Languages, APIs and Development Tools for GPU Computing

Phillip Miller | Director of Product Management, Workstation Software

SIGGRAPH ASIA | December 16, 2010
Agenda for This Morning’s Overview

- Introduction - GPU Computing
- Options - Languages
- Assistance - Development Tools
- Approach - Application Design Patterns
- Leveraging - Libraries and Engines
- Scaling - Grid and Cluster
- Learning - Developer Resources
“GPGPU or GPU Computing”

- Using all processors in the system for the things they are best at doing:
  - Evolution of CPUs makes them good at sequential, *serial* tasks
  - Evolution of GPUs makes them good at *parallel* processing
# CUDA - NVIDIA’s Architecture for GPU Computing

## Broad Adoption
- **+250M** CUDA-enabled GPUs in use
- **+650k** CUDA Toolkit downloads in last 2 Yrs
- **+350** Universities teaching GPU Computing on the CUDA Architecture

## Cross Platform:
- Linux, Windows, MacOS
- Uses span HPC to Consumer

## GPU Computing Applications

<table>
<thead>
<tr>
<th>CUDA C/C++</th>
<th>OpenCL</th>
<th>Direct Compute</th>
<th>Fortran</th>
<th>Python, Java, .NET, …</th>
</tr>
</thead>
<tbody>
<tr>
<td>+100k developers</td>
<td>Commercial OpenCL Conformant Driver</td>
<td>Microsoft API for GPU Computing</td>
<td>PGI Accelerator</td>
<td>PyCUDA</td>
</tr>
<tr>
<td>In production usage since 2008</td>
<td>Publicly Available for all CUDA capable GPU’s</td>
<td>Supports all CUDA-Architecture GPUs (DX10 and DX11)</td>
<td>PGI CUDA Fortran</td>
<td>GPU.NET</td>
</tr>
<tr>
<td>SDK + Libs + Visual Profiler and Debugger</td>
<td>SDK + Visual Profiler</td>
<td></td>
<td></td>
<td>jCUDA</td>
</tr>
</tbody>
</table>

## NVIDIA GPU
with the CUDA Parallel Computing Architecture

---

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.
**GPU Computing Software Stack**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your GPU Computing Application</td>
<td></td>
</tr>
<tr>
<td>Application Acceleration Engines</td>
<td>Middleware, Modules &amp; Plug-ins</td>
</tr>
<tr>
<td>Foundation Libraries</td>
<td>Low-level Functional Libraries</td>
</tr>
<tr>
<td>Development Environment</td>
<td>Languages, Device APIs, Compilers, Debuggers, Profilers, etc.</td>
</tr>
<tr>
<td>CUDA Architecture</td>
<td></td>
</tr>
</tbody>
</table>
# Language & APIs for GPU Computing

<table>
<thead>
<tr>
<th>Approach</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Level Integration</td>
<td>MATLAB, Mathematica, LabVIEW</td>
</tr>
<tr>
<td>Implicit Parallel Languages (high level)</td>
<td>PGI Accelerator, HMPP</td>
</tr>
<tr>
<td>Abstraction Layer or API Wrapper</td>
<td>PyCUDA, CUDA.NET, jCUDA</td>
</tr>
<tr>
<td>Explicit Language Integration (high level)</td>
<td>CUDA C/C++, PGI CUDA Fortran</td>
</tr>
<tr>
<td>Device API (low level)</td>
<td>CUDA C/C++, DirectCompute, OpenCL</td>
</tr>
</tbody>
</table>
Example: Application Level Integration

GPU support with MathWorks Parallel Computing Toolbox™ and Distributed Computing Server™

MATLAB Parallel Computing Toolbox (PCT)
- PCT enables high performance through parallel computing on workstations
- NVIDIA GPU acceleration now available

MATLAB Distributed Computing Server (MDCS)
- MDCS allows a MATLAB PCT application to be submitted and run on a compute cluster
- NVIDIA GPU acceleration now available
MATLAB Performance on Tesla (previous GPU generation)

Relative Performance, Black-Scholes Demo
Compared to Single Core CPU Baseline

Core 2 Quad Q6600 2.4 GHz, 6 GB RAM, Windows 7 64-bit, Tesla C1060, single precision operations
Example: Implicit Parallel Languages
PGI Accelerator Compilers

SUBROUTINE SAXPY (A,X,Y,N)
INTEGER N
REAL A,X(N),Y(N)
!
$ACC REGION
DO I = 1, N
  X(I) = A*X(I) + Y(I)
ENDDO
!
$ACC END REGION
END

typedef struct dim3{ unsigned int x,y,z } dim3;
typedef struct uint3{ unsigned int x,y,z } uint3;
extern uint3 const threadIdx, blockIdx;
extern dim3 const blockDim, gridDim;
static __attribute__((__global__)) void pgicuda(
  int tc,
  int il,
  int i2,
  float* c,
  float* b,
  float* a
) {
  int i; int pl; int _i; 
  i = blockIdx.x * 64 + threadIdx.x;
  if( i < tc ){
    a[i+i2-1] = (c[i+i2-1]+c[i+i2-1]+b[i+i2-1]);
    b[i+i2-1] = c[i+i2];
    i = (i+1);
    pl = (pl-1);
  } }
Example: Abstraction Layer/Wrapper

PyCUDA / PyOpenCL

- CUDA C Code = Strings
- Generate Code Easily
  - Automated Tuning
- Batteries included:
  GPU Arrays, RNG, ...
- Integration: numpy arrays,
  Plotting, Optimization, ...

- All of CUDA in a modern
  scripting language
- Full Documentation
- Free, open source (MIT)
- Also: PyOpenCL

http://mathema.tician.de/software/pycuda

Slide courtesy of Andreas Klöckner, Brown University
Example: Language Integration
CUDA C: C with a few keywords

```c
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);
```

```c
__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<<<nblobckes, 256>>>(n, 2.0, x, y);
```
Example: Low-level Device API

OpenCL

- Cross-vendor open standard
  - Managed by the Khronos Group

- Low-level API for device management and launching kernels
  - Close-to-the-metal programming interface
  - JIT compilation of kernel programs

- C-based language for compute kernels
  - Kernels must be optimized for each processor architecture

NVIDIA released the first OpenCL conformant driver for Windows and Linux to thousands of developers in June 2009

http://www.khronos.org/opencl
Example: Low-level Device API

Direct Compute

- Microsoft standard for all GPU vendors
  - Released with DirectX® 11 / Windows 7
  - Runs on all +100M CUDA-enabled DirectX 10 class GPUs and later

- Low-level API for device management and launching kernels
  - Good integration with DirectX 10 and 11

- Defines HLSL-based language for compute shaders
  - Kernels must be optimized for each processor architecture
Example: New Approach

**GPU.NET**

- Write GPU kernels in C#, F#, VB.NET, etc.
- Exposes a minimal API accessible from any .NET-based language
  - Learn a new API instead of a new language
- JIT compilation = *dynamic* language support
- Don’t rewrite your existing code
  - Just give it a “touch-up”
## Language & APIs for GPU Computing

<table>
<thead>
<tr>
<th>Approach</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Level Integration</td>
<td>MATLAB, Mathematica, LabVIEW</td>
</tr>
<tr>
<td>Implicit Parallel Languages (high level)</td>
<td>PGI Accelerator, HMPP</td>
</tr>
<tr>
<td>Abstraction Layer or API Wrapper</td>
<td>PyCUDA, CUDA.NET, jCUDA</td>
</tr>
<tr>
<td>Explicit Language Integration (high level)</td>
<td>CUDA C/C++, PGI CUDA Fortran</td>
</tr>
<tr>
<td>Device API (low level)</td>
<td>CUDA C/C++, DirectCompute, OpenCL</td>
</tr>
</tbody>
</table>
Parallel Nsight for Visual Studio
Integrated development for CPU and GPU
Windows GPU Development for 2010
NVIDIA Parallel Nsight™ 1.5

nvcc + FX Composer
CUDA-gdb + Shader Debugger
cuda-memcheck + PerfHUD
Visual Profiler + ShaderPerf
cudaprof + Platform Analyzer
### 4 Flexible GPU Development Configurations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Desktop**     | Single machine, Single NVIDIA GPU  
Analyzer, Graphics Inspector  
New: Single machine, Dual NVIDIA GPUs  
Analyzer, Graphics Inspector, Compute Debugger |
| **Networked**   | Two machines connected over the network  
Analyzer, Graphics Inspector, Compute Debugger, Graphics Debugger |
| **Workstation SLI** | SLI Multi OS workstation with multiple Quadro GPUs  
Analyzer, Graphics Inspector, Compute Debugger, Graphics Debugger |
Linux: NVIDIA cuda-gdb

CUDA debugging **integrated** into GDB on Linux

- Supported on 32bit & 64bit Linux, MacOS to come.
- **Seamlessly** debug both the host/CPU and device/GPU code
- Set **breakpoints** on any source line or symbol name
- Access and print all CUDA memory allocs, local, global, constant and shared vars

Included in the CUDA Toolkit
3rd Party: DDT debugger

Latest News from Allinea

- CUDA SDK 3.0 with DDT 2.6
  - Released June 2010
  - Fermi and Tesla support
  - cuda-memcheck support for memory errors
  - Combined MPI and CUDA support
  - Stop on kernel launch feature
  - Kernel thread control, evaluation and breakpoints
  - Identify thread counts, ranges and CPU/GPU threads easily

- SDK 3.1 in beta with DDT 2.6.1

- SDK 3.2
  - Coming soon: multiple GPU device support
3rd Party: TotalView Debugger

- Latest from TotalView debugger (in Beta)
  - Debugging of application running on the GPU device
  - Full visibility of both Linux threads and GPU device threads
    - Device threads shown as part of the parent Unix process
    - Correctly handle all the differences between the CPU and GPU
  - Fully represent the hierarchical memory
    - Display data at any level (registers, local, block, global or host memory)
    - Making it clear where data resides with type qualification
- Thread and Block Coordinates
  - Built in runtime variables display threads in a warp, block and thread dimensions and indexes
  - Displayed on the interface in the status bar, thread tab and stack frame

Device thread control
- Warps advance Synchronously

Handles CUDA function inlining
- Step in to or over inlined functions
- Reports memory access errors
  - CUDA memcheck
- Can be used with MPI
NVIDIA Visual Profiler

- Analyze GPU HW performance signals, kernel occupancy, instruction throughput, and more
- Highly configurable tables and graphical views
- Save/load profiler sessions or export to CSV for later analysis
- Compare results visually across multiple sessions to see improvements
- Windows, Linux and Mac OS X
- OpenCL support on Windows and Linux

Included in the CUDA Toolkit
GPU Computing SDK

Hundreds of code samples for CUDA C, DirectCompute and OpenCL

- Finance
- Oil & Gas
- Video/Image Processing
- 3D Volume Rendering
- Particle Simulations
- Fluid Simulations
- Math Functions
Approach - Design Patterns
Accelerating Existing Applications

1. Identify Possibilities
   - Profile for Bottlenecks, Inspect for Parallelism

2. Port Relevant Portion
   - A Debugger is a good starting point, Consider Libraries & Engines vs. Custom

3. Validate Gains
   - Benchmark vs. CPU version

4. Optimize
   - Parallel Nsight, Visual Profiler, GDB, Tau CUDA, etc.

5. Deploy
   - Maintain original as CPU fallback if desired.
Trivial Application (Accelerating a Process)

Design Rules:
- Serial task processing on CPU
- Data Parallel processing on GPU
  - Copy input data to GPU
  - Perform parallel processing
  - Copy results back
- Follow guidance in the CUDA C Best Practices Guide

The CUDA C Runtime could be substituted with other methods of accessing the GPU
Basic Application – using multi-GPU

“Trivial Application” plus:

- Maximize overlap of data transfers and computation
- Minimize communication required between processors
- Use one CPU thread to manage each GPU

Multi-GPU notebook, desktop, workstation and cluster node configurations are increasingly common
Graphics Application

“Basic Application” plus:

- Use graphics interop to avoid unnecessary copies
- In Multi-GPU systems, put buffers to be displayed in GPU Memory of GPU attached to the display
Basic Library

“Basic Application” plus:
- Avoid unnecessary memory transfers
  - Use data already in GPU memory
  - Create and leave data in GPU memory

These rules apply to plug-ins as well
Application with Plug-ins

“Basic Application” plus:

- **Plug-in Mgr**
  - Allows Application and Plug-ins to (re)use same GPU memory
  - Multi-GPU aware

- Follow “Basic Library” rules for the Plug-ins
“Basic Application” plus:

- Use Shared Memory for intra-node communication or pthreads, OpenMP, etc.
- Use MPI to communicate between nodes
Leveraging - Libraries & Engines
GPU Computing Software Stack

Your GPU Computing Application

Application Acceleration Engines
Middleware, Modules & Plug-ins

Foundation Libraries
Low-level Functional Libraries

Development Environment
Languages, Device APIs, Compilers, Debuggers, Profilers, etc.

CUDA Architecture
CUFFT Library 3.2:

Improving Radix-3, -5, -7

Radix-3 (SP, ECC off)

Radix-3 (DP, ECC off)

Radix-5, -7 and mixed radix improvements not shown

CUFFT 3.2 & 3.1 on NVIDIA Tesla C2070 GPU
MKL 10.2.3.029 on Quad-Core Intel Core i7 (Nehalem)
Up to 2x average speedup over CUBLAS 3.1

Less variation in performance for different dimensions vs. 3.1

Average speedup of \{S/D/C/Z\}GEMM x \{NN,NT,TN,TT\}

CUFFT 3.2 & 3.1 on NVIDIA Tesla C2050 GPU

MKL 10.2.3.029 on Quad-Core Intel Core i7 (Nehalem)
3rd Party Example: CULA (LAPACK for heterogeneous systems)

"CULAPACK" Library
- Dense linear algebra
- C/C++ & FORTRAN
- 150+ Routines

MATLAB Interface
- 15+ functions
- Up to 10x speedup

Partnership
Developed in partnership with NVIDIA

Supercomputer Speeds
Performance 7x of Intel’s MKL LAPACK

GPU Accelerated Linear Algebra
Supercomputing Speeds

This graph shows the relative speed of many CULA functions when compared to Intel’s MKL 10.2. Benchmarks were obtained comparing an NVIDIA Tesla C2050 (Fermi) and an Intel Core i7 860. More at www.culatools.com
CUSparse Library: Matrix Performance vs. CPU

Multiplication of a sparse matrix by multiple vectors

MKL 10.2

Average speedup across S,D,C,Z

CUSPARSE 3.2 on NVIDIA Tesla C2050 GPU
MKL 10.2.3.029 on Quad-Core Intel Core i7 (Nehalem)
CURan Libray: Random Number Generation

Generating 100K Sobol' Samples - GPU vs. CPU

CURAND 3.2 on NVIDIA Tesla C2050 GPU
MKL 10.2.3.029 on Quad-Core Intel Core i7 (Nehalem)
NAG GPU Library

- Monte Carlo related
  - L’Ecuyer, Sobol RNGs
  - Distributions, Brownian Bridge

- Coming soon
  - Mersenne Twister RNG
  - Optimization, PDEs

- Seeking input from the community

- For up-to-date information: www.nag.com/numeric/gpus
NVPP Library: Graphics Performance Primitives

- Similar to Intel IPP focused on image and video processing
- 6x - 10x average speedup vs. IPP
  - 2800 performance tests
- Core i7 (new) vs. GTX 285 (old)
- Now available with CUDA Toolkit

[Graph showing Aggregate Performance Results]
OpenVIDIA

- Open source, supported by NVIDIA
- Computer Vision Workbench (CVWB)
  - GPU imaging & computer vision
  - Demonstrates most commonly used image processing primitives on CUDA
  - Demos, code & tutorials/information

http://openvidia.sourceforge.net
More Open Source Projects

- **Thrust**: Library of parallel algorithms with high-level STL-like interface
  - [http://code.google.com/p/thrust](http://code.google.com/p/thrust)

- **OpenCurrent**: C++ library for solving PDE’s over regular grids
  - [http://code.google.com/p/opencurrent](http://code.google.com/p/opencurrent)

- **200+ projects** on Google Code & SourceForge
  - Search for CUDA, OpenCL, GPGPU
GPU Computing Software Stack

Your GPU Computing Application

Application Acceleration Engines
Middleware, Modules & Plug-ins

Foundation Libraries
Low-level Functional Libraries

Development Environment
Languages, Device APIs, Compilers, Debuggers, Profilers, etc.

CUDA Architecture
NVIDIA PhysX™ - the World’s Most Deployed Physics API

Major PhysX Site Licensees

Integrated in Major Game Engines
- UE3
- Gamebryo
- Vision
- Instinct
- Trinigy
- Diesel
- Unity 3d
- Hero
- BigWorld

Cross Platform Support

Middleware & Tool Integration (APEX)
- SpeedTree
- 3ds Max
- Natural Motion
- Maya
- Fork Particles
- Softimage
- Emotion FX

Presented by NVIDIA
Scaling - Grid & Cluster Mngmnt.
## GPU Management & Monitoring

### NVIDIA Systems Management Interface (nvidia-smi)

<table>
<thead>
<tr>
<th>Products</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>All GPUs</td>
<td>• List of GPUs</td>
</tr>
<tr>
<td></td>
<td>• Product ID</td>
</tr>
<tr>
<td></td>
<td>• GPU Utilization</td>
</tr>
<tr>
<td></td>
<td>• PCI Address to Device Enumeration</td>
</tr>
<tr>
<td>Server products</td>
<td>• Exclusive use mode</td>
</tr>
<tr>
<td></td>
<td>• ECC error count &amp; location (Fermi only)</td>
</tr>
<tr>
<td></td>
<td>• GPU temperature</td>
</tr>
<tr>
<td></td>
<td>• Unit fan speeds</td>
</tr>
<tr>
<td></td>
<td>• PSU voltage/current</td>
</tr>
<tr>
<td></td>
<td>• LED state</td>
</tr>
<tr>
<td></td>
<td>• Serial number</td>
</tr>
<tr>
<td></td>
<td>• Firmware version</td>
</tr>
</tbody>
</table>

Use `CUDA_VISIBLE_DEVICES` to assign GPUs to process
Most Advanced Cluster Management Solution for GPU clusters

Includes:

- NVIDIA CUDA, OpenCL libraries and GPU drivers
- Automatic sampling of all available NVIDIA GPU metrics
- Flexible graphing of GPU metrics against time
- Visualization of GPU metrics in Rackview
- Powerful cluster automation, setting alerts, alarms and actions when GPU metrics exceed set thresholds
- Health checking framework based on GPU metrics
- Support for all Tesla GPU cards and GPU Computing Systems, including the most recent “Fermi” models
Selecting GPGPU Nodes
NVIDIA Developer Resources

**DEVELOPMENT TOOLS**
- **CUDA Toolkit**
  Complete GPU computing development kit
- **cuda-gdb**
  GPU hardware debugging
- **Visual Profiler**
  GPU hardware profiler for CUDA C and OpenCL
- **Parallel Nsight**
  Integrated development environment for Visual Studio
- **NVPerfKit**
  OpenGL|D3D performance tools
- **FX Composer**
  Shader Authoring IDE

**SDKs AND CODE SAMPLES**
- **GPU Computing SDK**
  CUDA C, OpenCL, DirectCompute code samples and documentation
- **Graphics SDK**
  DirectX & OpenGL code samples
- **PhysX SDK**
  Complete game physics solution
- **OpenAutomate**
  SDK for test automation

**VIDEO LIBRARIES**
- **Video Decode Acceleration**
  NVCUVID / NVCUVCENC
  DXVA
  Win7 MFT
- **Video Encode Acceleration**
  NVCUVCENC
  Win7 MFT
- **Post Processing**
  Noise reduction / De-interlace /
  Polyphase scaling / Color process

**ENGINES & LIBRARIES**
- **Math Libraries**
  CUFFT, CUBLAS,CUSPARSE, CURAND, ...
- **NPP Image Libraries**
  Performance primitives for imaging
- **App Acceleration Engines**
  Optimized software modules for GPU acceleration
- **Shader Library**
  Shader and post processing
- **Optimization Guides**
  Best Practices for GPU computing and Graphics development

http://developer.nvidia.com
10 Published books with 4 in Japanese, 3 in English, 2 in Chinese, 1 in Russian
An empirically tuned 2D and 3D FFT library on CUDA GPU
L Gu, X Li, J Siegel - Proceedings of the 24th ACM International ..., 2010 - portal.acm.org
Page 1. An Empirically Tuned 2D and 3D FFT Library on CUDA GPU Lian Gu Department of ECE University of Delaware Newark, DE. USA lianggu@udel.edu ... A CUDA GPU is most easily described as a collection of Multiprocessors(MPs). ...
Related articles

Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters
CT Yang, CL Huang, CF Lin - Computer Physics Communications, 2010 - Elsevier
Nowadays, NVIDIA's CUDA is a general purpose scalable parallel programming model for writing highly parallel applications. It provides several key abstractions - a hierarchy of thread blocks, shared memory, and barrier synchronization. This model has proven quite ...

Accelerating SSL with GPUs
K Jong, S Han, S Han, S Moon, KS .... - ACM SIGCOMM Computer ..., 2010 - portal.acm.org ...
... General Terms Design, experimentation, performance Keywords SSL, CUDA, GPU 1. INTRODUCTION Secure Sockets Layer (SSL) and Transport Layer Security (TLS) have served as a secure communication channel in the Internet for the past 15 years. ...

High Precision Numerical Simulations of Rotating Black Holes Accelerated by CUDA
R Ginjupalli, G Khanna, G Carbone, M Scargiali ..., - Arxiv preprint arXiv ..., 2010 - arxiv.org ...
... It is this code that we accelerate in our work using the Tesla CUDA GPU and also the Cell BE. ... II. NVIDIA CUDA GPU AND ST: CELL BE. All processor manufacturers have moved towards multi-core designs today in the quest for higher performance. ...
Related articles - View as HTML - All 4 versions

An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters
DA Jacobsen, JC Thibault, J .... - Mechanical and ..., 2010 - scholarworks.boisestate.edu ...
... Boise State University jtcBoise State University jctThibault@gmail.com jtcBoise State University. senocal@boisestate.edu This paper is posted at ScholarWorks. http://scholarworks.boisestate.edu/mecheng_facpubs/5 ... An MPI-CUDA Implementation ...
Cited by 1 - All 5 versions

An effective GPU implementation of breadth-first search
L Luo, M Wong, W Hua - Proceedings of the 47th Design ..., 2010 - portal.acm.org ...
... General Terms Algorithms, Performance Keywords CUDA, GPU computing, BFS ...

GPU Computing Research & Education

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

Premier Academic Partners

Proven Research Vision
Launched June 1st
with 5 premiere Centers
and more in review
John Hopkins University, USA
Nanyan University, Singapore
Technical University of Ostrava, Czech
CSIRO, Australia
SINTEF, Norway

Exclusive Events, Latest HW, Discounts

Quality GPGPU Teaching
Launched June 1st
with 7 premiere Centers
and more in review
McMaster University, Canada
Potsdam, USA
UNC-Charlotte, USA
Cal Poly San Luis Obispo, USA
ITESM, Mexico
Czech Technical University, Prague, Czech
Qingdao University, China

Teaching Kits, Discounts, Training

Academic Partnerships / Fellowships
Supporting 100’s of Researchers
around the globe ever year

NV Research
http://research.nvidia.com

Education
350+ Universities

PRESENTED BY NVIDIA
Thank you!
Database Application

- Minimize network communication
- Move the analysis “upstream” to stored procedures
- Treat each stored procedure like a “Basic Application” in the DB itself
- An App Server could also be a “Basic Application”
- A dedicated Client could also be a “Basic Application”

Data Mining, Business Intelligence, etc.