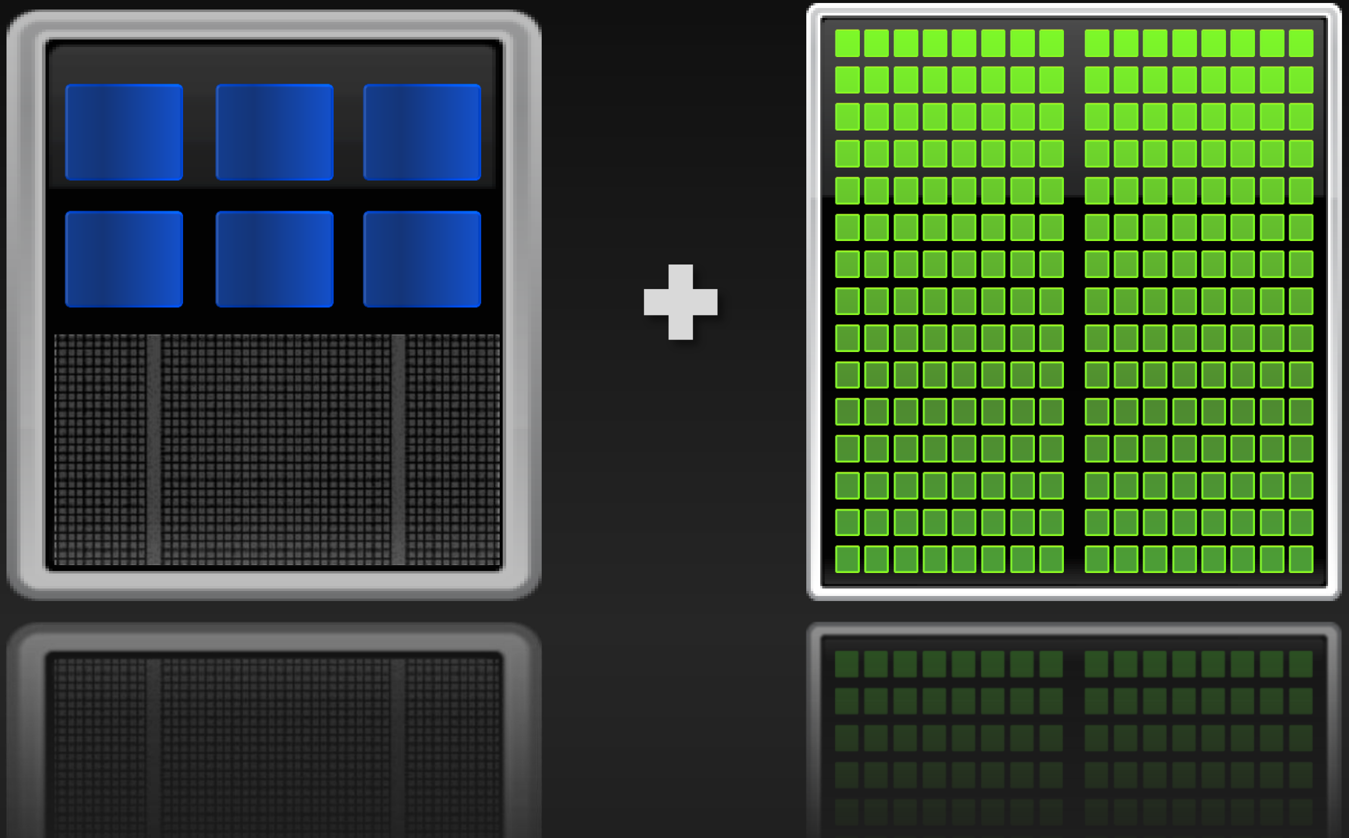
Approaches to GPU Computing

Libraries, OpenACC Directives, and Languages

Add GPUs: Accelerate Applications

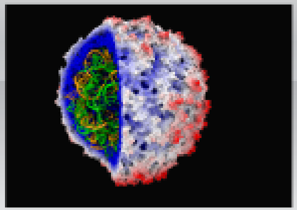


CPU

GPU

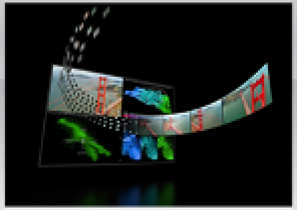
146X

Medical Imaging
U of Utah



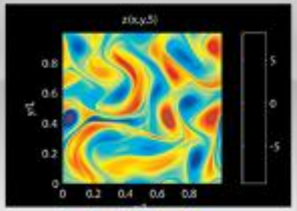
36X

Molecular Dynamics
U of Illinois, Urbana



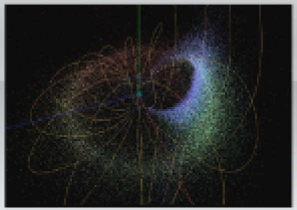
18X

Video Transcoding
Elemental Tech



50X

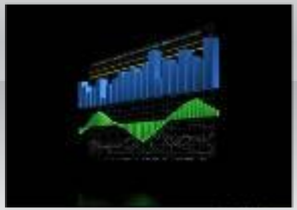
Matlab Computing
AccelerEyes



100X

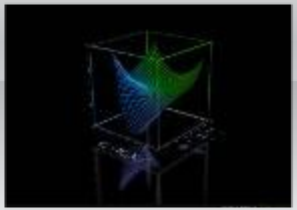
Astrophysics
RIKEN

GPUs Accelerate Science



149X

Financial Simulation
Oxford



47X

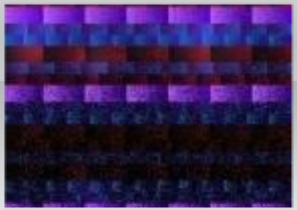
Linear Algebra
Universidad Jaime

20X

3D Ultrasound
Techniscan

130X

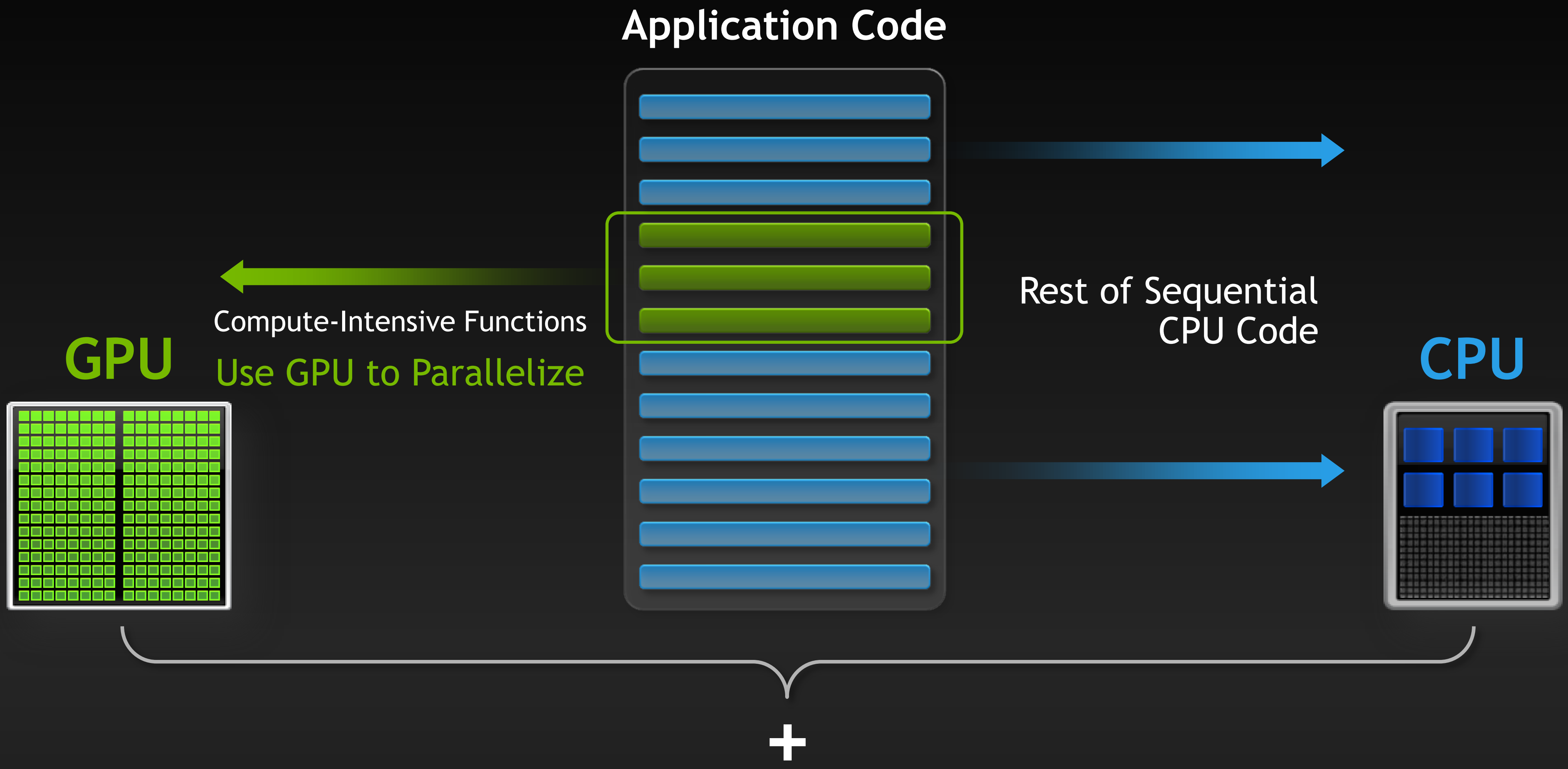
Quantum Chemistry
U of Illinois, Urbana



30X

Gene Sequencing
U of Maryland

Small Changes, Big Speed-up



Application Code

Compute-Intensive Functions

Use GPU to Parallelize

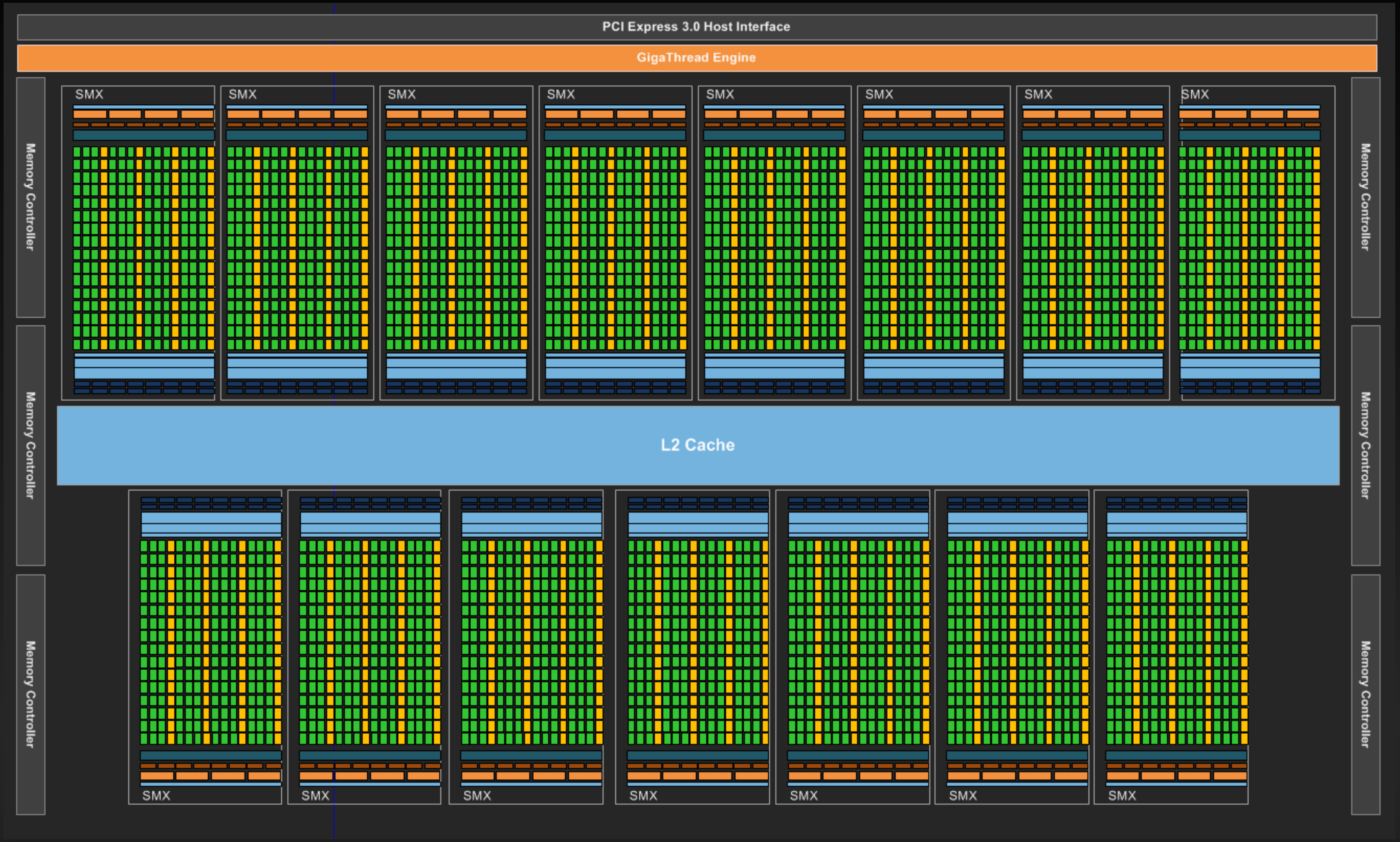
GPU

Rest of Sequential
CPU Code

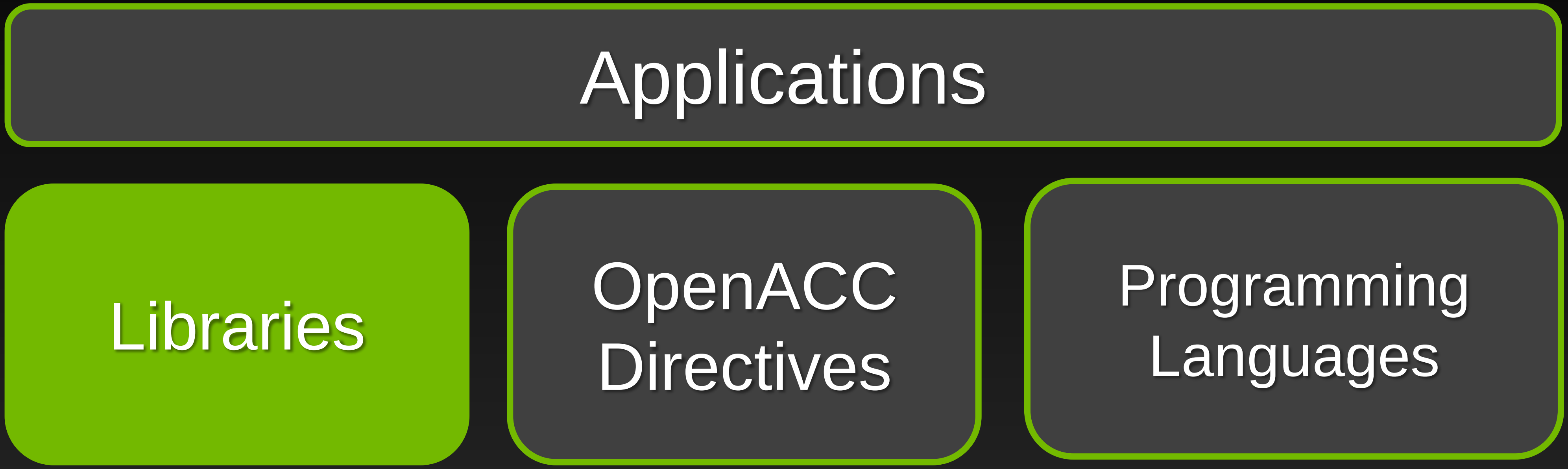
CPU

+

Why more than one approach?



3 Ways to Accelerate Applications



Applications

Libraries

OpenACC Directives

Programming Languages

"Drop-in" Acceleration

Easily Accelerate Applications

Maximum Performance

Easy, High-Quality Acceleration

- **Ease of use:** Using libraries enables GPU acceleration without in-depth knowledge of GPU programming

- **“Drop-in”:** Many GPU-accelerated libraries follow standard APIs, thus enabling acceleration with minimal code changes

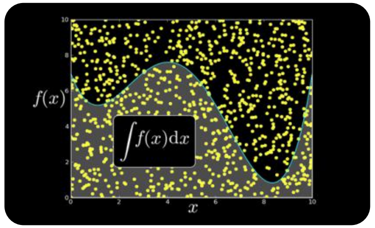
- **Quality:** Libraries offer high-quality implementations of functions encountered in a broad range of applications

- **Performance:** NVIDIA libraries are tuned by experts

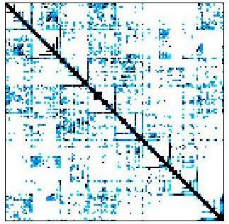
GPU Accelerated Libraries

“Drop-in” Acceleration for your Applications

NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE

NVIDIA NPP

GPU VSIPL

Vector Signal
Image Processing

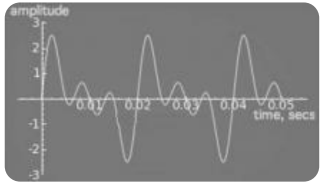
CULA | tools

GPU Accelerated
Linear Algebra

MAGMA

Matrix Algebra on
GPU and Multicore

open source initiative



NVIDIA cuFFT

ROGUE WAVE
SOFTWARE

IMSL Library

ArrayFire Matrix
Computations

CUSP

Sparse Linear
Algebra

open source initiative

Thrust

C++ STL Features
for CUDA

open source initiative

3 Steps to CUDA-accelerated application

- **Step 1:** Substitute library calls with equivalent CUDA library calls

  `saxpy ( … )` → `cublasSaxpy ( … )`

- **Step 2:** Manage data locality

  - with CUDA: `cudaMalloc(), cudaMemcpy(), etc.`
  - with CUBLAS: `cublasAlloc(), cublasSetVector(), etc.`

- **Step 3:** Rebuild and link the CUDA-accelerated library

```
nvcc myobj.o –l cublas
```

Drop-In Acceleration (Step 1)

```
int N = 1 << 20;




// Perform SAXPY on 1M elements: d_y[]=a*d_x[]+d_y[]
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);
```

Add “cublas” prefix and use device variables

Drop-In Acceleration (Step 2)

```
int N = 1 << 20;
cublasInit();
cublasAlloc(N, sizeof(float), (void**)&d_x);
cublasAlloc(N, sizeof(float), (void**)&d_y);

cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);

// Perform SAXPY on 1M elements: d_y[]=a*d_x[]+d_y[]
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);

cublasFree(d_x);
cublasFree(d_y);
cublasShutdown();
```

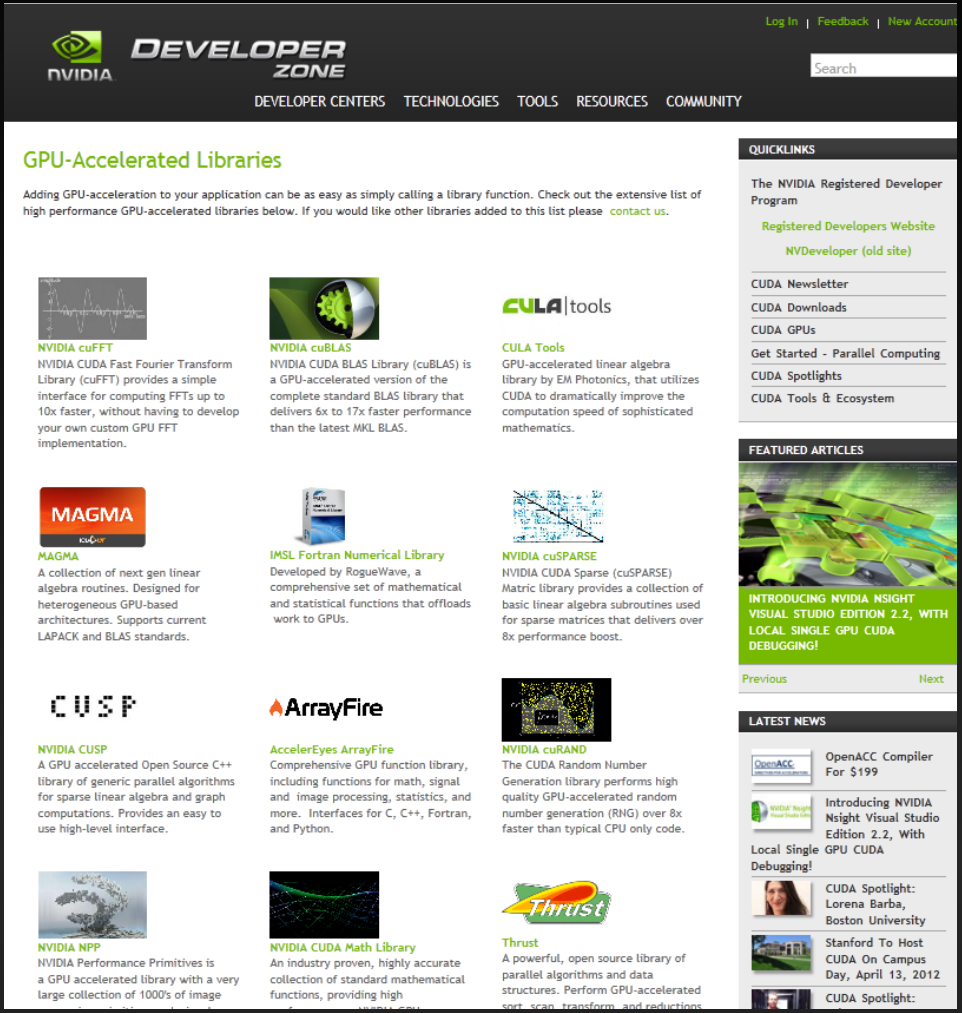
Transfer data to GPU

Read data back GPU

Explore the CUDA (Libraries) Ecosystem

- CUDA Tools and Ecosystem described in detail on NVIDIA Developer Zone:

  developer.nvidia.com/cuda-tools-ecosystem



Log In | Feedback | New Account

Search

DEVELOPER CENTERS TECHNOLOGIES TOOLS RESOURCES COMMUNITY

GPU-Accelerated Libraries

Adding GPU-acceleration to your application can be as easy as simply calling a library function. Check out the extensive list of high performance GPU-accelerated libraries below. If you would like other libraries added to this list please contact us.

NVIDIA cuFFT
NVIDIA CUDA Fast Fourier Transform Library (cuFFT) provides a simple interface for computing FFTs up to 10x faster, without having to develop your own custom GPU FFT implementation.

NVIDIA cuBLAS
NVIDIA CUDA BLAS Library (cuBLAS) is a GPU-accelerated version of the complete standard BLAS library that delivers 6x to 17x faster performance than the latest MKL BLAS.

CULA Tools
GPU-accelerated linear algebra library by EM Photonics, that utilizes CUDA to dramatically improve the computation speed of sophisticated mathematics.

MAGMA
A collection of next gen linear algebra routines. Designed for heterogeneous GPU-based architectures. Supports current LAPACK and BLAS standards.

IMSL Fortran Numerical Library
Developed by RogueWave, a comprehensive set of mathematical and statistical functions that offloads work to GPUs.

NVIDIA cuSPARSE
NVIDIA CUDA Sparse (cuSPARSE) Matric library provides a collection of basic linear algebra subroutines used for sparse matrices that delivers over 8x performance boost.

NVIDIA CUSP
A GPU accelerated Open Source C++ library of generic parallel algorithms for sparse linear algebra and graph computations. Provides an easy to use high-level interface.

AccelerEyes ArrayFire
Comprehensive GPU function library, including functions for math, signal and image processing, statistics, and more. Interfaces for C, C++, Fortran, and Python.

NVIDIA cuRAND
The CUDA Random Number Generation library performs high quality GPU-accelerated random number generation (RNG) over 8x faster than typical CPU only code.

NVIDIA NPP
NVIDIA Performance Primitives is a GPU accelerated library with a very large collection of 1000's of image

NVIDIA CUDA Math Library
An industry proven, highly accurate collection of standard mathematical functions, providing high

Thrust
A powerful, open source library of parallel algorithms and data structures. Perform GPU-accelerated sort, scan, transform, and reductions

QUICKLINKS

The NVIDIA Registered Developer Program

Registered Developers Website

NVDeveloper (old site)

CUDA Newsletter

CUDA Downloads

CUDA GPUs

Get Started - Parallel Computing

CUDA Spotlights

CUDA Tools & Ecosystem

FEATURED ARTICLES

INTRODUCING NVIDIA NSIGHT VISUAL STUDIO EDITION 2.2, WITH LOCAL SINGLE GPU CUDA DEBUGGING!

Previous Next

LATEST NEWS

OpenACC Compiler For $199

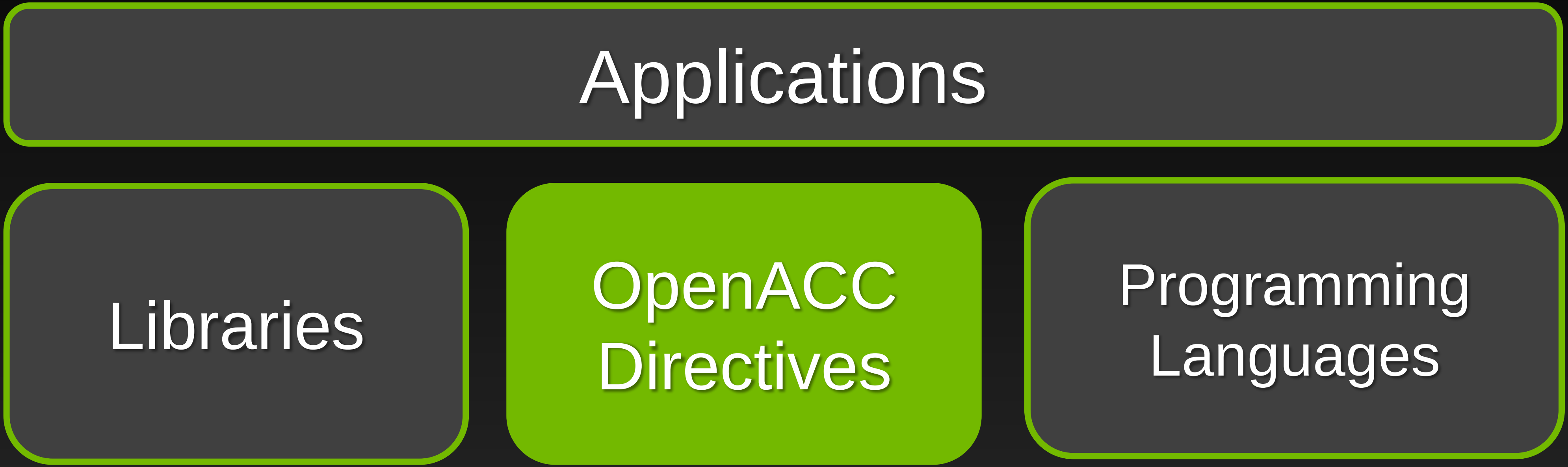
Introducing NVIDIA Nsight Visual Studio Edition 2.2, With Local Single GPU CUDA Debugging!

CUDA Spotlight: Lorena Barba, Boston University

Stanford To Host CUDA On Campus Day, April 13, 2012

CUDA Spotlight:

3 Ways to Accelerate Applications



Applications

Libraries

OpenACC Directives

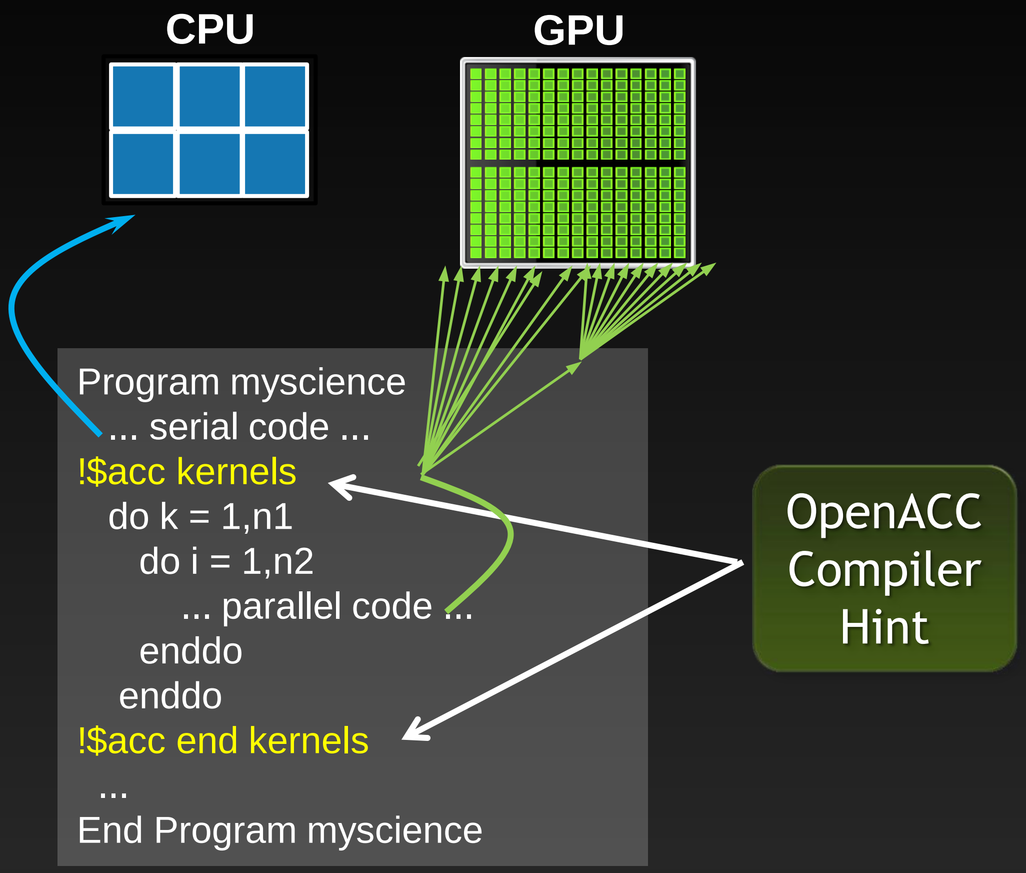
Programming Languages

"Drop-in"
Acceleration

Easily Accelerate
Applications

Maximum
Performance

OpenACC Directives



```
Program myscience
 ... serial code ...
!$acc kernels
 do k = 1,n1
   do i = 1,n2
       ... parallel code ...
   enddo
  enddo
!$acc end kernels
 ...
End Program myscience
```

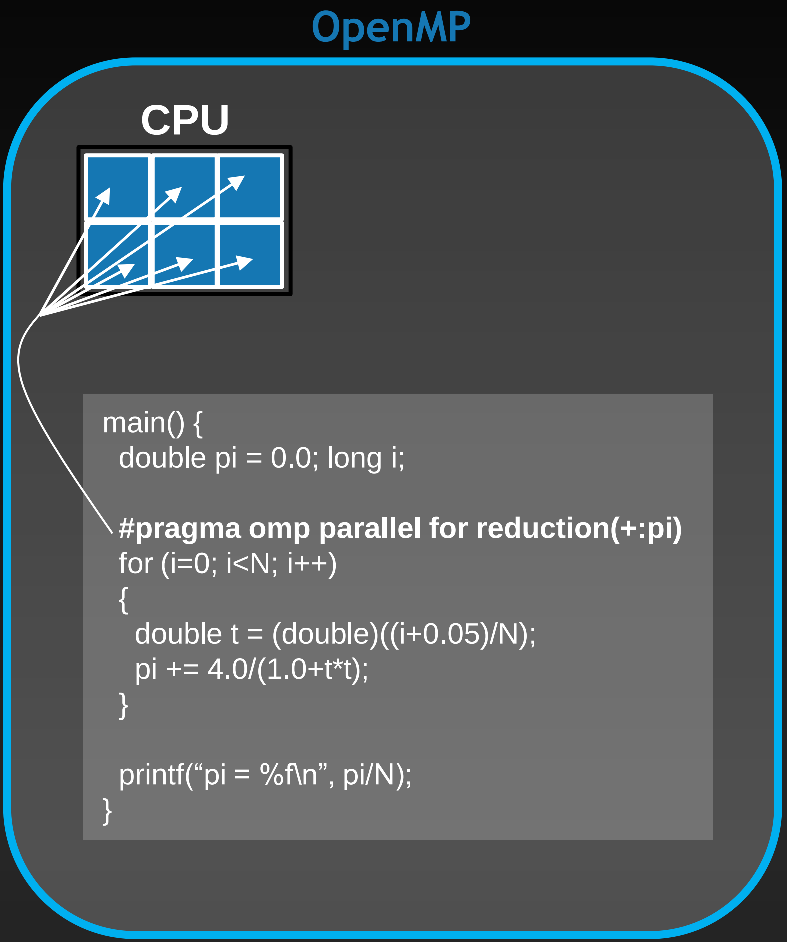
Your original Fortran or C code

Simple Compiler hints

Compiler Parallelizes code

Works on many-core GPUs & multicore CPUs

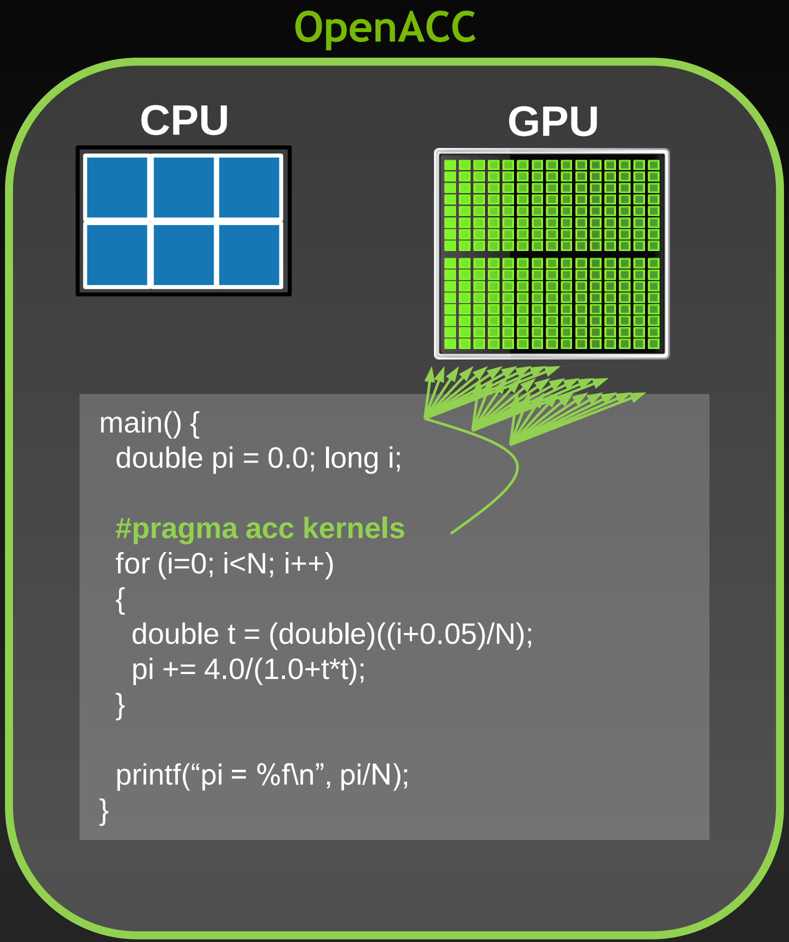
Familiar to OpenMP Programmers



```
main() {
  double pi = 0.0; long i;

  #pragma omp parallel for reduction(+:pi)
  for (i=0; i<N; i++)
  {
    double t = (double)((i+0.05)/N);
    pi += 4.0/(1.0+t*t);
  }

  printf(“pi = %f\n”, pi/N);
}
```



```
main() {
  double pi = 0.0; long i;

  #pragma acc kernels
  for (i=0; i<N; i++)
  {
    double t = (double)((i+0.05)/N);
    pi += 4.0/(1.0+t*t);
  }

  printf(“pi = %f\n”, pi/N);
}
```

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

The Standard for GPU Directives

- **Easy:** Directives are the easy path to accelerate compute intensive applications
- **Open:** OpenACC is an open GPU directives standard, making GPU programming straightforward and portable across parallel and multi-core processors
- **Powerful:** GPU Directives allow complete access to the massive parallel power of a GPU

OpenACC®

DIRECTIVES FOR ACCELERATORS

Directives: Easy & Powerful

Real-Time Object Detection

Global Manufacturer of Navigation Systems

5x in 40 Hours

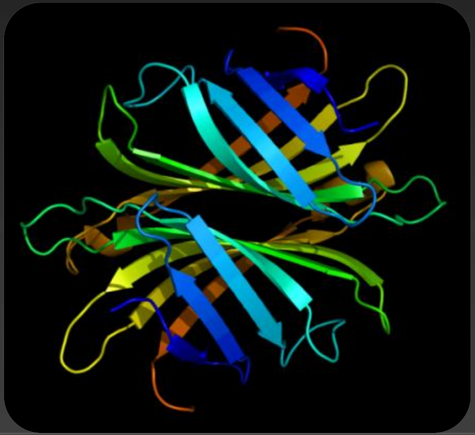
Valuation of Stock Portfolios using Monte Carlo

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

A Very Simple Exercise: SAXPY

SAXPY in C

```
void saxpy(int n,
           float a,
           float *x,
           float *restrict y)
{
#pragma acc kernels
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}


...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

SAXPY in Fortran

```
subroutine saxpy(n, a, x, y)
  real :: x(:), y(:), a
  integer :: n, i
$!acc kernels
  do i=1,n
    y(i) = a*x(i)+y(i)
  enddo
$!acc end kernels
end subroutine saxpy


...
$ Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...
```

OpenACC Specification and Website

- Full OpenACC 1.0 Specification available online

http://www.openacc.org

- Implementations available now from PGI and CAPS

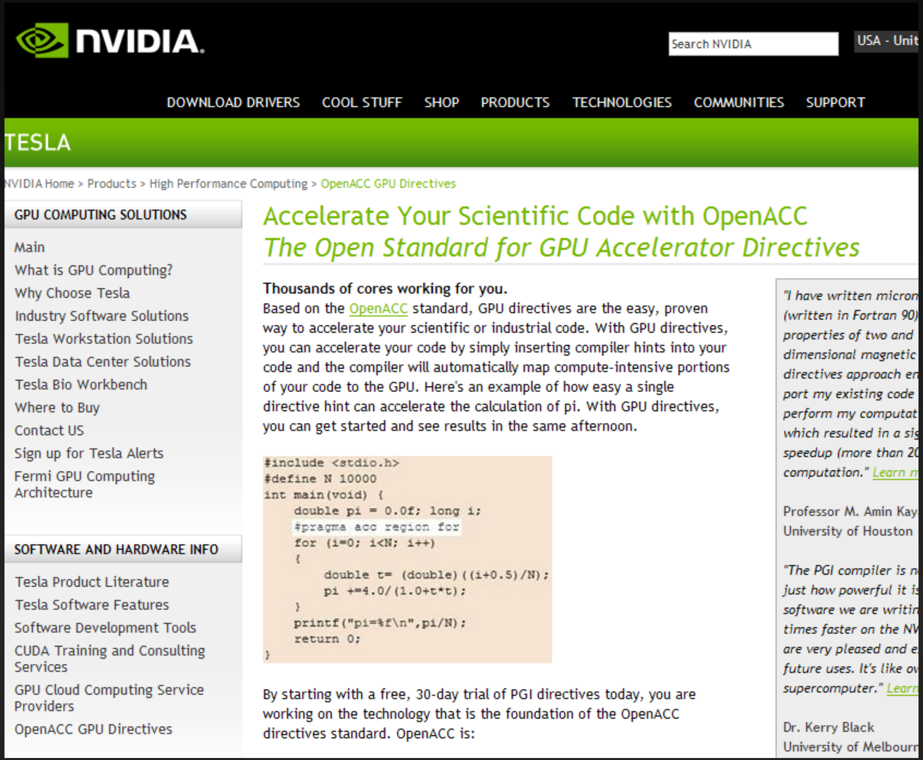
Start Now with OpenACC Directives

Sign up for a free trial of the directives compiler now!

Free trial license to PGI Accelerator

Tools for quick ramp

www.nvidia.com/gpudirectives



NVIDIA Home > Products > High Performance Computing > OpenACC GPU Directives

GPU COMPUTING SOLUTIONS

- Main
- What is GPU Computing?
- Why Choose Tesla
- Industry Software Solutions
- Tesla Workstation Solutions
- Tesla Data Center Solutions
- Tesla Bio Workbench
- Where to Buy
- Contact US
- Sign up for Tesla Alerts
- Fermi GPU Computing Architecture

SOFTWARE AND HARDWARE INFO

- Tesla Product Literature
- Tesla Software Features
- Software Development Tools
- CUDA Training and Consulting Services
- GPU Cloud Computing Service Providers
- OpenACC GPU Directives

Accelerate Your Scientific Code with OpenACC

The Open Standard for GPU Accelerator Directives

Thousands of cores working for you.

Based on the OpenACC standard, GPU directives are the easy, proven way to accelerate your scientific or industrial code. With GPU directives, you can accelerate your code by simply inserting compiler hints into your code and the compiler will automatically map compute-intensive portions of your code to the GPU. Here's an example of how easy a single directive hint can accelerate the calculation of pi. With GPU directives, you can get started and see results in the same afternoon.

```
#include <stdio.h>
#define N 10000
int main(void) {
    double pi = 0.0f; long i;
    #pragma acc region for
    for (i=0; i<N; i++)
    {
        double t= (double)((i+0.5)/N);
        pi +=4.0/(1.0+t*t);
    }
    printf("pi=%f\n",pi/N);
    return 0;
}
```

By starting with a free, 30-day trial of PGI directives today, you are working on the technology that is the foundation of the OpenACC directives standard. OpenACC is:

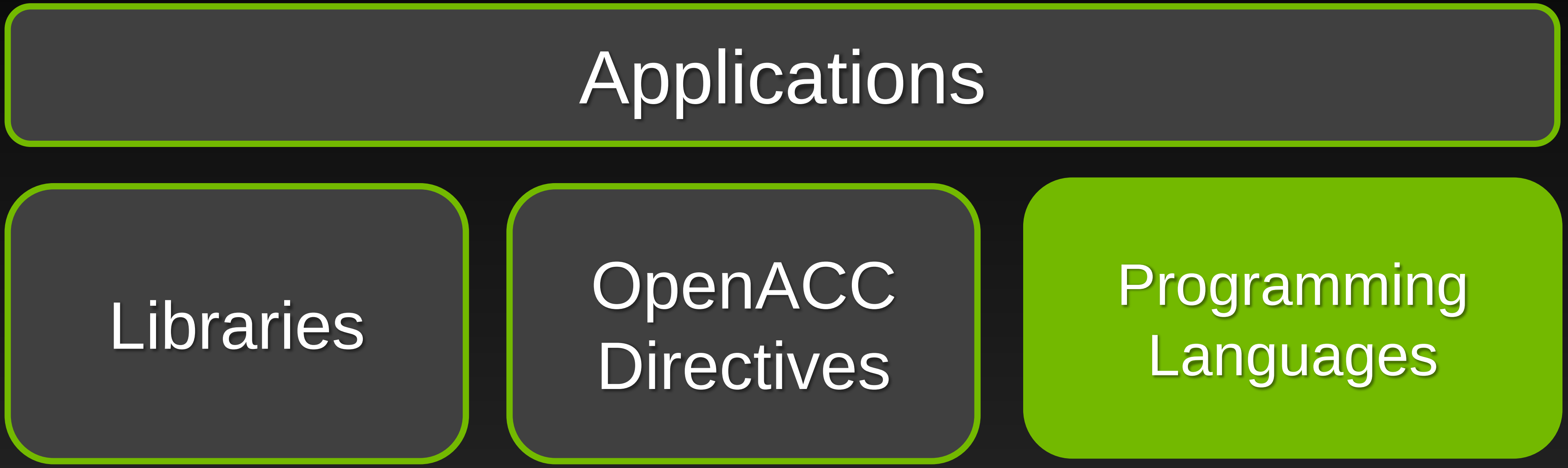
"I have written micron
(written in Fortran 90)
properties of two and
dimensional magnetic
directives approach en
port my existing code
perform my computat
which resulted in a sig
speedup (more than 20
computation." Learn m

Professor M. Amin Kay
University of Houston

"The PGI compiler is n
just how powerful it is
software we are writin
times faster on the NV
are very pleased and e
future uses. It's like ov
supercomputer." Learn

Dr. Kerry Black
University of Melbourn

3 Ways to Accelerate Applications



Applications

Libraries

OpenACC Directives

Programming Languages

"Drop-in" Acceleration

Easily Accelerate Applications

Maximum Performance

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);
```

http://developer.nvidia.com/cuda-toolkit

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 (=)

```fortran
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
```

http://developer.nvidia.com/cuda-fortran

CUDA C++: Develop Generic Parallel Code

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

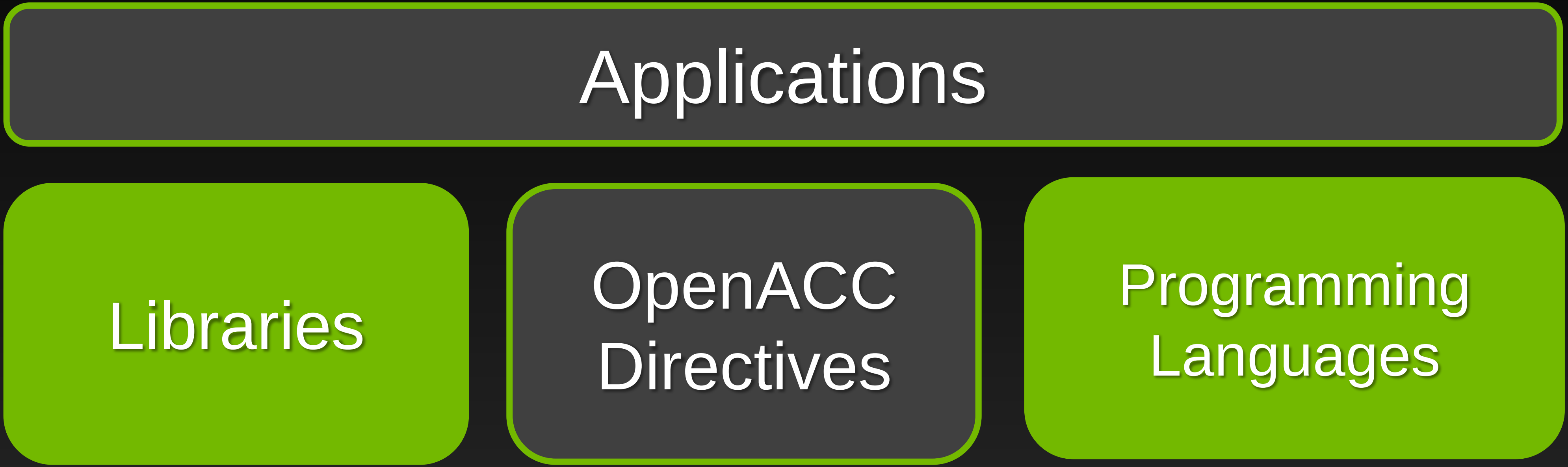
- Class hierarchies
- __device__ methods
- Templates
- Operator overloading
- Functors (function objects)
- Device-side new/delete
- More…

```
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
}
```

http://developer.nvidia.com/cuda-toolkit

~~4~~3+ Ways to Accelerate Applications



Applications

Libraries

OpenACC Directives

Programming Languages

“Drop-in” Acceleration

Easily Accelerate Applications

Maximum Performance

Rapid Parallel C++ Development

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

Thrust

```
// 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());
```

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

Six Ways to SAXPY

Programming Languages for GPU Computing

Single precision Alpha X Plus Y (SAXPY)

Part of Basic Linear Algebra Subroutines (BLAS) Library

$$z = \alpha x + y$$

x, y, z : vector

α : scalar

GPU SAXPY in multiple languages and libraries

A menagerie* of possibilities, not a tutorial

*technically, a *program chrestomathy*: http://en.wikipedia.org/wiki/Chrestomathy

1

OpenACC Compiler Directives

Parallel C Code

```c
void saxpy(int n,
           float a,
           float *x,
           float *y)
{
#pragma acc kernels
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

Parallel Fortran Code

```fortran
subroutine saxpy(n, a, x, y)
  real :: x(:), y(:), a
  integer :: n, i
!$acc kernels
  do i=1,n
    y(i) = a*x(i)+y(i)
  enddo
!$acc end kernels
end subroutine saxpy


...
! Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...
```

http://developer.nvidia.com/openacc or http://openacc.org

2

CUBLAS Library

Serial BLAS Code

```
int N = 1<<20;


...

// Use your choice of blas library

// Perform SAXPY on 1M elements
blas_saxpy(N, 2.0, x, 1, y, 1);
```

Parallel cuBLAS Code

```
int N = 1<<20;


cublasInit();
cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);

// Perform SAXPY on 1M elements
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);

cublasShutdown();
```

You can also call cuBLAS from Fortran, C++, Python, and other languages

http://developer.nvidia.com/cublas

3

CUDA C

Standard C

```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

int N = 1<<20;



// Perform SAXPY on 1M elements
saxpy(N, 2.0, x, y);
```

Parallel C

```
__global__
void saxpy(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];
}

int N = 1<<20;
cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, d_x, d_y);

cudaMemcpy(y, d_y, N, cudaMemcpyDeviceToHost);
```

http://developer.nvidia.com/cuda-toolkit

4

Thrust C++ Template Library

Serial C++ Code
with STL and Boost

```
int N = 1<<20;
std::vector<float> x(N), y(N);

...




// Perform SAXPY on 1M elements
std::transform(x.begin(), x.end(),
               y.begin(), y.end(),
              2.0f * _1 + _2);
```

www.boost.org/libs/lambda

Parallel C++ Code

```
int N = 1<<20;
thrust::host_vector<float> x(N), y(N);

...

thrust::device_vector<float> d_x = x;
thrust::device_vector<float> d_y = y;

// Perform SAXPY on 1M elements
thrust::transform(d_x.begin(), d_x.end(),
                  d_y.begin(), d_y.begin(),
                  2.0f * _1 + _2);
```

http://thrust.github.com

5

CUDA Fortran

Standard Fortran

```
module mymodule contains
  subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    do i=1,n
      y(i) = a*x(i)+y(i)
    enddo
  end subroutine saxpy
end module mymodule

program main
  use mymodule
  real :: x(2**20), y(2**20)
  x = 1.0, y = 2.0

  ! Perform SAXPY on 1M elements
  call saxpy(2**20, 2.0, x, y)

end program main
```

Parallel Fortran

```
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

  ! Perform SAXPY on 1M elements
  call saxpy<<<4096,256>>>(2**20, 2.0, x_d, y_d)

end program main
```

http://developer.nvidia.com/cuda-fortran

6

Python

Standard Python

```
import numpy as np


def saxpy(a, x, y):
  return [a * xi + yi
          for xi, yi in zip(x, y)]

x = np.arange(2**20, dtype=np.float32)
y = np.arange(2**20, dtype=np.float32)





cpu_result = saxpy(2.0, x, y)
```

http://numpy.scipy.org

Copperhead: Parallel Python

```
from copperhead import *
import numpy as np

@cu
def saxpy(a, x, y):
  return [a * xi + yi
          for xi, yi in zip(x, y)]

x = np.arange(2**20, dtype=np.float32)
y = np.arange(2**20, dtype=np.float32)

with places.gpu0:
  gpu_result = saxpy(2.0, x, y)

with places.openmp:
  cpu_result = saxpy(2.0, x, y)
```

http://copperhead.github.com

Enabling Endless Ways to SAXPY

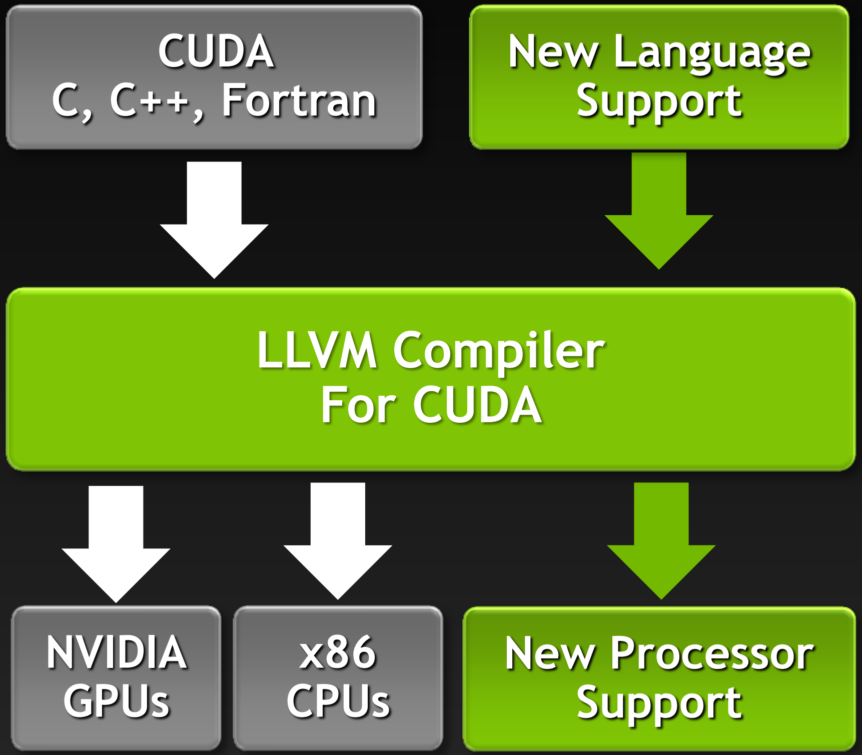
Developers want to build front-ends for

Java, Python, R, DSLs

Target other processors like

ARM, FPGA, GPUs, x86

CUDA Compiler Contributed to Open Source LLVM



CUDA
C, C++, Fortran

New Language Support

LLVM Compiler For CUDA

NVIDIA GPUs

x86 CPUs

New Processor Support

LLVM COMPILER INFRASTRUCTURE

Recommended Approaches

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

How to get started

www.nvidia.com/cudazone

www.nvidia.com/getcuda

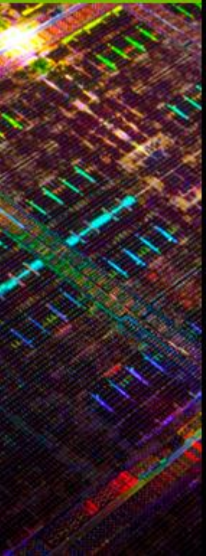
GPUs with fricken laserbeams!

- Created by Intellectual Ventures to help fight malaria in third world countries
- Image detection and targeting is done with NVIDIA GPUs

GPU TECHNOLOGY CONFERENCE

GTC 2013 | March 18-21 | San Jose, CA

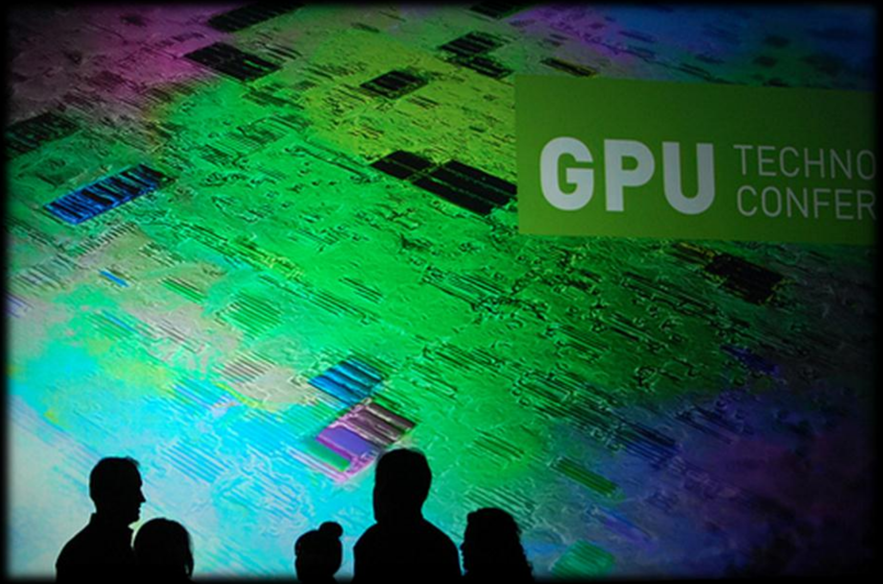
The Smartest People. The Best Ideas. The Biggest Opportunities.



Opportunities for Participation:

SPEAK - Showcase your work among the elite of graphics computing

- Call for Sessions: August 2012
- Call for Posters: October 2012



REGISTER - learn from the experts and network with your peers

- Use promo code **GM10SIGG** for a 10% discount

SPONSOR - Reach influential IT decision-makers

Learn more at www.gputechconf.com