU Computing Gems
Programming Massively
    Parallel Processors
Many more...

Optimization Guides
Best Practices for GPU computing and graphics development

Libraries and Engines

Math Libraries
CUFFT, CUBLAS, CUSPARSE, CURAND, math.h

3rd Party Libraries
CULA LAPACK, VSIPL

NPP Image Libraries
Performance primitives for imaging

App Acceleration Engines
Ray Tracing: Optix, iRay

Video Encoding / Decoding

NVCUVENC / VCUVID

http://developer.nvidia.com

CUDA 3rd Party Ecosystem

Cluster Tools

Cluster Management

Platform HPC

Platform Symphony

Bright Cluster manager

Ganglia Monitoring System

Moab Cluster Suite

Altair PBS Pro

Job Scheduling

Altair PBSpro

TORQUE

Platform LSF

MPI Libraries

Coming soon...

Parallel Language Solutions & APIs

PGI CUDA Fortran

PGI Accelerator (C/Fortran)

PGI CUDA x86

CAPS HMPP

pyCUDA (Python)

Tidepowerd GPU.NET (C#)

JCuda (Java)

Khronos OpenCL

Microsoft DirectCompute

3rd Party Math Libraries

CULA Tools (EM Photonics)

MAGMA Heterogeneous LAPACK

IMSL (Rogue Wave)

VSIPL (GPU VSIPL)

NAG

Parallel Tools

Parallel Debuggers

MS Visual Studio with Parallel Nsight Pro

Allinea DDT Debugger

TotalView Debugger

Parallel Performance Tools

ParaTools VampirTrace

TauCUDA Performance Tools

PAPI

HPC Toolkit

Compute Platform Providers

Cloud Providers

Amazon EC2

Peer 1

OEM's

Dell

HP

IBM

Infiniband Providers

Mellanox

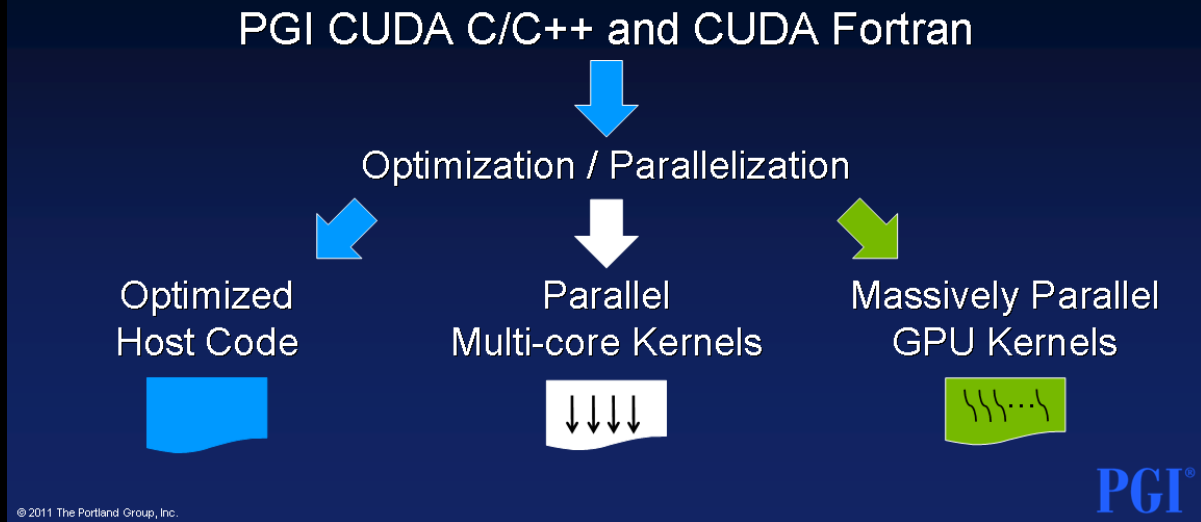QLogic

PGI CUDA x86 Compiler

Benefits

- Deploy CUDA apps on legacy systems without GPUs
- Less code maintenance for developers

Timeline

- **April/May 1.0 initial release**
  Develop, debug, test functionality
- **Aug 1.1 performance release**
  Multicore, SSE/AVX support

PGI CUDA Compilers for x86 and NVIDIA GPUs

PGI CUDA C/C++ and CUDA Fortran

Optimization / Parallelization

Optimized Host Code

Parallel Multi-core Kernels

Massively Parallel GPU Kernels

© 2011 The Portland Group, Inc.

PGI®

GPU Computing Research & Education

http://research.nvidia.com

World Class Research Leadership and Teaching

- **University of Cambridge**
- **Harvard University**
- **University of Utah**
- **University of Tennessee**
- **University of Maryland**
- **University of Illinois at Urbana-Champaign**
- **Tsinghua University**
- **Tokyo Institute of Technology**
- **Chinese Academy of Sciences**
- **National Taiwan University**
- **Georgia Institute of Technology**

Proven Research Vision

John Hopkins University	Mass. Gen. Hospital/NE Univ
Nanyan University	North Carolina State University
Technical University-Czech	Swinburne University of Tech.
CSIRO	Techische Univ. Munich
SINTEF	UCLA
HP Labs	University of New Mexico
ICHEC	University Of Warsaw-ICM
Barcelona SuperComputer Center	VSB-Tech
Clemson University	University of Ostrava
Fraunhofer SCAI	And more coming shortly.
Karlsruhe Institute Of Technology	

Academic Partnerships / Fellowships

GPGPU Education
350+ Universities
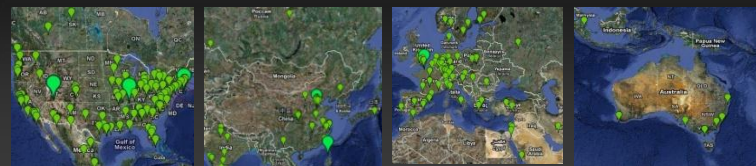
"Don't kid yourself. **GPUs are a game-changer**." said Frank Chambers, a GTC conference attendee shopping for GPUs for his finite element analysis work. "What we are seeing here is **like going from propellers to jet engines**. That made transcontinental flights routine. Wide access to this kind of computing power is making things like artificial retinas possible, and that wasn't predicted to happen until 2060."

- Inside HPC (Sept 22, 2010)

GPU Technology Conference 2011
October 11-14 | San Jose, CA

The one event you can't afford to miss

- Learn about leading-edge advances in GPU computing

- Explore the research as well as the commercial applications

- Discover advances in computational visualization

- Take a deep dive into parallel programming

Ways to participate

- Speak – share your work and gain exposure as a thought leader

- Register – learn from the experts and network with your peers

- Exhibit/Sponsor – promote your company as a key player in the GPU ecosystem

www.gputechconf.com