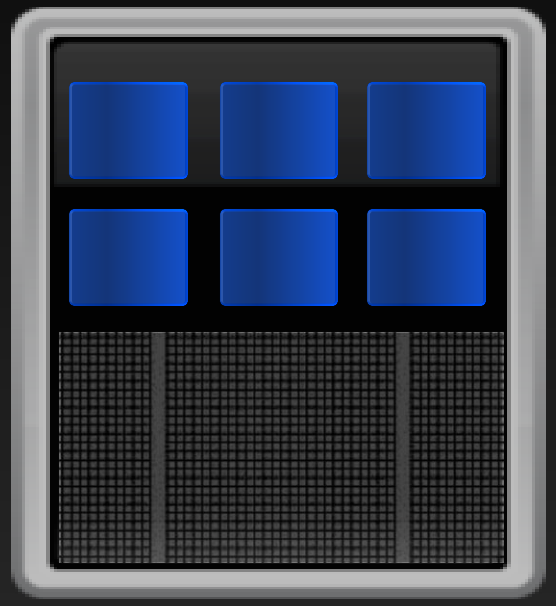
GPU TECHNOLOGY CONFERENCE

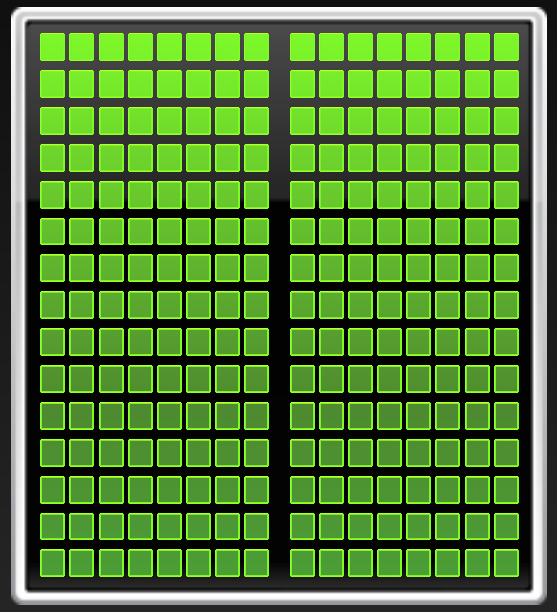
Languages, APIs & Development Tools for GPU Computing

Will Ramey, Sr. Product Manager, NVIDIA

CPU



GPU

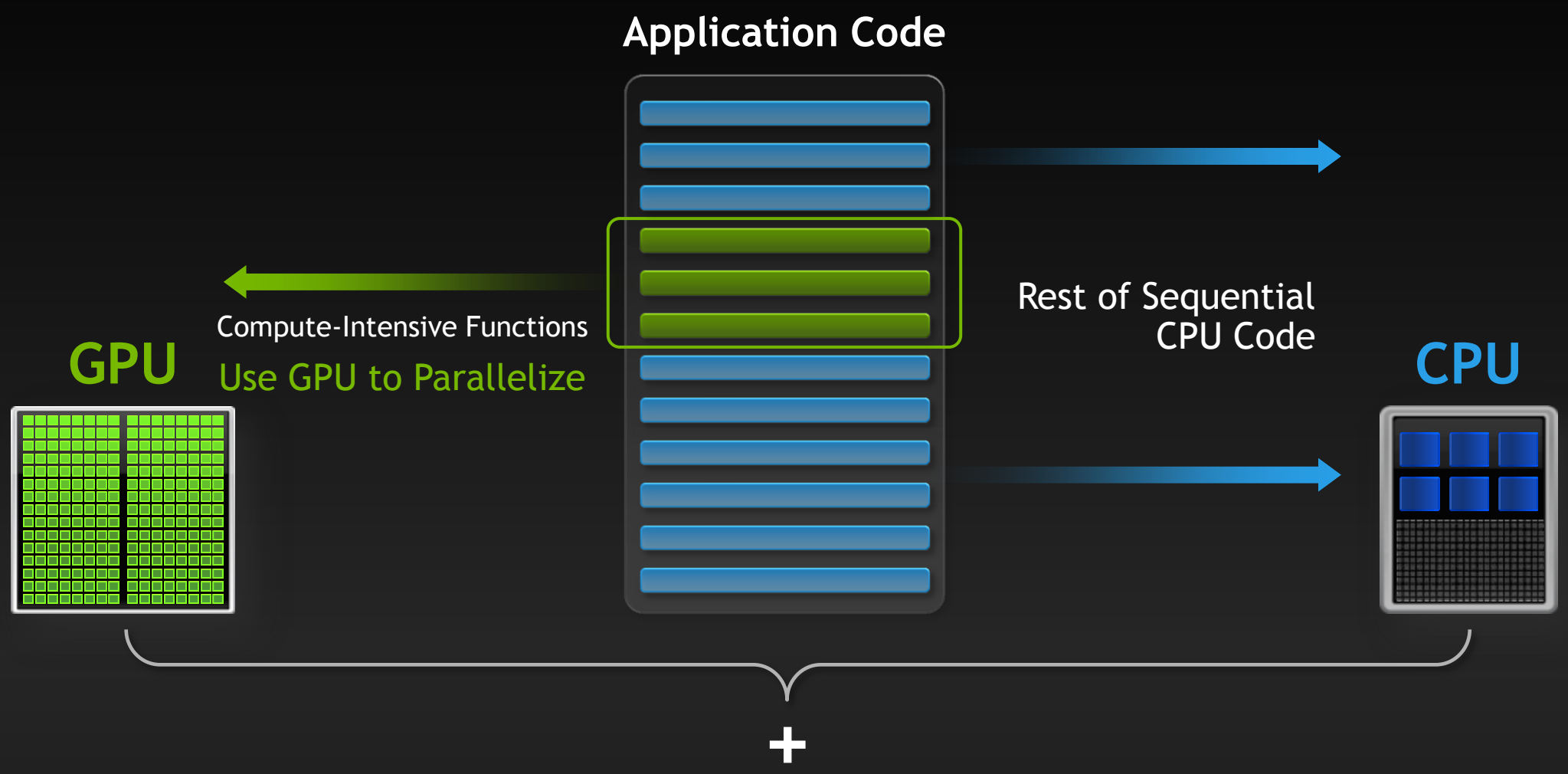


“GPUs have evolved to the point where many real-world applications are easily implemented on them and run significantly faster than on multi-core systems.

Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs.”

Jack Dongarra
Professor, University of Tennessee
Director of the Innovative Computing Laboratory
Author of LINPACK

Small Changes, Big Speed-up



“In testing our key applications, the Tesla GPUs delivered speed-ups that we had never seen before, sometimes even orders of magnitude.”

Satoshi Matsuoka
Professor, Global Scientific Information and Computing Center,
Tokyo Institute of Technology
Technical Lead for the TSUBAME Supercomputer

3 Ways to Accelerate Applications

Applications

Libraries	OpenACC Directives	Programming Languages
"Drop-in" Acceleration	Easily Accelerate Applications	Maximum Flexibility

GPU Accelerated Libraries

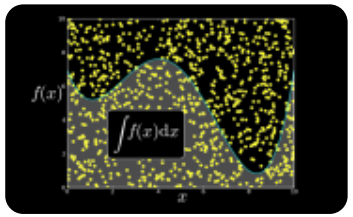
“Drop-in” Acceleration for your Applications

NVIDIA cuFFT

NVIDIA cuBLAS

NVIDIA cuSPARSE

NVIDIA NPP



NVIDIA cuRAND

MAGMA

Matrix Algebra on GPU and Multicore

CULA | tools

GPU Accelerated Linear Algebra

GPU VSIPL

Vector Signal Image Processing

ROGUE WAVE SOFTWARE

IMSL Library

CUSP

Sparse Linear Algebra

ArrayFire

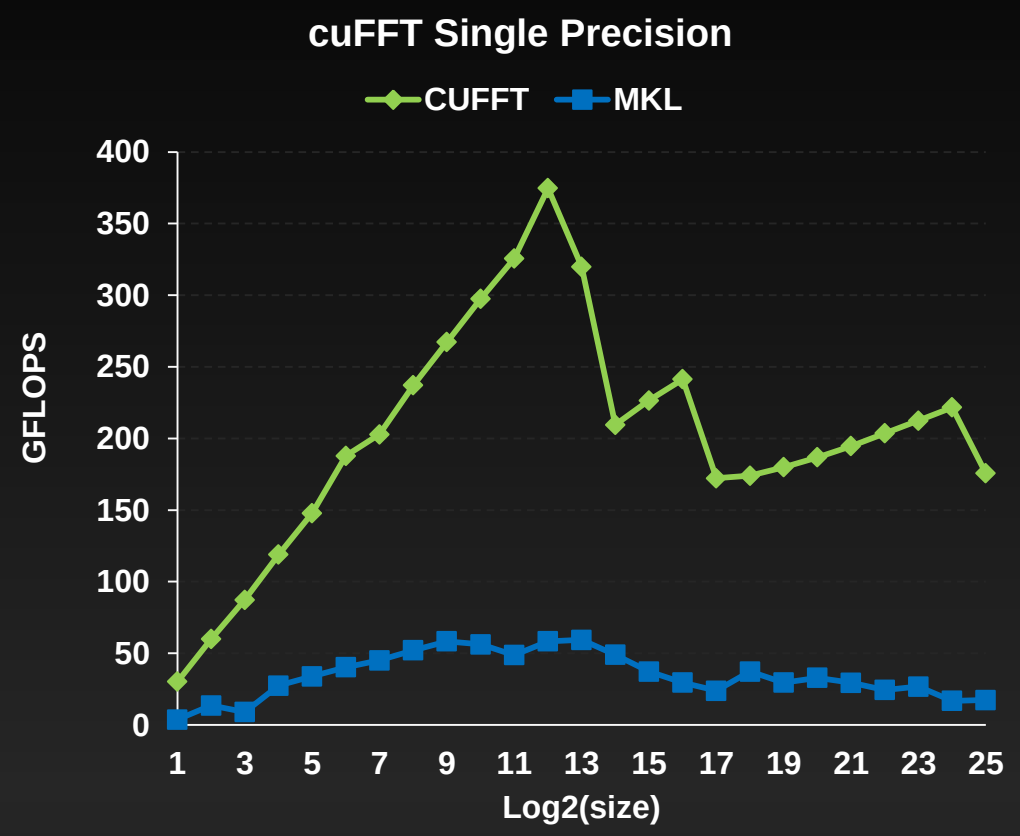
Building-block Algorithms

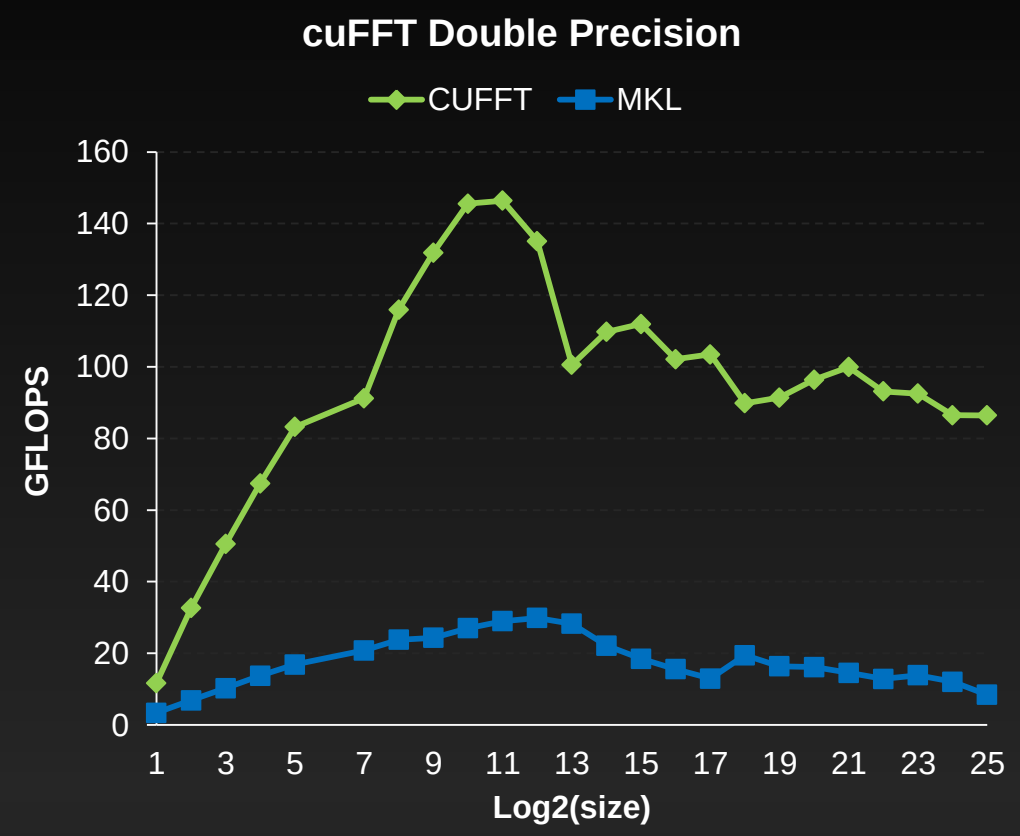
Thrust

C++ Templated Parallel Algorithms

FFTs up to 10x Faster than MKL

1D used in audio processing and as a foundation for 2D and 3D FFTs



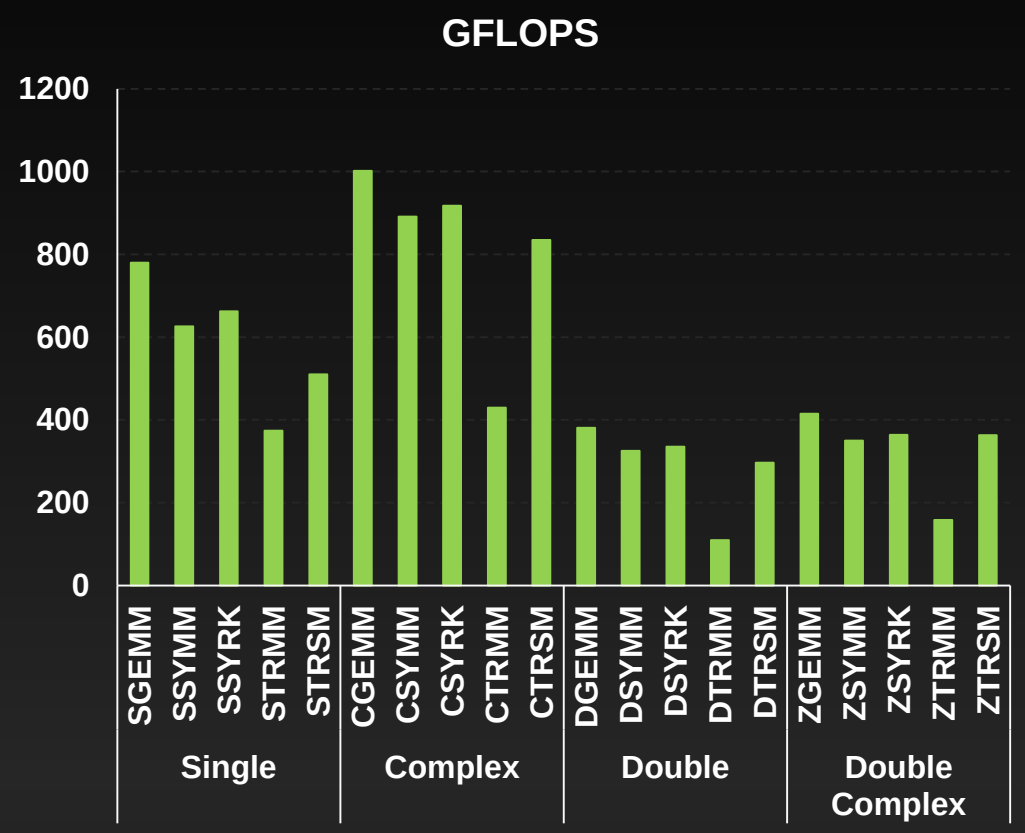


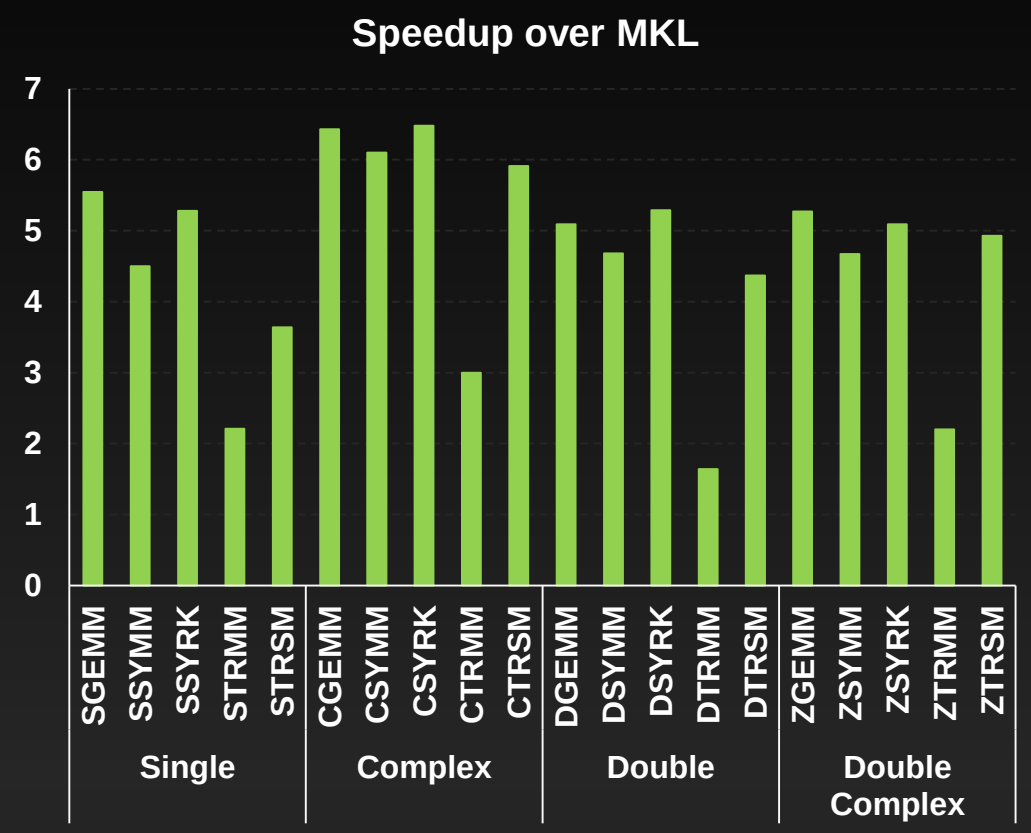
- Measured on sizes that are exactly powers-of-2
- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

Performance may vary based on OS version and motherboard configuration

cuBLAS Level 3 Performance

Up to 1 TFLOPS sustained performance and **>6x** speedup over Intel MKL

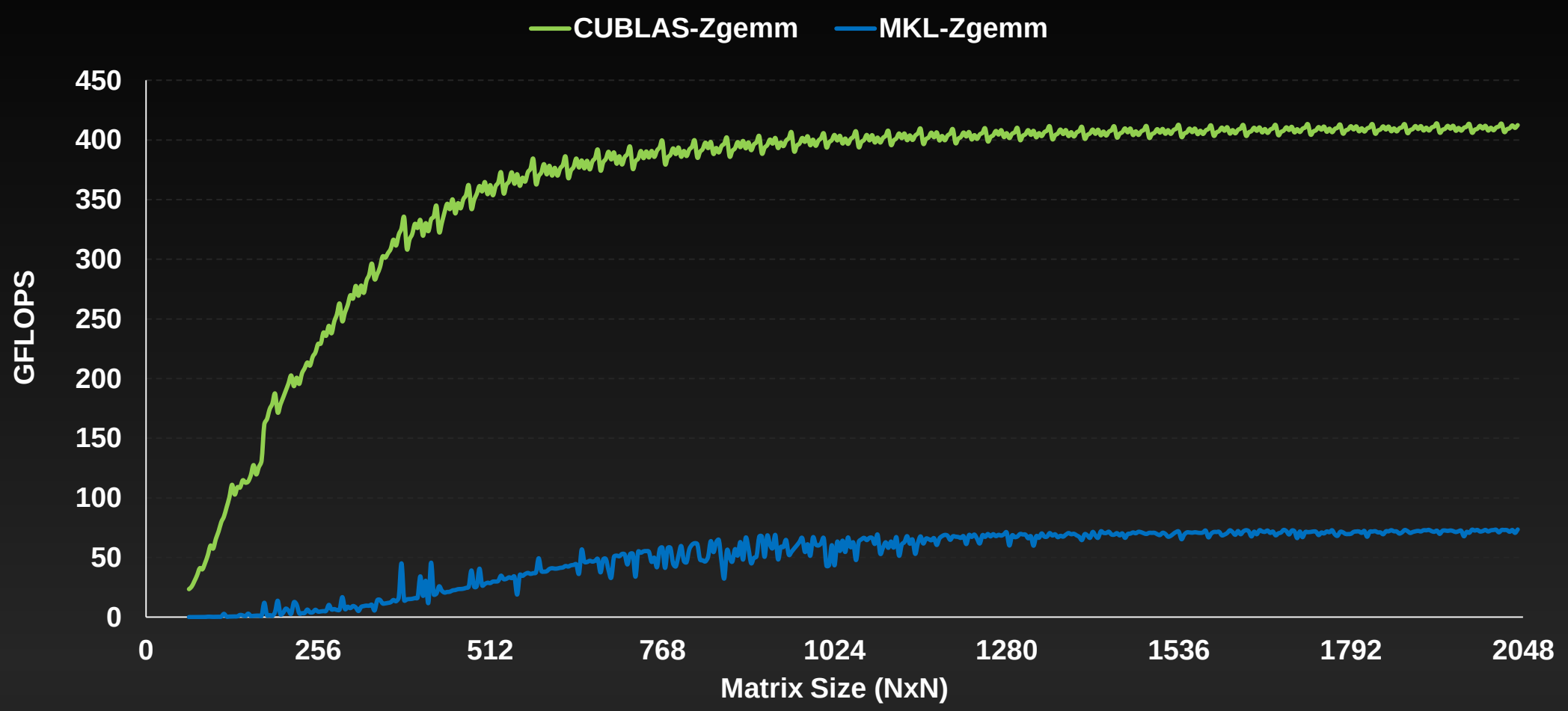




- 4Kx4K matrix size
- cuBLAS 4.1, Tesla M2090 (Fermi), ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

Performance may vary based on OS version and motherboard configuration

ZGEMM Performance vs. Intel MKL



Performance may vary based on OS version and motherboard configuration

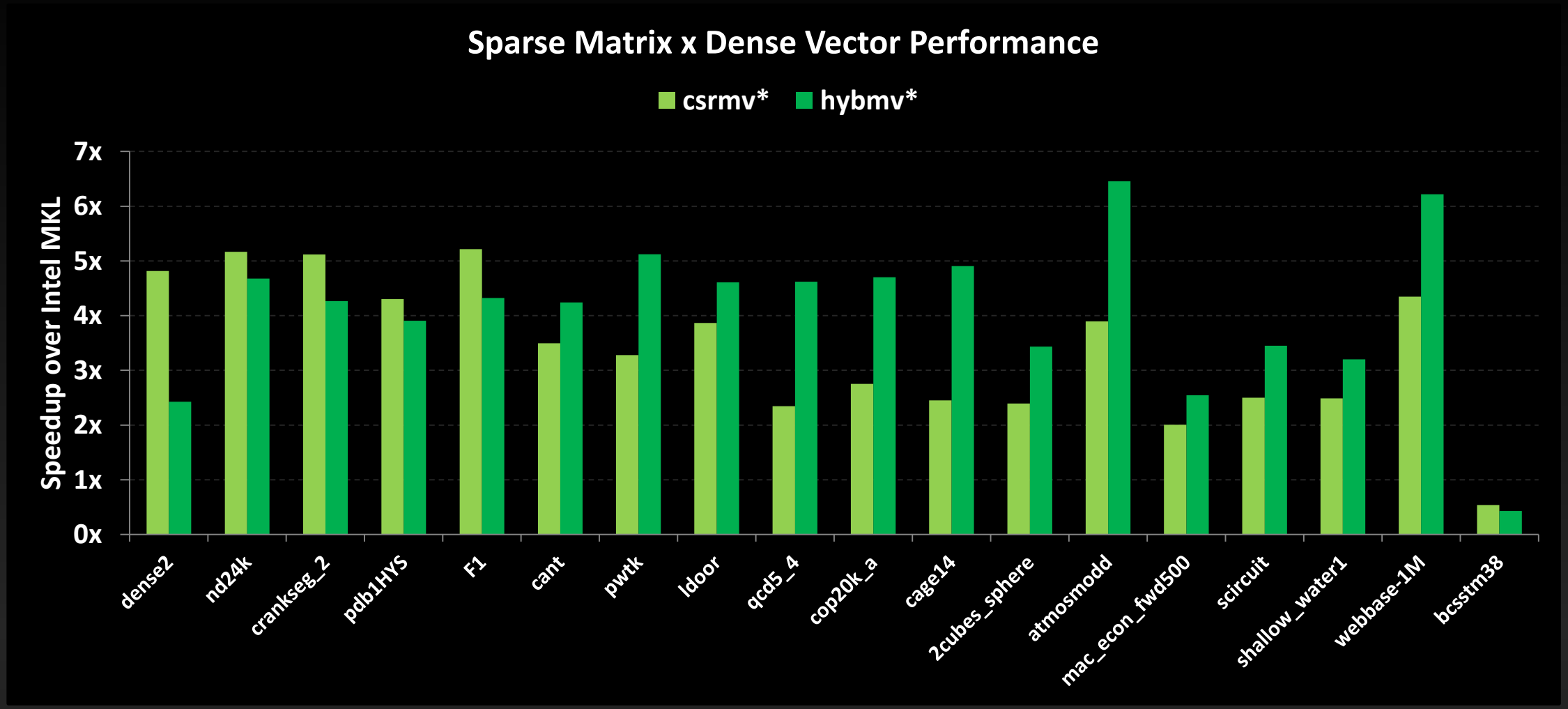
- cuBLAS 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

cuSPARSE: Sparse linear algebra routines

- Sparse matrix-vector multiplication & triangular solve
  - APIs optimized for iterative methods
  - Tri-diagonal solver 10x faster vs. Intel MKL
  - ELL-HYB format offers 2x faster matrix-vector multiplication

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \alpha \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix} + \beta \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

cuSPARSE is >6x Faster than Intel MKL



*Average speedup over single, double, single complex & double-complex

Performance may vary based on OS version and motherboard configuration

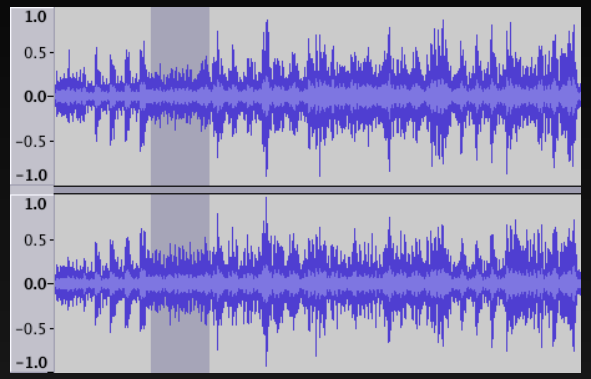
- cuSPARSE 4.1, Tesla M2090 (Fermi), ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

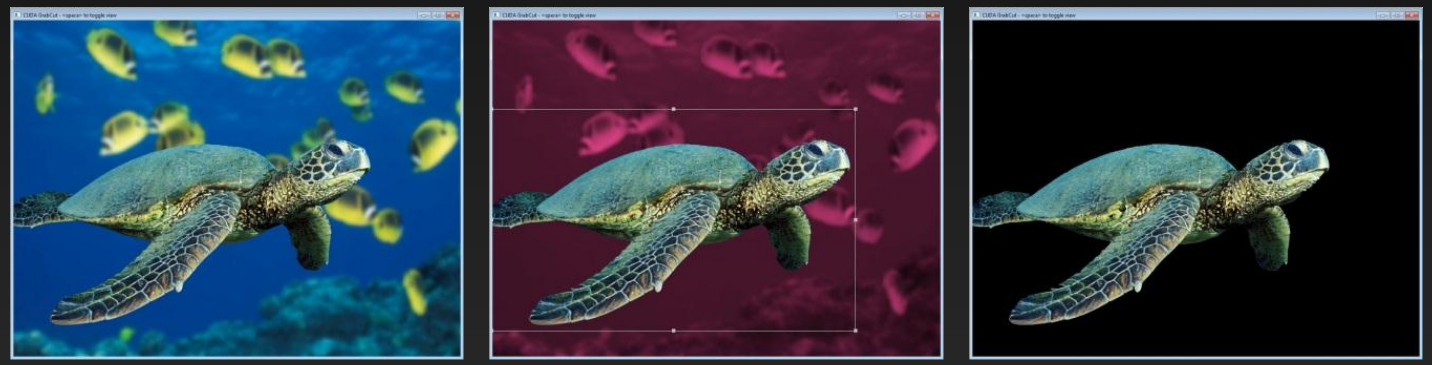
1000+ New Imaging Functions in NPP

- NVIDIA Performance Primitives (NPP) library includes over 2200 GPU-accelerated functions for image & signal processing

  Arithmetic, Logic, Conversions, Filters, Statistics, etc.

- Up to 40x faster performance than Intel IPP





developer.nvidia.com/content/graphcuts-using-npp

* NPP 4.1, NVIDIA C2050 (Fermi)
* IPP 6.1, Dual Socket Core™ i7 920 @ 2.67GHz

GPU-Aware MPI Libraries

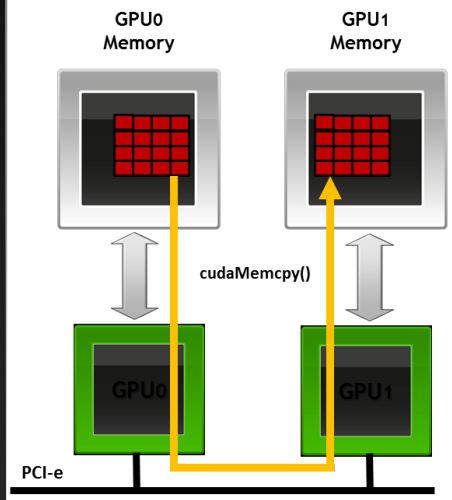
Integrated Support for GPU Computing

OpenMPI

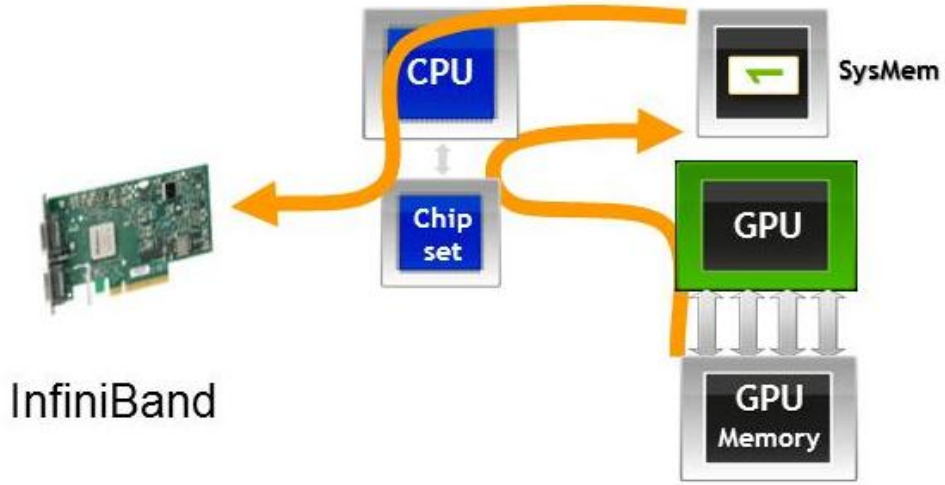
MVAPICH

Platform Computing

Platform MPI



Peer-to-Peer Transfers



Accelerated Communication

developer.nvidia.com/gpudirect

3 Ways to Accelerate Applications

Applications

Libraries	OpenACC Directives	Programming Languages
"Drop-in" Acceleration	Easily Accelerate Applications	Maximum Flexibility

OpenACC

Open Programming Standard for Parallel Computing

"OpenACC will enable programmers to easily develop portable applications that maximize the performance and power efficiency benefits of the hybrid CPU/GPU architecture of Titan."

--Buddy Bland, Titan Project Director, Oak Ridge National Lab

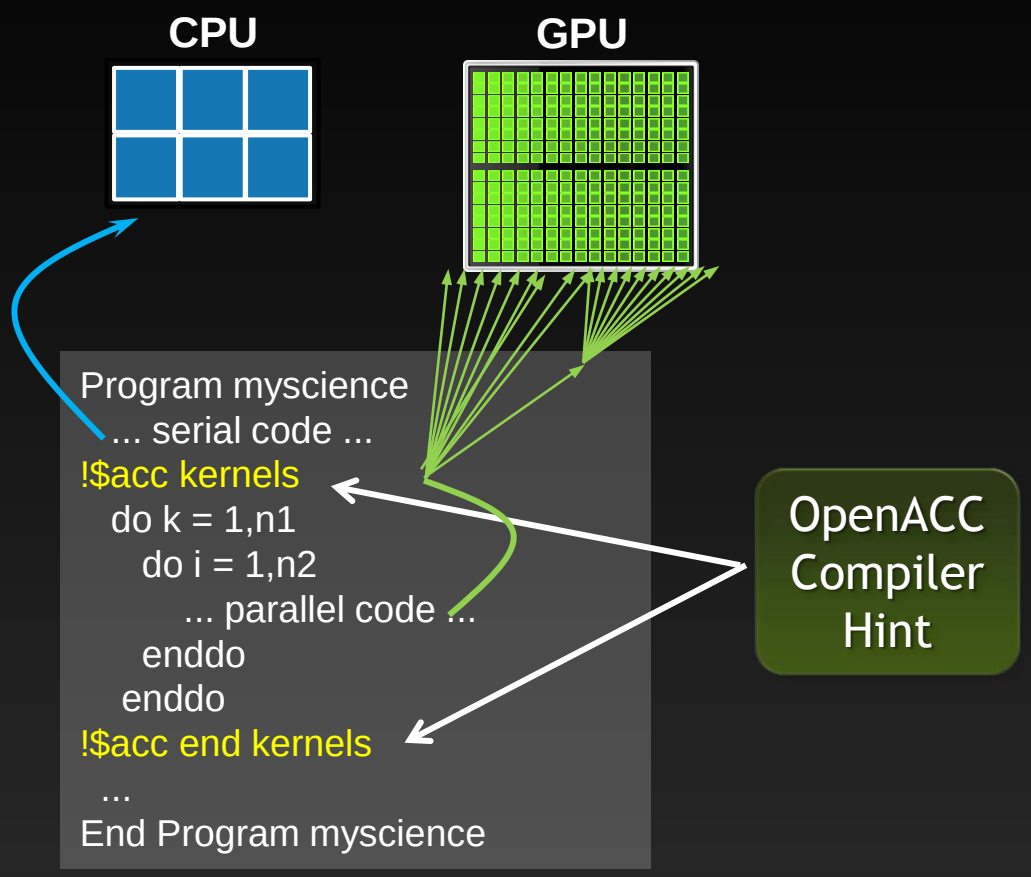
"OpenACC is a technically impressive initiative brought together by members of the OpenMP Working Group on Accelerators, as well as many others. We look forward to releasing a version of this proposal in the next release of OpenMP."

--Michael Wong, CEO OpenMP Directives Board

OpenACC Standard

NVIDIA CRAY THE SUPERCOMPUTER COMPANY PGI CAPS

OpenACC Directives



Your original Fortran or C code

Simple Compiler hints

Compiler Parallelizes code

Works on many-core GPUs & multicore CPUs

OpenACC®
DIRECTIVES FOR ACCELERATORS

www.nvidia.com/gpudirectives

Directives: Easy & Powerful

Real-Time Object Detection

Global Manufacturer of Navigation Systems

5x in 40 Hours

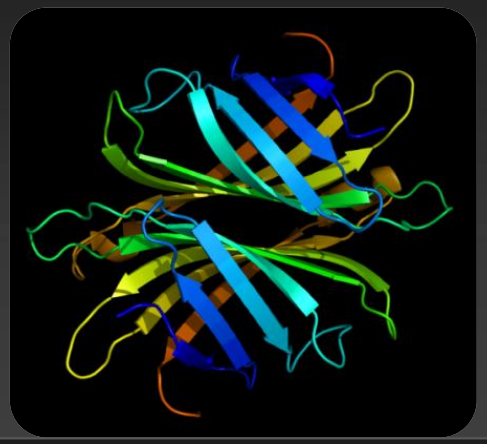
Valuation of Stock Portfolios using Monte Carlo

Global Technology Consulting Company

2x in 4 Hours

Interaction of Solvents and Biomolecules

University of Texas at San Antonio



5x in 8 Hours

"Optimizing code with directives is quite easy, especially compared to CPU threads or writing CUDA kernels. The most important thing is avoiding restructuring of existing code for production applications."

-- Developer at the Global Manufacturer of Navigation Systems

3 Ways to Accelerate Applications

Applications

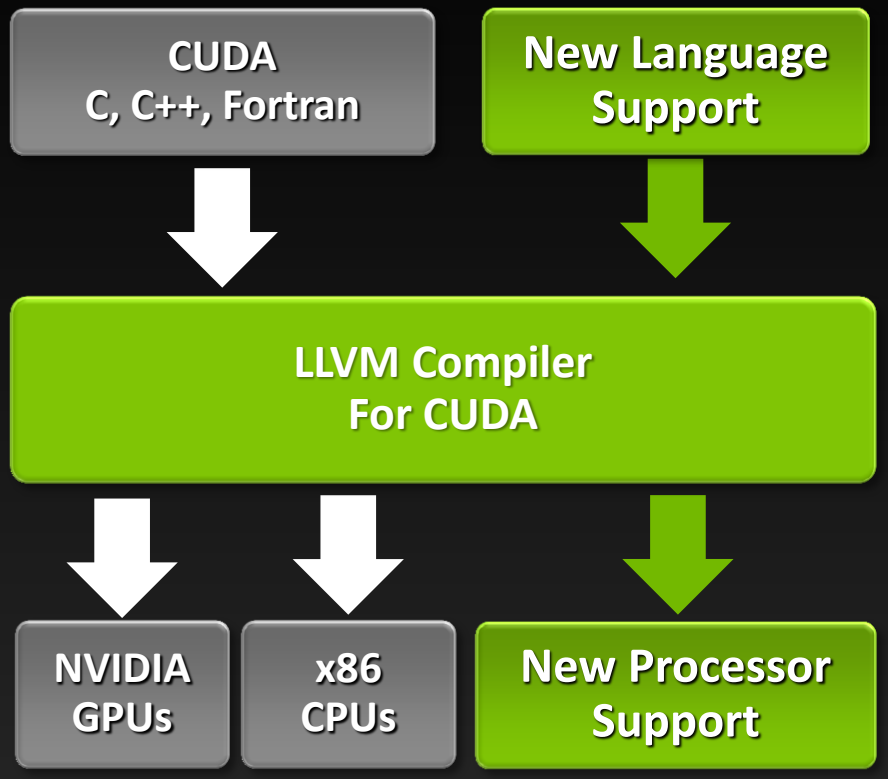
Libraries	OpenACC Directives	Programming Languages
"Drop-in" Acceleration	Easily Accelerate Applications	Maximum Flexibility

Opening the CUDA Platform with LLVM

CUDA compiler source contributed
to open source LLVM compiler project

SDK includes specification documentation,
examples, and verifier

Provides ability for anyone to add CUDA to new languages and processors



Learn more at
developer.nvidia.com/cuda-source

LLVM
COMPILER
INFRASTRUCTURE

CUDA C

Standard C 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];
}

// Perform SAXPY on 1M elements
saxpy_serial(4096*256, 2.0, x, y);
```

Parallel 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];
}

// Perform SAXPY on 1M elements
saxpy_parallel<<<4096,256>>>(n,2.0,x,y);
```

developer.nvidia.com/cuda-toolkit

CUDA C++: Develop Generic Parallel Code

CUDA C++ features enable sophisticated and flexible applications and middleware

- Class hierarchies
- __device__ methods
- Templates
- Operator overloading
- Functors (function objects)
- Device-side new/delete
- More…

```
template <typename T>
struct Functor {
  __device__ Functor(_a) : a(_a) {}
  __device__ T operator(T x) { return a*x; }
  T a;
}

template <typename T, typename Oper>
__global__ void kernel(T *output, int n) {
  Oper op(3.7);
  output = new T[n]; // dynamic allocation
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n)
    output[i] = op(i);  // apply functor
}
```

developer.nvidia.com/cuda-toolkit

Thrust: Rapid Parallel C++ Development

- Resembles C++ STL
- High-level interface
  - Enhances developer productivity
  - Enables performance portability between GPUs and multicore CPUs
- Flexible
  - Backends for CUDA, OpenMP, TBB
  - Extensible and customizable
  - Integrates with existing software
- Open source

```
// generate 32M random numbers on host
thrust::host_vector<int> h_vec(32 << 20);
thrust::generate(h_vec.begin(),
                 h_vec.end(),
                 rand);

// transfer data to device (GPU)
thrust::device_vector<int> d_vec = h_vec;

// sort data on device
thrust::sort(d_vec.begin(), d_vec.end());

// transfer data back to host
thrust::copy(d_vec.begin(),
             d_vec.end(),
             h_vec.begin());
```

developer.nvidia.com/thrust or thrust.googlecode.com

CUDA Fortran

- Program GPU using Fortran
  - Key language for HPC
- Simple language extensions
  - Kernel functions
  - Thread / block IDs
  - Device & data management
  - Parallel loop directives
- Familiar syntax
  - Use allocate, deallocate
  - Copy CPU-to-GPU with assignment (=)

```
module mymodule contains
  attributes(global) subroutine saxpy(n,a,x,y)
    real :: x(:), y(:), a,
    integer n, i
    attributes(value) :: a, n
    i = threadIdx%x+(blockIdx%x-1)*blockDim%x
    if (i<=n) y(i) = a*x(i) + y(i);
  end subroutine saxpy
end module mymodule

program main
  use cudafor; use mymodule
  real, device :: x_d(2**20), y_d(2**20)
  x_d = 1.0; y_d = 2.0
  call saxpy<<<4096,256>>>(2**20,3.0,x_d,y_d,)
  y = y_d
  write(*,*) 'max error=', maxval(abs(y-5.0))
end program main
```

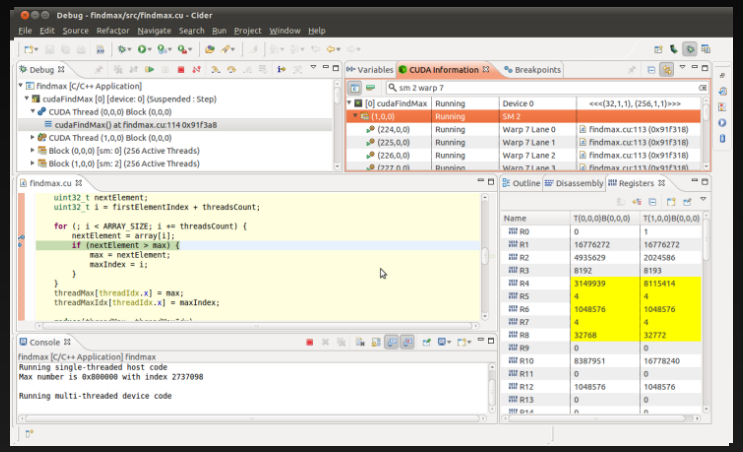
developer.nvidia.com/cuda-fortran

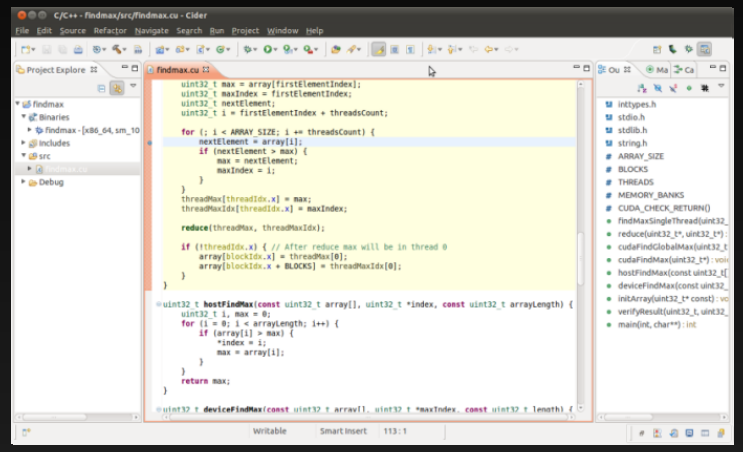
GPU Programming Languages

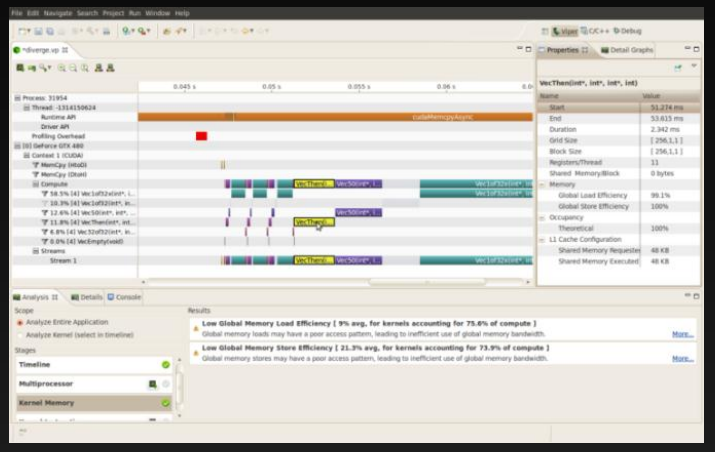
Numerical analytics	MATLAB, Mathematica, LabVIEW
Fortran	OpenACC, CUDA Fortran
C	OpenACC, CUDA C
C++	Thrust, CUDA C++
Python	PyCUDA
C#	GPU.NET

Development Tools

NVIDIA® Nsight™ Eclipse Edition
for Linux and MacOS







CUDA-Aware Editor

- Automated CPU to GPU code refactoring
- Semantic highlighting of CUDA code
- Integrated code samples & docs

Nsight Debugger

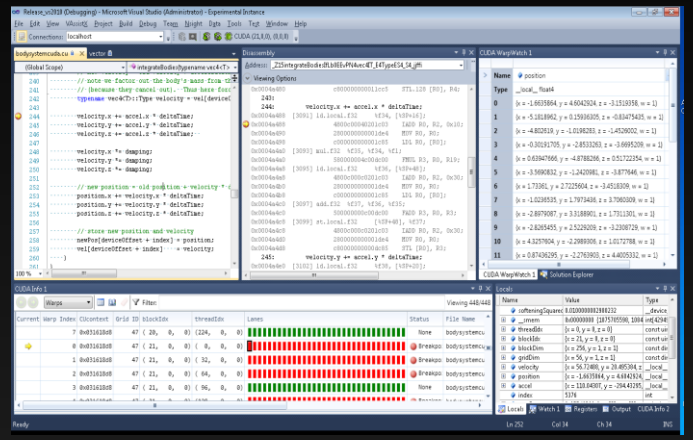
- Simultaneously debug CPU and GPU
- Inspect variables across CUDA threads
- Use breakpoints & single-step debugging

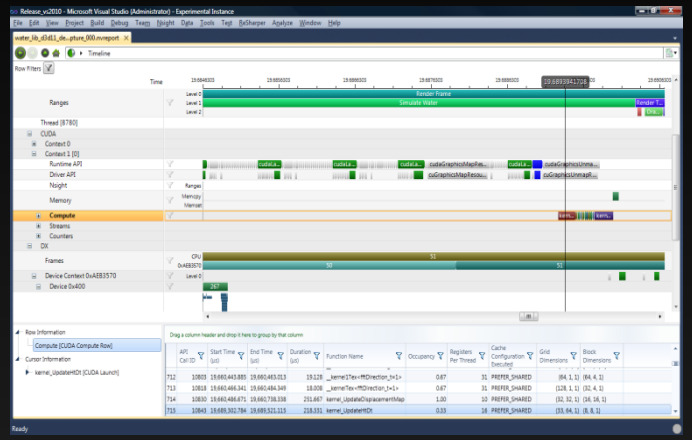
Nsight Profiler

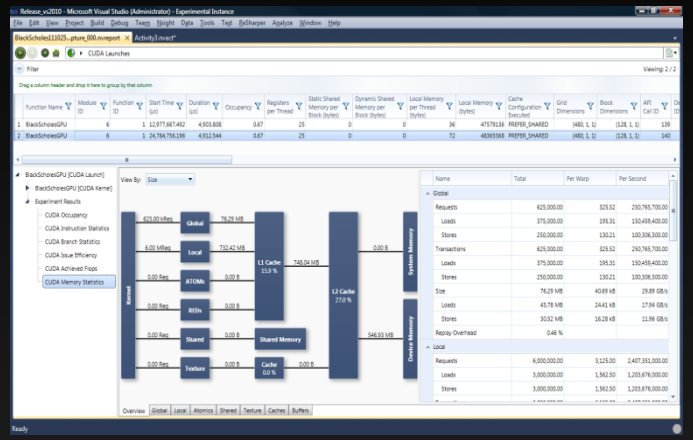
- Quickly identifies performance issues
- Integrated expert system
- Source line correlation

developer.nvidia.com/nsight

NVIDIA® Nsight™, Visual Studio Ed.







CUDA Debugger

- Debug CUDA kernels directly on GPU hardware
- Examine thousands of threads executing in parallel
- Use on-target conditional breakpoints to locate errors

CUDA Memory Checker

- Enables precise error detection

System Trace

- Review CUDA activities across CPU and GPU
- Perform deep kernel analysis to detect factors limiting maximum performance

CUDA Profiler

- Advanced experiments to measure memory utilization, instruction throughput and stalls

Debugging Solutions

Command Line to Cluster-Wide

NVIDIA Nsight
Eclipse & Visual Studio Editions

NVIDIA CUDA-GDB
for Linux & Mac

NVIDIA CUDA-MEMCHECK
for Linux & Mac

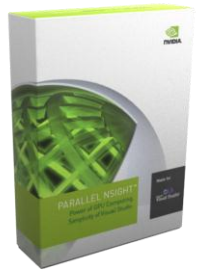
Allinea DDT with CUDA
Distributed Debugging Tool

TotalView for CUDA
for Linux Clusters

developer.nvidia.com/debugging-solutions

Performance Analysis Tools

Single Node to Hybrid Cluster Solutions



NVIDIA Nsight
Eclipse & Visual Studio Editions

NVIDIA Visual Profiler

Vampir Trace Collector

Tuning and Analysis Utilities

TAU Performance System

PAPI

PAPI CUDA Component

HPCToolkit

Under Development

developer.nvidia.com/performance-analysis-tools

Job Scheduling & Cluster Management

Platform Computing

LSF, HPC, Cluster Manager

Bright Computing

Bright Cluster Manager

Adaptive COMPUTING

ROCKS+ MOAB

PBS Works™

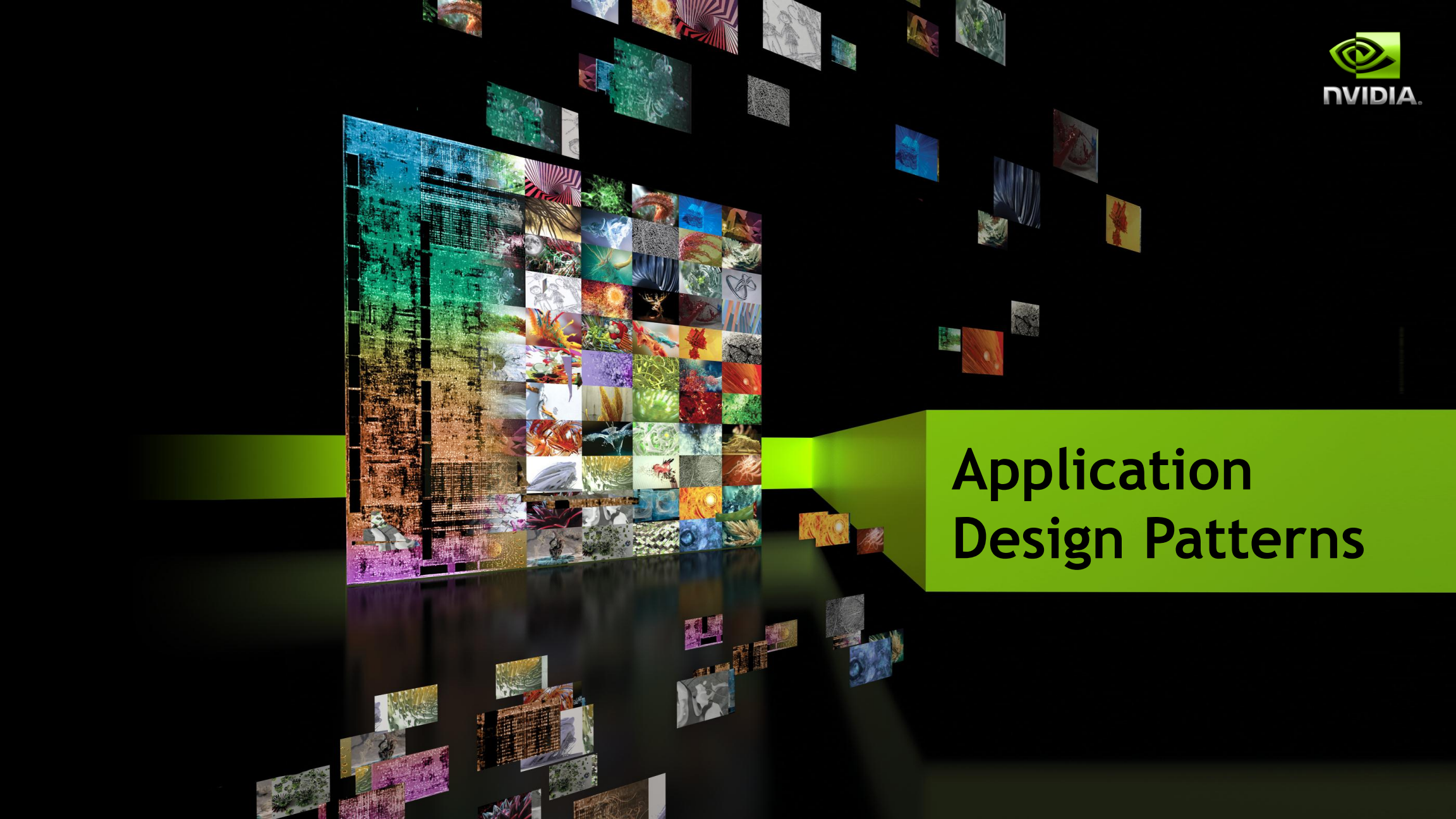
PBS Professional

Ganglia

NVML Plugin for GPUs

UNIVA

Univa Grid Engine



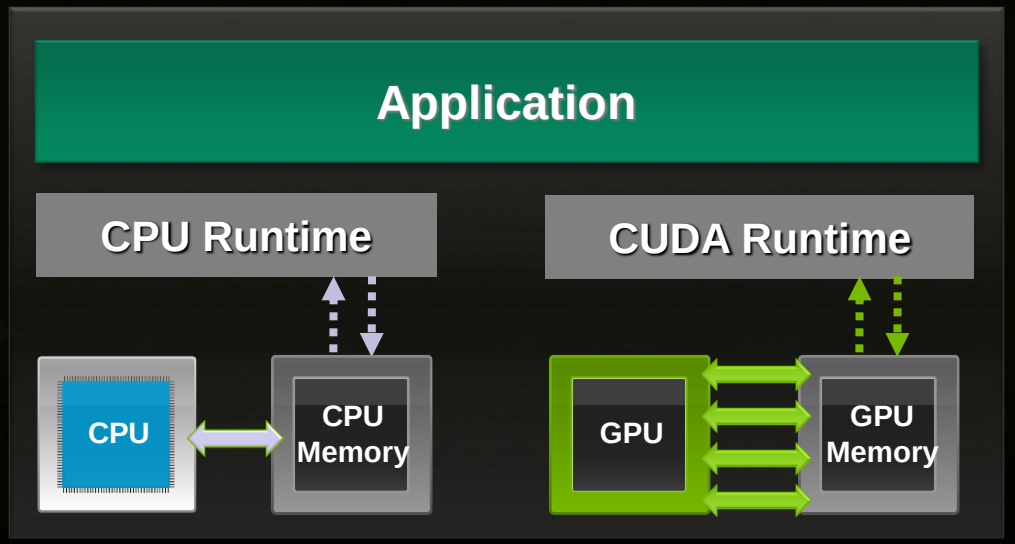
Application Design Patterns

Trivial Application

Design Rules:

- Serial task processing on CPU
- Data Parallel processing on GPU
  - Copy input data to GPU
  - Perform parallel processing
  - Copy results back

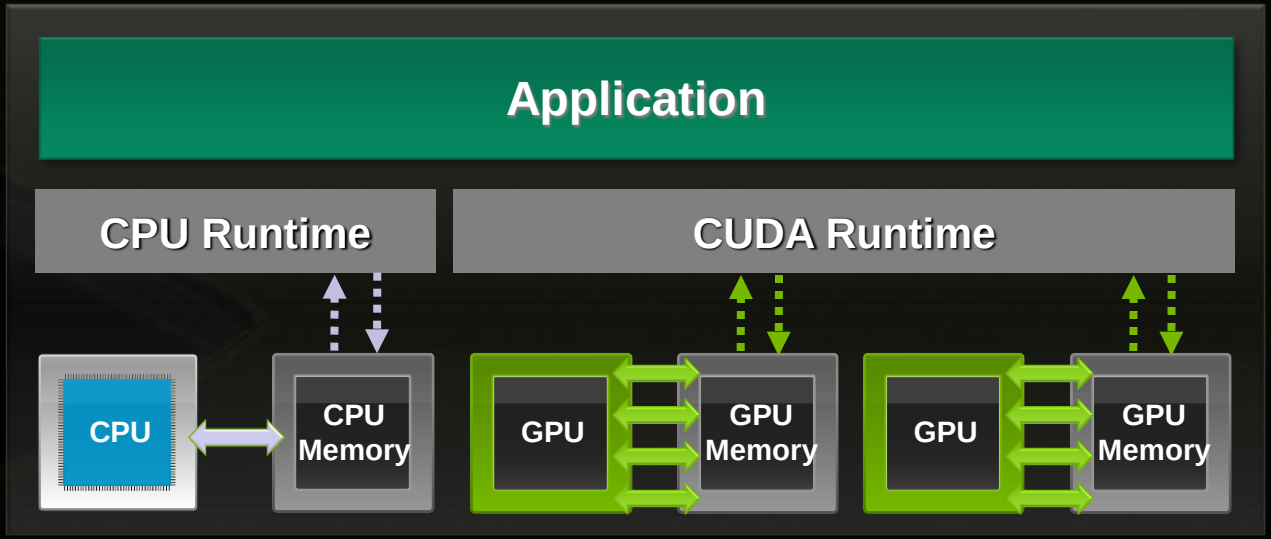
- Follow guidance in the CUDA C Best Practices Guide



Basic Application

"Trivial Application" plus:

- **Maximize overlap of data transfers and computation**
- **Minimize communication required between processors**
- **Use peer-to-peer communication between GPUs**

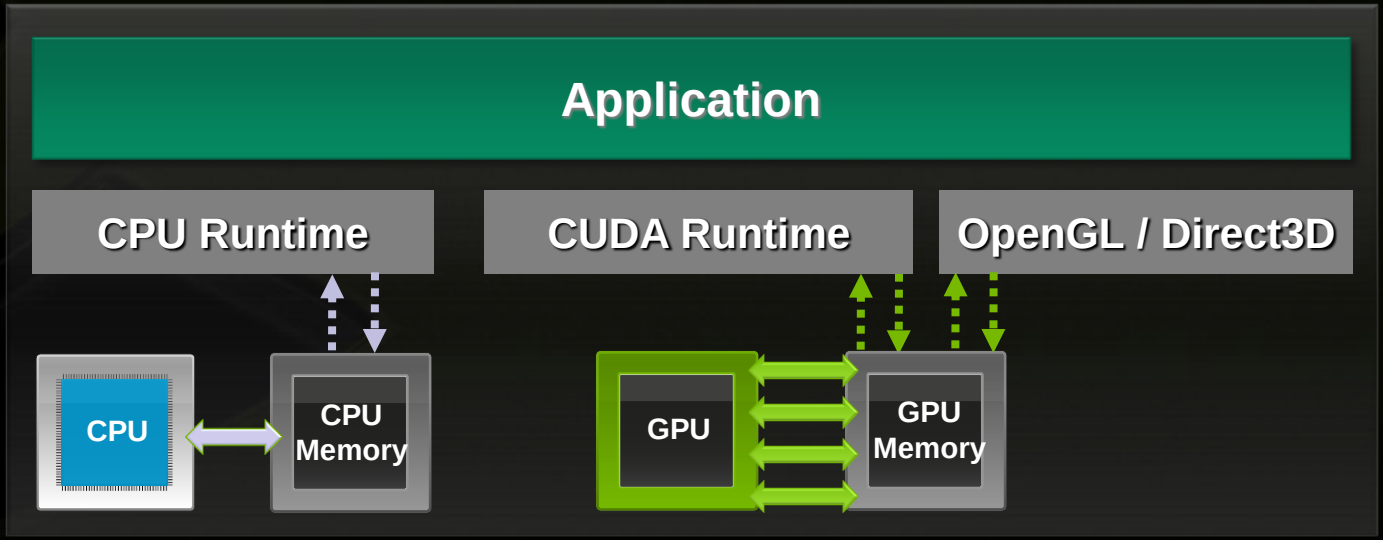


Multi-GPU notebook, desktop, workstation and cluster node configurations are common

Graphics Application

"Basic Application" plus:

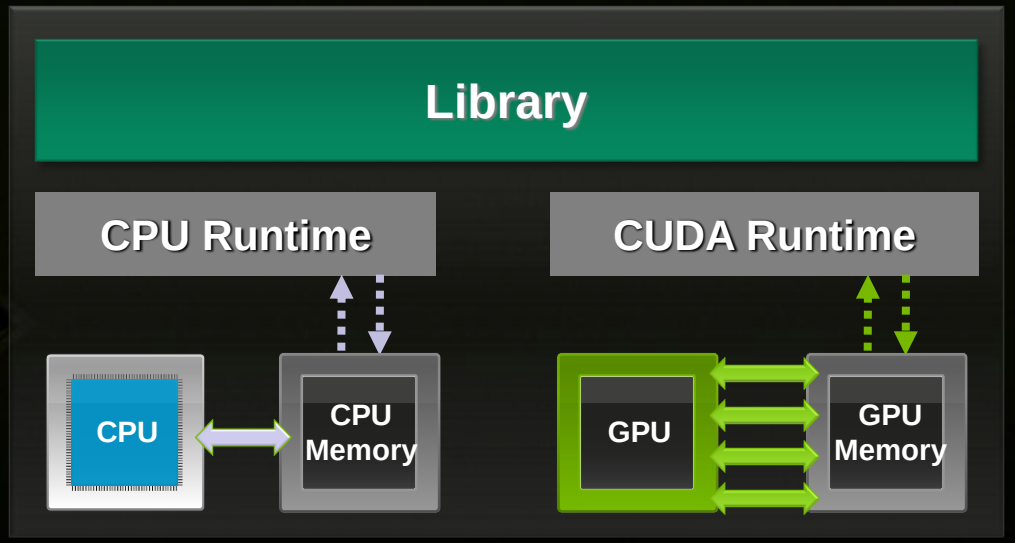
- Use graphics interop to avoid unnecessary copies
- In Multi-GPU systems, put buffers to be displayed in GPU Memory of GPU attached to the display



Basic Library

"Basic Application" plus:

- Avoid unnecessary memory transfers
  - Use data already in GPU memory
  - Create and leave data in GPU memory

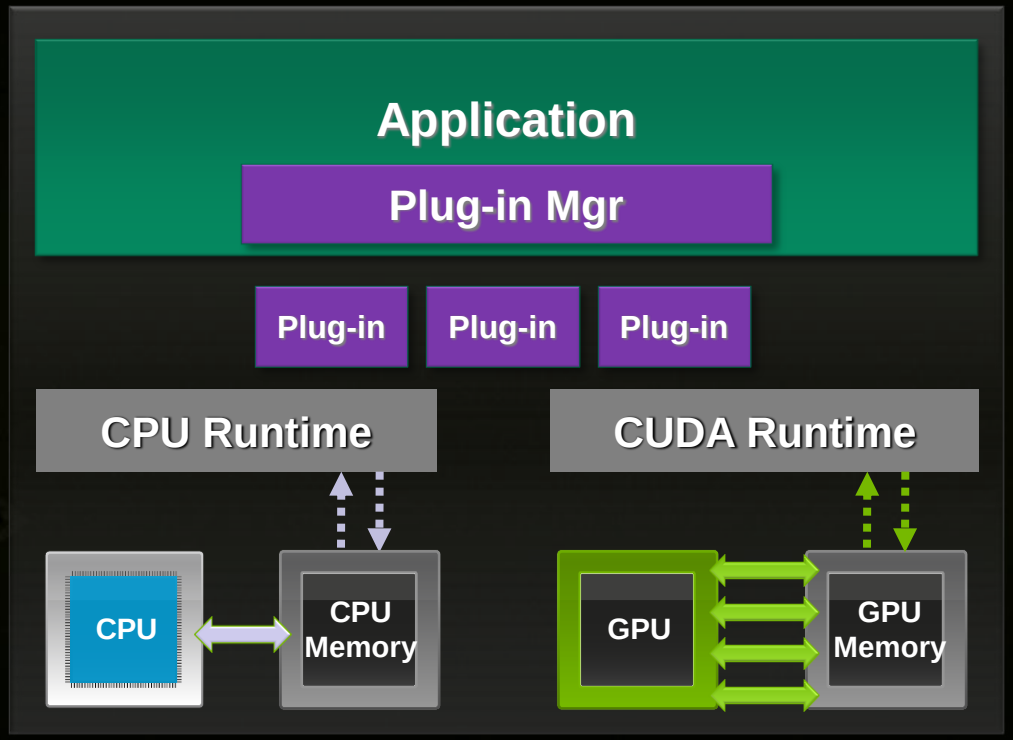


These rules apply to plug-ins as well

Application with Plug-ins

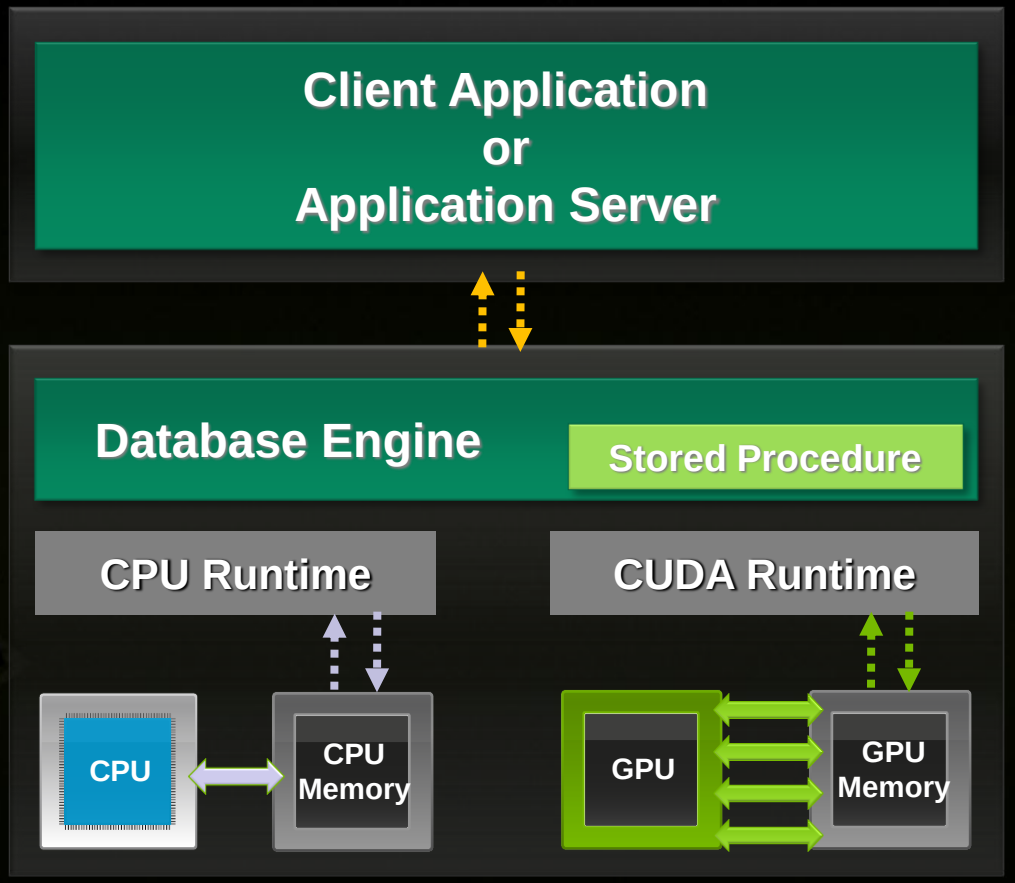
"Basic Application" plus:

- 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



Database Application

- Minimize network communication
- Move analysis "upstream" to stored procedures
- Treat each stored procedure like a "Basic Application"
- App Server could also be a "Basic Application"
- Client Application is also a "Basic Application"

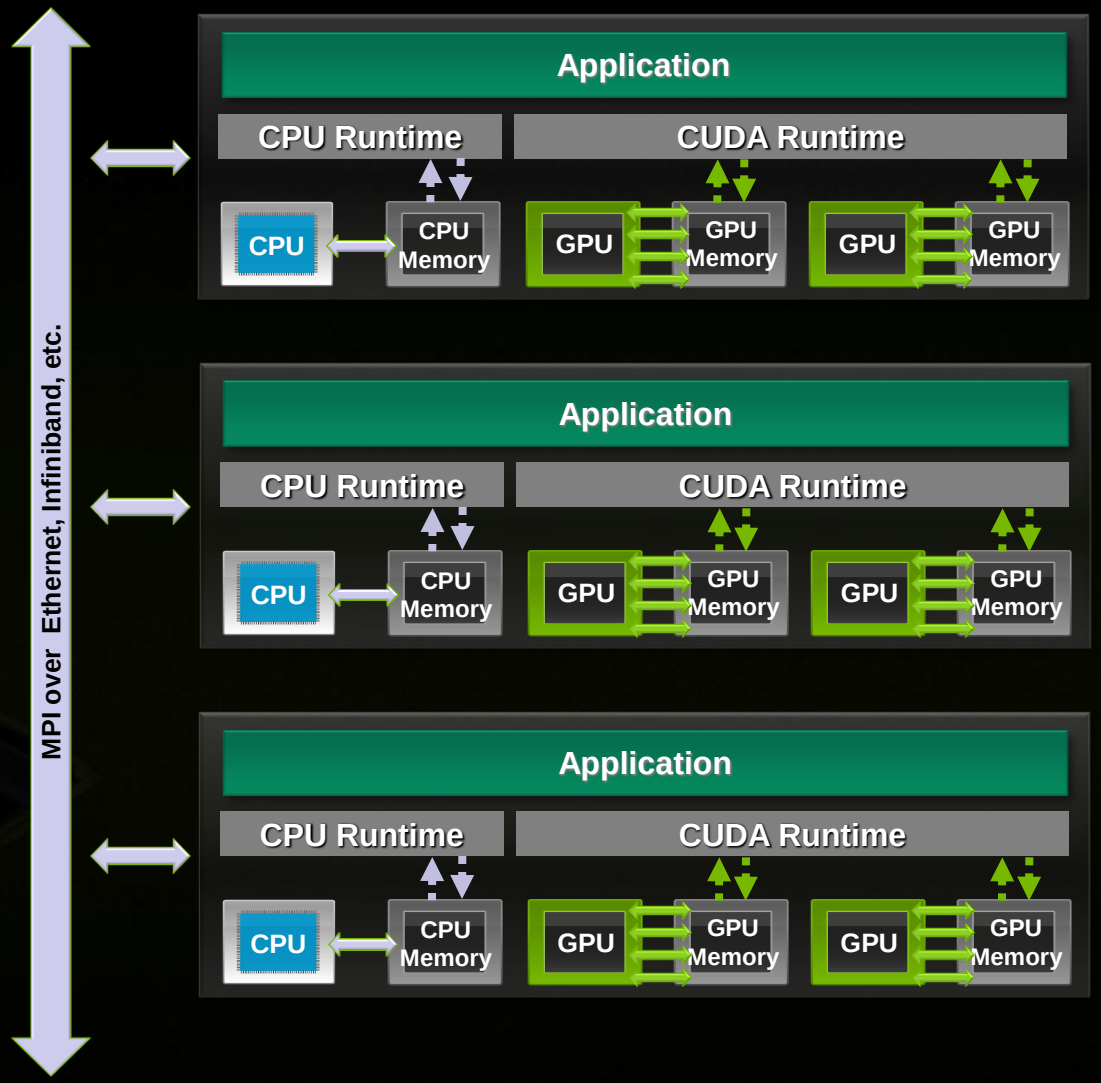


Data Mining, Business Intelligence, etc.

Multi-GPU Cluster Application

"Basic Application" plus:

- Use Shared Memory for intra-node communication
  - or pthreads, OpenMP, etc.
- Use GPU-aware MPI to communicate between nodes



GPU TECHNOLOGY CONFERENCE

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

Nsight
Eclipse & Visual Studio Editions

NVPerfKit
OpenGL|D3D performance tools

FX Composer
Shader Authoring IDE

SDKs AND CODE SAMPLES

GPU Computing SDK
Code samples and docs

CUDA Compiler SDK
Specifications & examples

Graphics SDK
DirectX & OpenGL code samples

PhysX SDK
Complete game physics solution

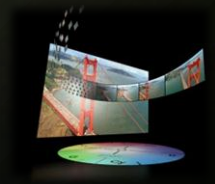
OpenAutomate
SDK for test automation

VIDEO LIBRARIES

Video Decode Acceleration
NVCUVID / NVCUVENC
DXVA
Win7 MFT

Video Encode Acceleration
NVCUVENC
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 & video

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

developer.nvidia.com