CUDA Libraries and Ecosystem Overview

Cliff Woolley, NVIDIA Corporation

GTC Asia 2011

3 Ways to Accelerate on GPU

Application

Libraries

Directives

Programming Languages

Easiest Approach for 2x to 10x Acceleration

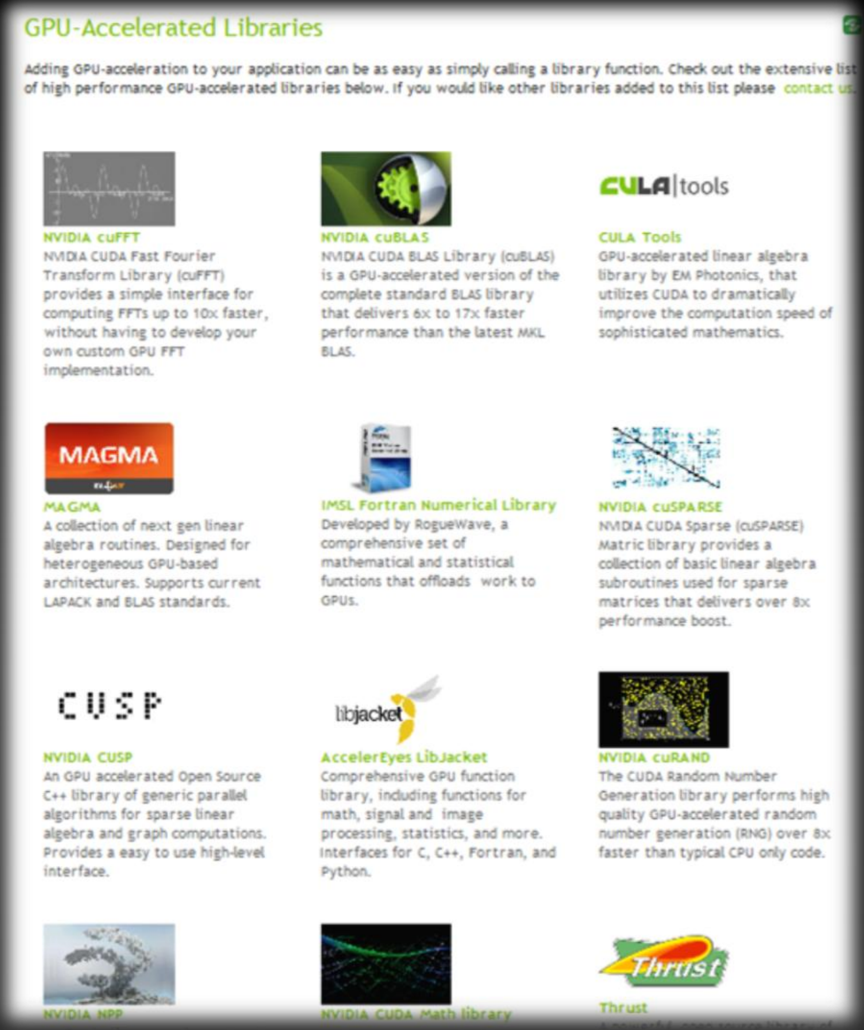
Maximum Performance

Effort Level

CUDA Libraries

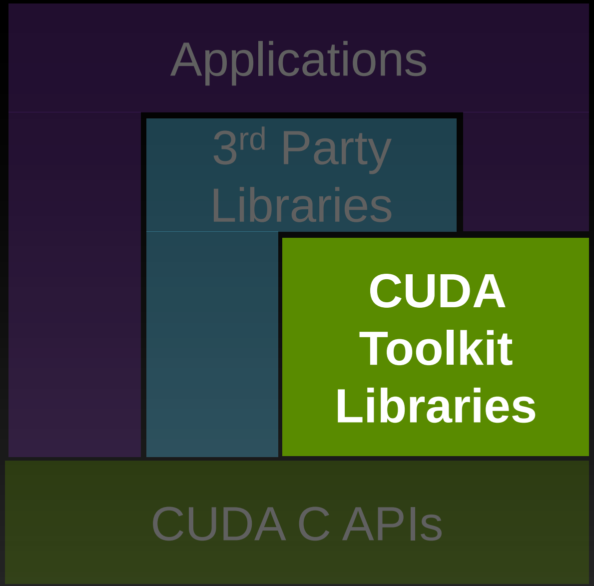
CUDA library ecosystem spans many fields

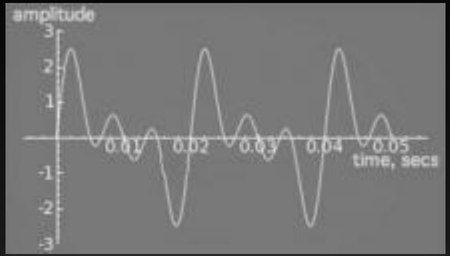
- Math, Numerics, Statistics
- Dense & Sparse Linear Algebra
- Algorithms (sort, etc.)
- Image Processing
- Computer Vision
- Signal Processing
- Finance

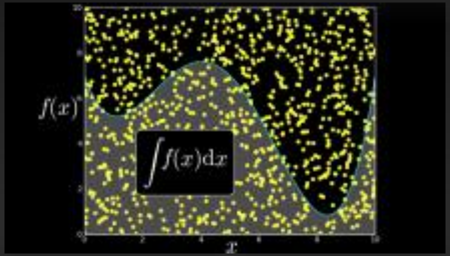


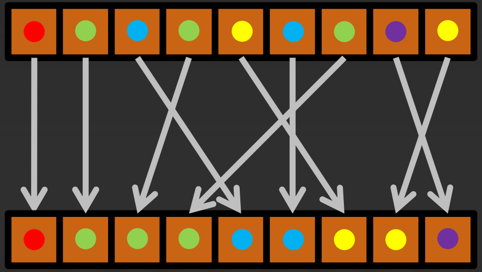
http://developer.nvidia.com/gpu-accelerated-libraries

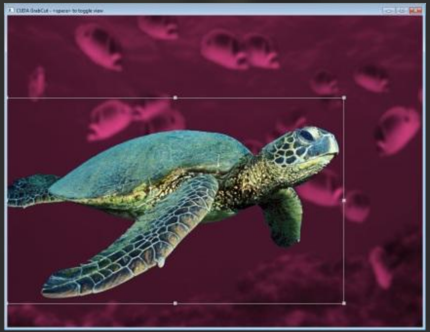
NVIDIA CUDA Toolkit Libraries







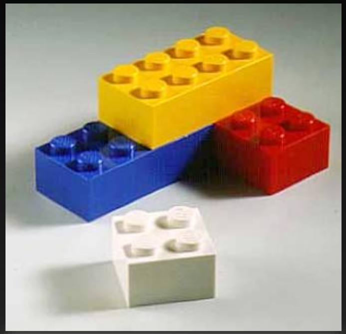




cuBLAS	*dense linear algebra*
cuSPARSE	*sparse linear algebra*
cuFFT	*discrete Fourier transforms*
cuRAND	*random number generation*
NPP	*signal and image processing*
Thrust	*scan, sort, reduce, transform*
math.h	*floating point*
system calls	*printf, malloc, assert*

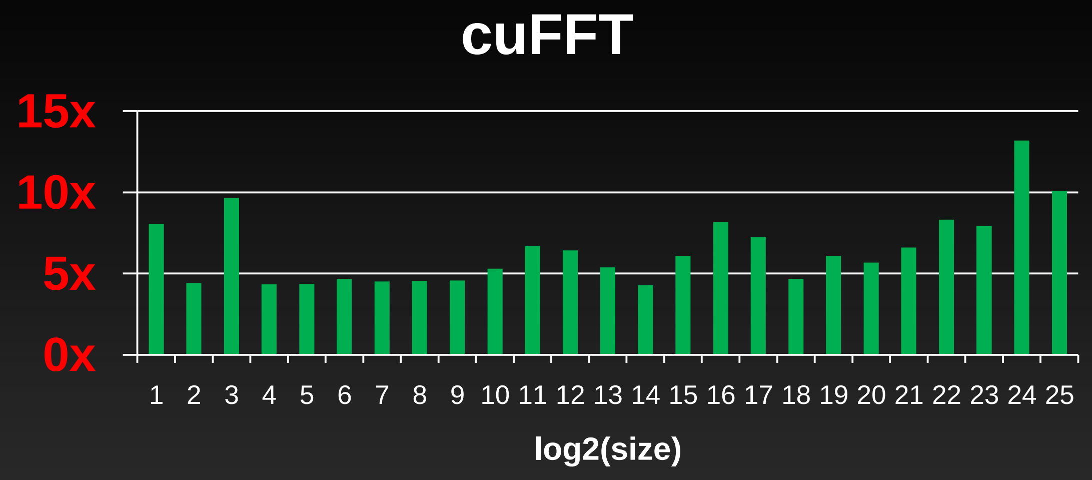
NVIDIA CUDA Library Approach

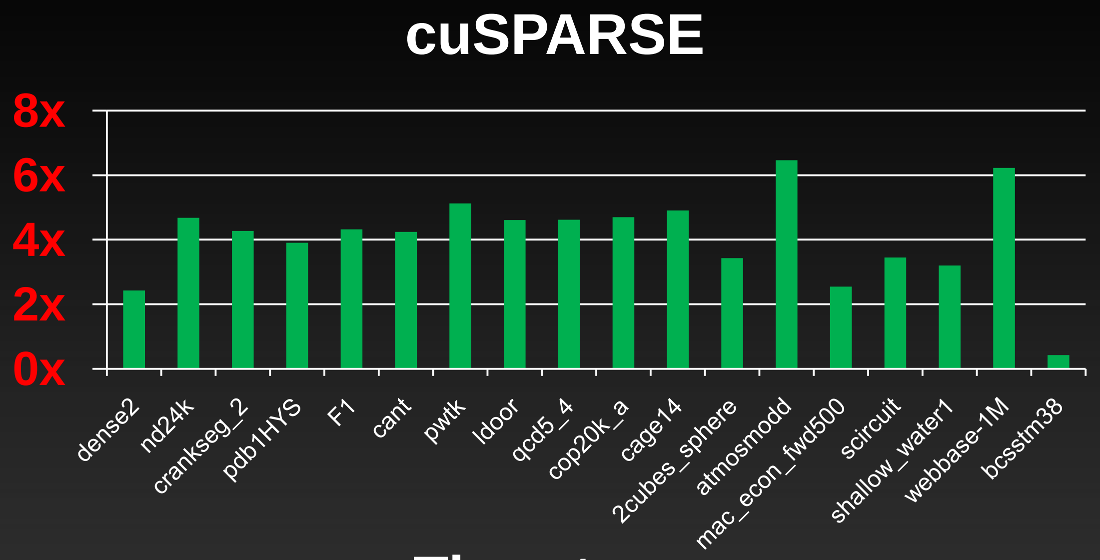
- Provide basic building blocks
- Make them easy to use
- Make them fast

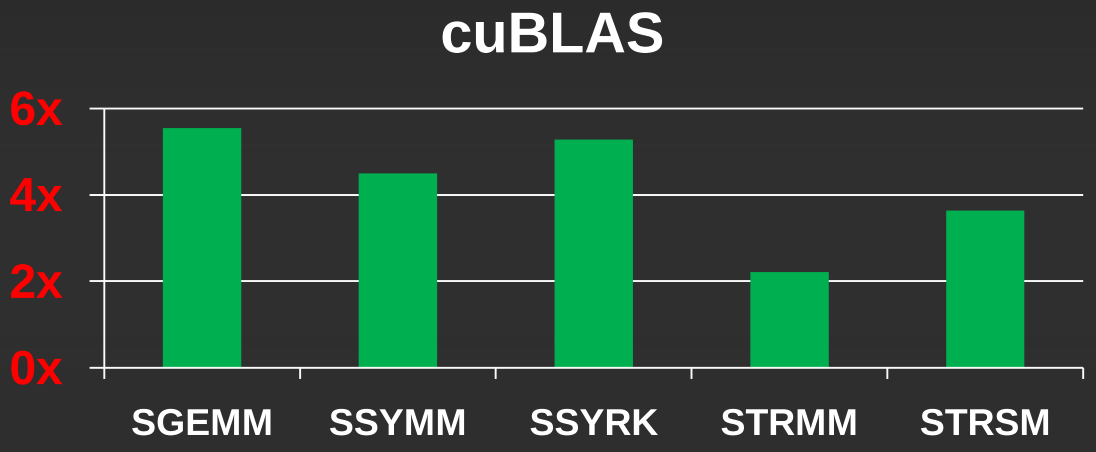


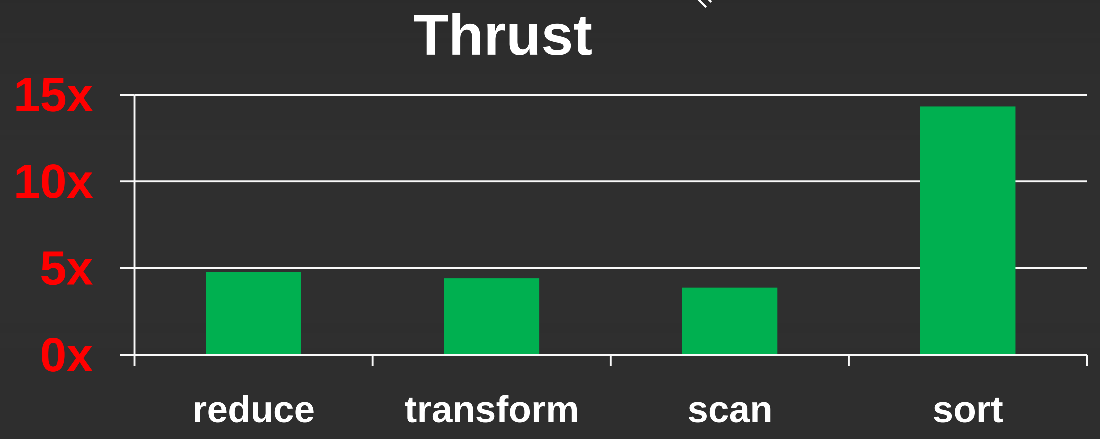
- Provides a quick path to GPU acceleration
- Enables ISVs to focus on their "secret sauce"
- Ideal for applications that use CPU libraries

4x-10x speedups over Intel on single precision





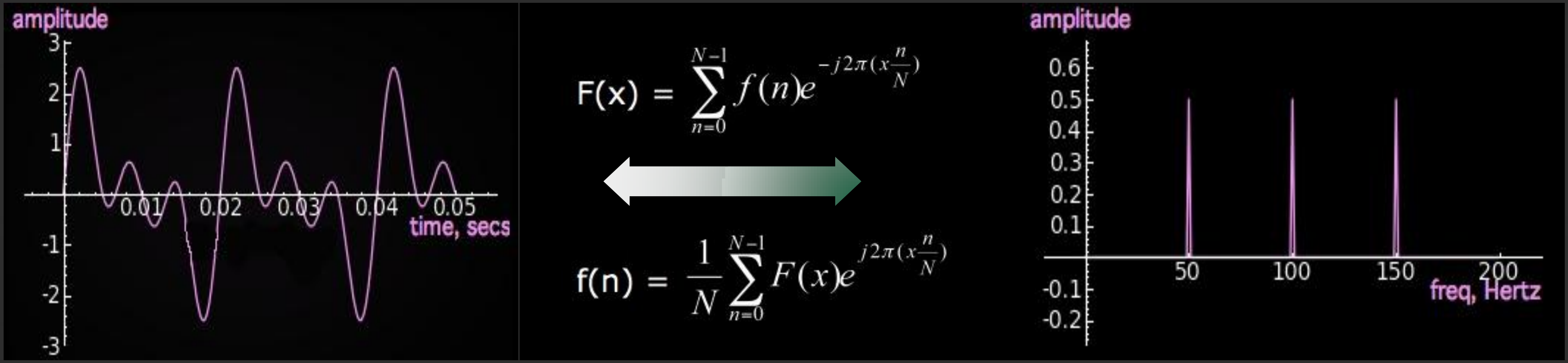




- CUDA 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

cuFFT: Multi-dimensional FFTs

- New features in CUDA 4.1:
  - Flexible input & output data layouts for all transform types
    - Similar to the FFTW “Advanced Interface”
    - Eliminates extra data transposes and copies
  - API is now thread-safe & callable from multiple host threads
  - Restructured documentation to clarify data layouts



cuBLAS: Dense linear algebra on GPUs

- Complete BLAS implementation plus useful extensions
  - Supports all 152 standard routines for single, double, complex, and double complex

- New features in CUDA 4.1:
  - New batched GEMM API provides >4x speedup over MKL
    - Useful for batches of 100+ small matrices from 4x4 to 128x128
  - 5%-10% performance improvement to large GEMMs

cuSPARSE: Sparse linear algebra routines

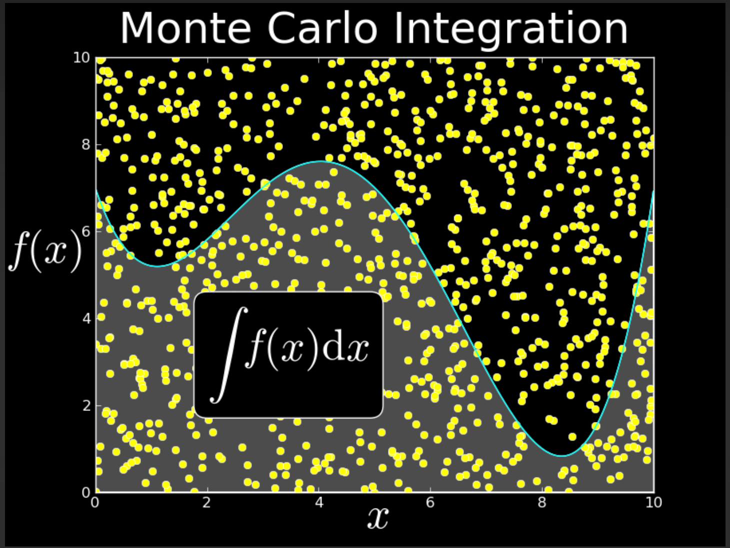
- Sparse matrix-vector multiplication & triangular solve
  - APIs optimized for iterative methods
- New features in 4.1:
  - Tri-diagonal solver with speedups up to 10x over Intel MKL
  - ELL-HYB format offers 2x faster matrix-vector multiplication

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

cuRAND: Random Number Generation

- Pseudo- and Quasi-RNGs
- Supports several output distributions
- Statistical test results reported in documentation

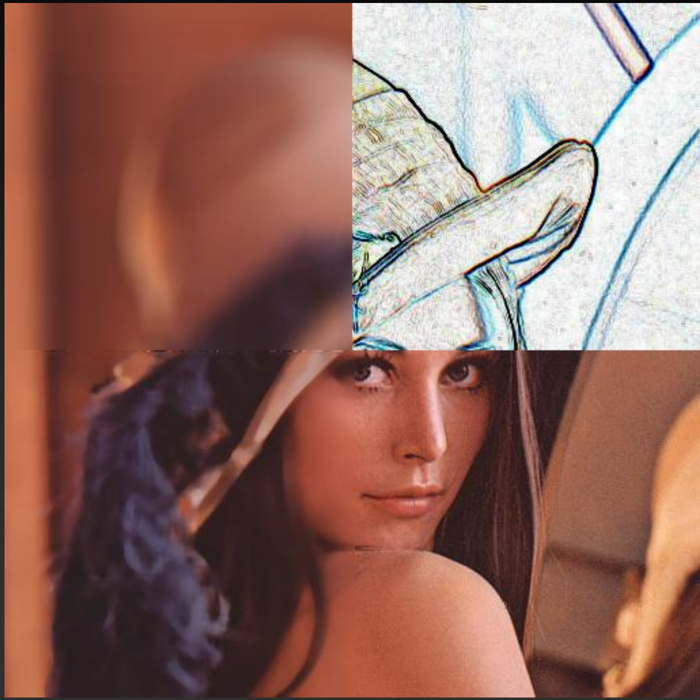
- New RNGs in CUDA 4.1:
  - MRG32k3a RNG
  - MTGP11213 Mersenne Twister RNG



NPP: NVIDIA Performance Primitives

Up to **40x** speedups

- Arithmetic, Logic, Conversions, Filters, Statistics, etc.
- Majority of primitives 5x to 10x faster than analogous routines in Intel IPP
- 1,000+ new image primitives in 4.1



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

Thrust: CUDA C++ Template Library

- Template library for CUDA mimics the C++ STL
  - Optimized algorithms for sort, reduce, scan, etc.
  - OpenMP backend for portability
- Allows applications and prototypes to be built *quickly*
- New in 4.1: Boost-style placeholders allow inline functors
  - Example: *saxpy* in 1 line:

```
thrust::transform(x.begin(), x.end(), y.begin(), y.begin(), a * _1 + _2);
```

math.h: C99 floating-point library + extras

- Basic: +, *, /, 1/, sqrt, FMA (all IEEE-754 accurate for float, double, all rounding modes)
- Exponentials: exp, exp2, log, log2, log10, ...
- Trigonometry: sin, cos, tan, asin, acos, atan2, sinh, cosh, asinh, acosh, ...
- Special functions: lgamma, tgamma, erf, erfc
- Utility: fmod, remquo, modf, trunc, round, ceil, floor, fabs, ...
- Extras: rsqrt, rcbrt, exp10, sinpi, sincos, cospi, erfinv, erfcinv, ...

- **New features in 4.1:**
  - **Bessel functions: j0, j1, jn, y0, y1, yn**
  - **Scaled complementary error function: erfcx**
  - **Average and rounded average: __{u}hadd, __{u}rhadd**

Directives: OpenACC

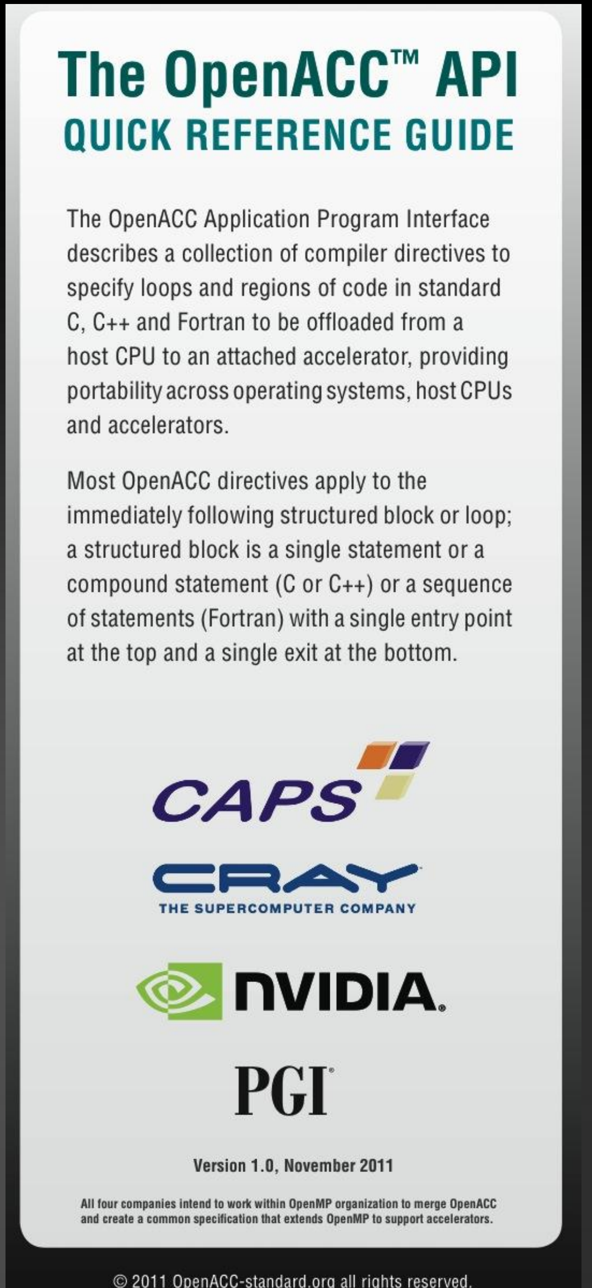
OpenACC Directives

Easy to Use

- Add compiler hints to identify loop nests for acceleration.

Portable

- Single code base works for accelerated systems and generic CPU multi-core systems.



The OpenACC™ API
QUICK REFERENCE GUIDE

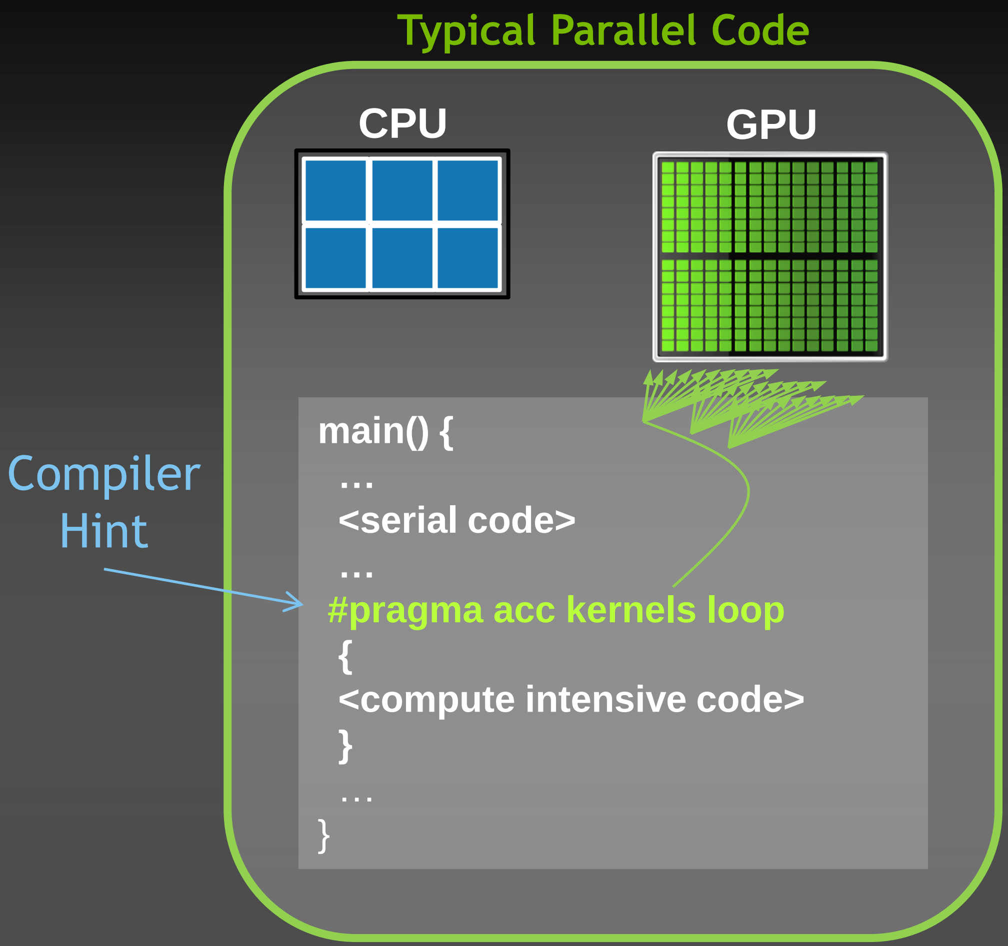
The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C or C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.

Version 1.0, November 2011

All four companies intend to work within OpenMP organization to merge OpenACC and create a common specification that extends OpenMP to support accelerators.

Directives: Simple Hints for the Compiler



Quickest Path to

Massive parallelism

www.openacc-standard.org

Directives: An Example

```
!$acc data region copy(A,Anew)
 iter=0
 do while ( err > tol .and. iter < iter_max )

  iter = iter +1
  err=0._fp_kind
!$acc region
   do j=1,m
    do i=1,n
     Anew(i,j) = .25_fp_kind *( A(i+1,j  ) + A(i-1,j  ) &
                               +A(i  ,j-1) + A(i  ,j+1))
     err = max( err, Anew(i,j)-A(i,j))
    end do
   end do
!$acc end region
   IF(mod(iter,100)==0 .or. iter == 1)    print *, iter, err
   A= Anew
 end do
!$acc end data region
```

Copy arrays into GPU memory within data region

Parallelize code inside region

Close off parallel region

Close off data region, copy data back

Development, Debugging, and Deployment Tools

Programming Languages & APIs

Microsoft® DirectX® 11

HMPP Compiler

Python for CUDA

CUDA Fortran

CUDA-x86

NVIDIA CUDA

NVIDIA C Compiler

Microsoft AMP C/C++

PGI Accelerator

OpenCL

OpenGL™

NVIDIA Parallel Nsight
for Visual Studio

NVIDIA CUDA-MEMCHECK
for Linux & Mac

Allinea DDT with CUDA
Distributed Debugging Tool

NVIDIA CUDA-GDB
for Linux & Mac

TotalView for CUDA
for Linux Clusters

Debugging CUDA

Command Line to Cluster Solutions

Performance Analysis Tools

Single GPU to Hybrid Cluster Solutions

NVIDIA Parallel Nsight
for Visual Studio

Vampir Trace Collector

NVIDIA Visual Profiler
for Linux & Mac

Tuning and Analysis Utilities

TAU Performance System

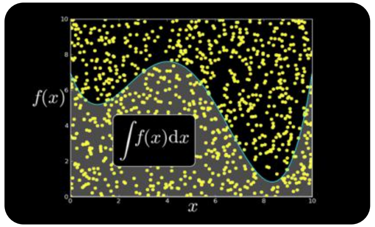
PAPI

Performance API Library

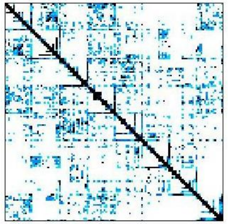
HPCToolkit

Under Development

NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE

NVIDIA NPP

GPU VSIPL

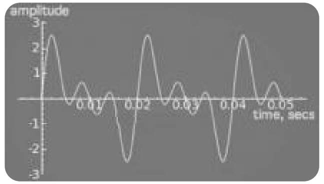
Vector Signal
Image Processing

CULA tools

GPU Accelerated
Linear Algebra

MAGMA

Matrix Algebra on
GPU and Multicore



NVIDIA cuFFT

ROGUE WAVE
SOFTWARE

IMSL Library

CUSP

Sparse Linear
Algebra

libjacket

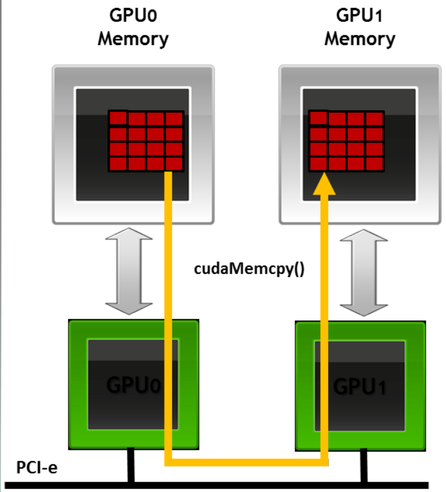
Matrix companion
to CUDA

Thrust

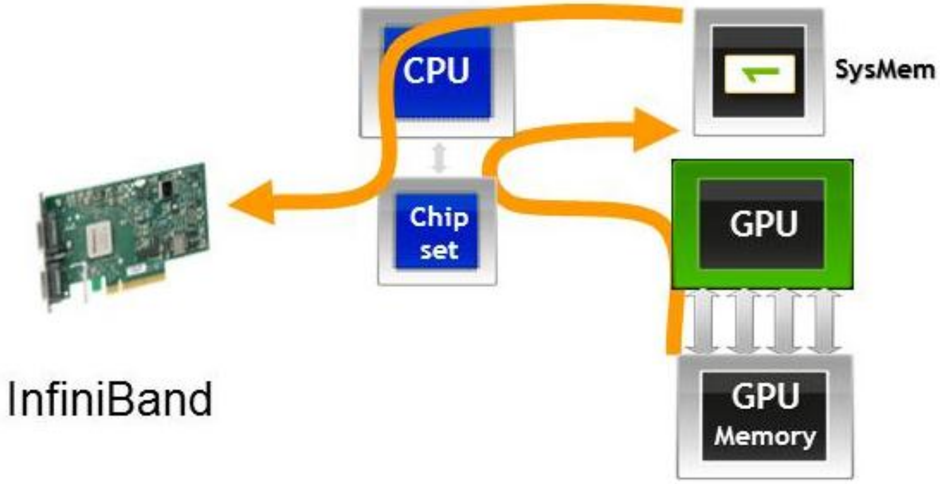
C++ STL Features
for CUDA

GPU Accelerated Libraries

Optimized Popular Numerical Solutions



Peer-Peer Transfers



GPUDirect™

OPENFABRICS ALLIANCE

As of OFED 1.5.2

MVAPICH

Announced
pre-release at SC2011

Platform Computing

Platform MPI
Announced beta at SC2011

MPI Libraries

Using Latest GPU Features

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

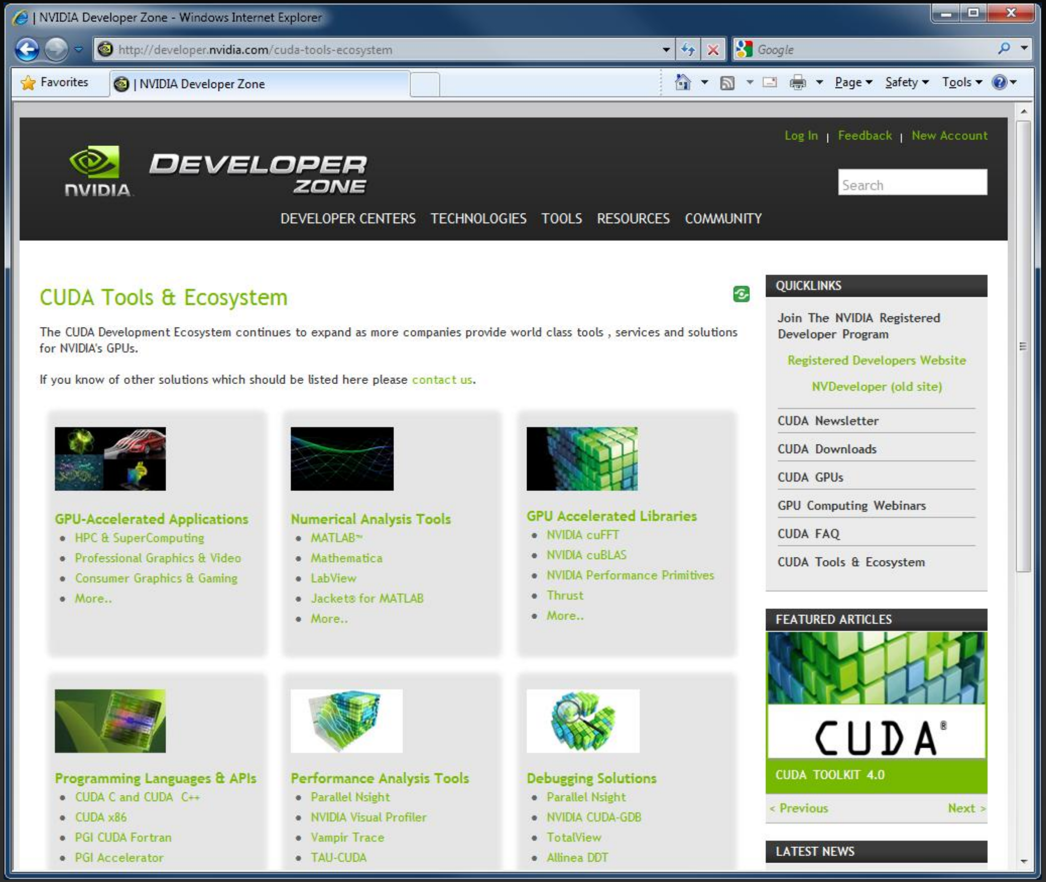
Univa Grid Engine

Cluster Management & Job Scheduling

Next Steps

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

  developer.nvidia.com/cuda-tools-ecosystem



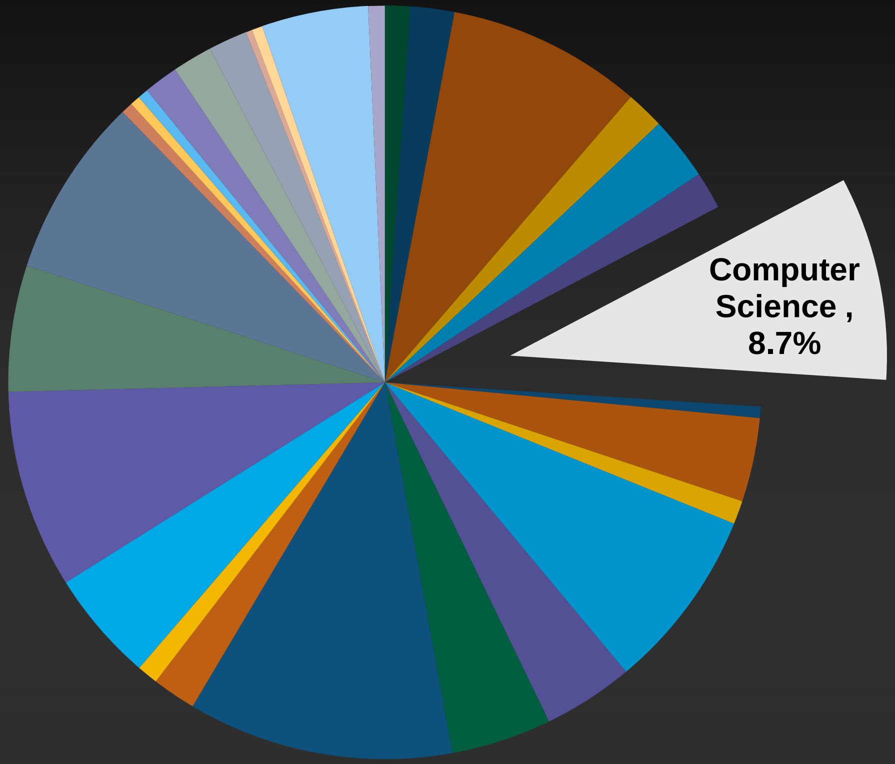
Thank You!

Backup

OpenACC

- Industry Standard for Directives-based Parallel Programming
- Supported by majority of accelerator technology providers
- Provides portability across hardware platforms and compiler vendors

Motivation: Provide Parallel Execution to Domain Scientists



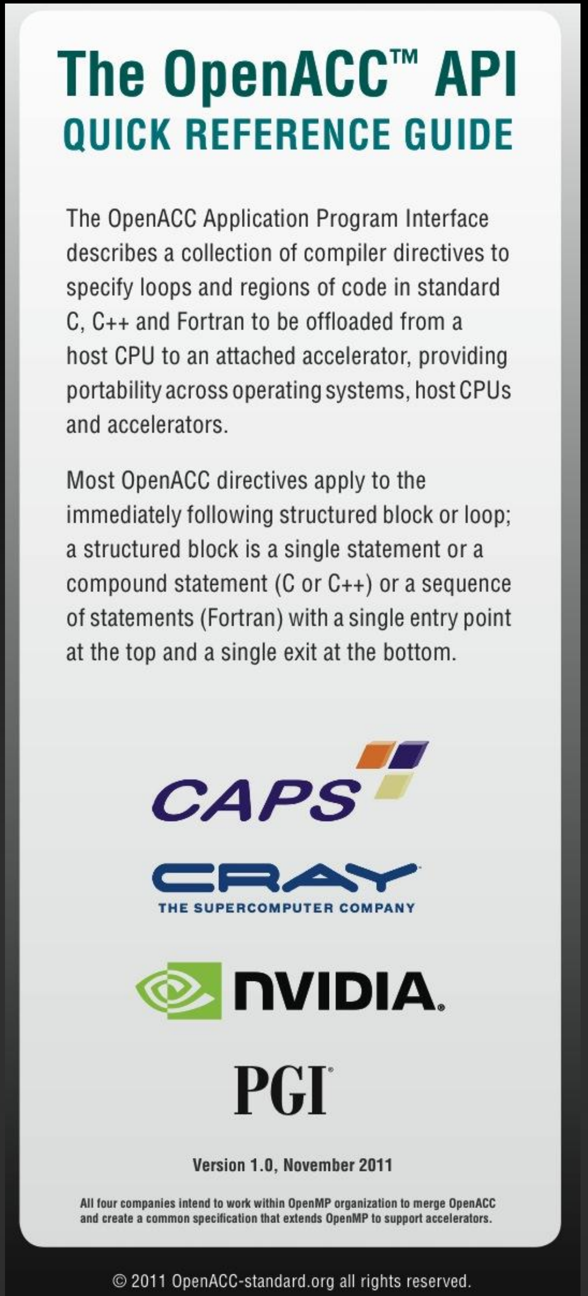
Engineering Physics
Physics
Civil Engineering
Bioengineering
General Engineering
Industrial Engineering
Financial Engineering
Aerospace Engineering
Engineering Mechanics
Mechanical Engineering
Materials Science & Engr
Computer Engineering
Electrical Engineering
Nuclear Engineering
Atmospheric Sciences

Mathematics
Biology
Chemistry
Molecular and Cellular Biology
Biophysics & Computational Bio
Neuroscience
Astronomy
Biochemistry
Physics
Statistics
Cell and Developmental Biology
Geology
Chemical Engineering
Cell and Microbiology

14,090 Students focused on Science and technology out of 42,606
12,089 of the 14,000 are not in Computer Science

OpenACC Specification

- Hardware agnostic and platform independent (CPU only, different GPUs)
- OpenACC is an open standard for directives based computing
- Announced at SC11
- Caps, Cray, and PGI to ship OpenACC Compilers beginning Q1 2012



The OpenACC™ API
QUICK REFERENCE GUIDE

The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C or C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.

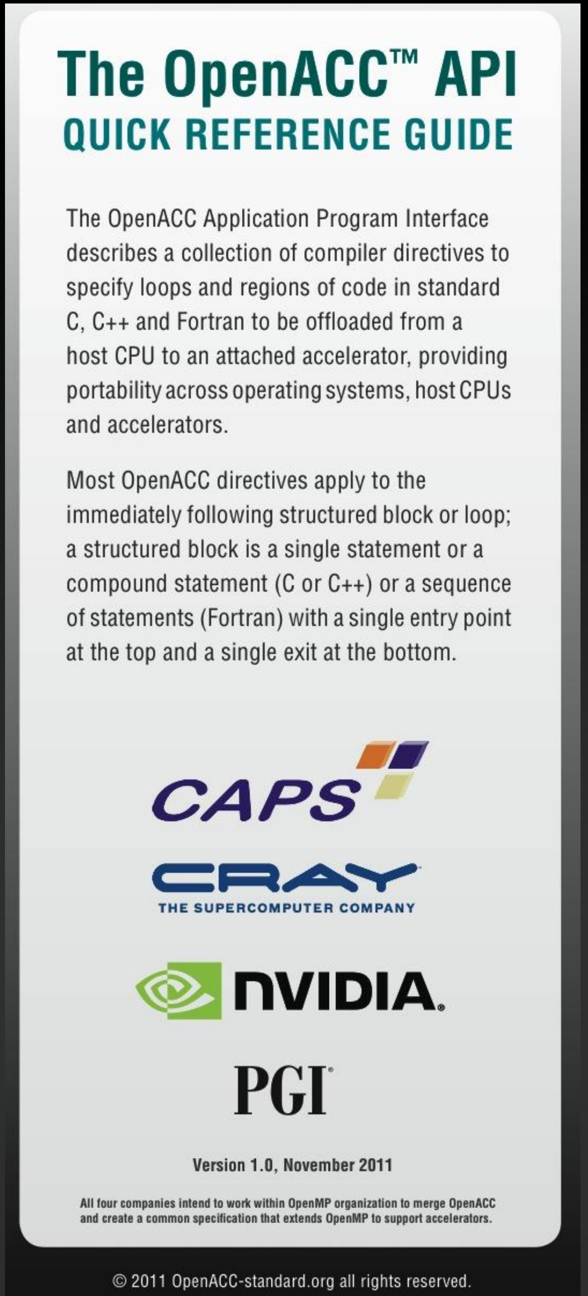
Version 1.0, November 2011

All four companies intend to work within OpenMP organization to merge OpenACC and create a common specification that extends OpenMP to support accelerators.

OpenACC Target Audience

OpenACC targets three classes of users:

1. Users with parallel codes, ideally with some OpenMP experience, but less GPU knowledge
2. Users with serial codes looking for portable parallel performance with and without GPUs
3. “Hardcore" GPU programmers with existing CUDA ports



The OpenACC™ API

QUICK REFERENCE GUIDE

The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C or C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.

Version 1.0, November 2011

All four companies intend to work within OpenMP organization to merge OpenACC and create a common specification that extends OpenMP to support accelerators.

Small Effort. Real Impact.

Large Oil Company

Dr. Jorge Pita

7 days and 3X

Solving billions of equations iteratively for oil exporation at world’s largest petroleum reservoirs

Univ. of Houston

Prof. Kayali

2 days and 20X

Analyzing magnetostatic interaction for innovations in areas such as storage, memories, and biosensing

Uni. Of Melbourne

Prof. Black

2 Days and 60x

Better understand complex reasons by lifecycles of snapper fish in Port Phillip Bay

Ufa State Aviation

Prof. Arthur Yuldashev

4 Weeks and 7X

Generating stochastic geological models of oilfield reservoirs with borehole data

GAMESS-UK

Prof. Karl Wilkinson

10x

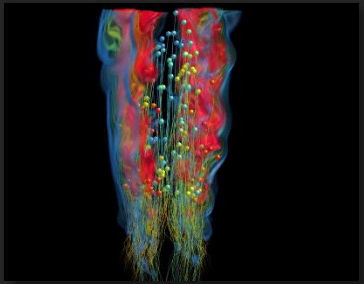
Used for various fields such as investigating biofuel production and molecular sensors.

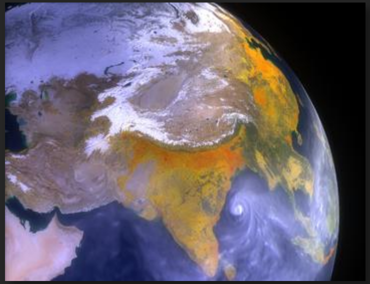
Focus on Exposing Parallelism

With Directives, tuning work focusses on *exposing parallelism*, which makes codes inherently better

Example: Application tuning work for new Titan supercomputer at ORNL

S3D
Research more efficient combustion with next-generation fuels





CAM-SE
Answer questions about specific climate change adaptation and mitigation scenarios

- Code tuning of key kernel using directives
- Doubled performance of all-CPU version
- Still 6.5x faster on CPU+GPU vs. CPU+CPU

- Code tuning of key kernels using directives
- Improved performance of CPU version by 50%
- Still 3 to 6x faster on CPU+GPU vs. CPU+CPU

Jeff Vetter Ph.D.

Gordon Bell Prize 2010

Georgia Institute of Technology, Professor

Oak Ridge National Labs, Group Leader, Science and Mathematics

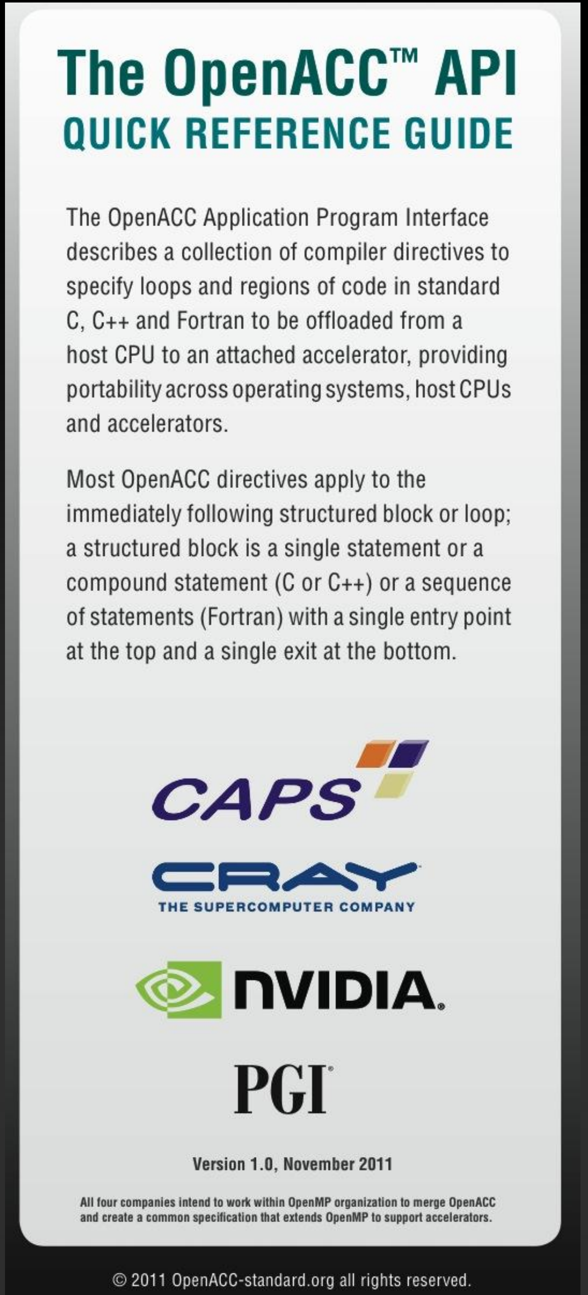
"Our NSF Keeneland project serves a diverse user community in terms of both application domains and level of GPU experience. We believe that a directive-based strategy to programming heterogeneous systems, such as OpenACC, will quickly broaden the collection of users that can productively use these systems, and, in part, will help preserve their investments in application software development."

OpenACC Benefits

- Code easier to maintain
- Helps with Legacy code bases
- Portable: Can run same code CPU/GPU
- Programmer familiar with OpenMP

- Some performance loss
- Cray goal: 90% of CUDA

Note: Some compilers support mixing directives and CUDA



The OpenACC™ API
QUICK REFERENCE GUIDE

The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C or C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.

Version 1.0, November 2011

All four companies intend to work within OpenMP organization to merge OpenACC and create a common specification that extends OpenMP to support accelerators.

Constant progress on library development

2007 2008 2009 2010 2011

CUDA Toolkit 1.x

- Single precision
- cuBLAS
- cuFFT
- math.h

CUDA Toolkit 2.x

- Double Precision support in all libraries

CUDA Toolkit 3.x

- cuSPARSE
- cuRAND
- printf()
- malloc()

CUDA Toolkit 4.x

- Thrust
- NPP
- assert()

Overall library ecosystem

- Linear Algebra
  - cuBLAS, cuSPARSE, CUSP, MAGMA, CULA Dense, CULA Sparse, phiGEMM, libFLAME, FMSLib
- Math / Numerics
  - cuRAND, NAG, ArrayFire, IMSL
- Algorithms
  - Thrust, CUDPP, b40c
- Image Processing and Computer Vision
  - NPP, OpenVIDIA, OpenCV, cuvi, VSIPL
- Signal Processing
  - cuFFT , NukadaFFT
- Finance
  - Kooderive

New functionality in 4.1

- Over 1,000 image processing functions added to NPP
- Sparse tri-diagonal solver added to cuSPARSE
- Widely used random number generators added to cuRAND
  - MRG32k3a and Mersenne Twister (MTGP, based on MT11213)
- Additions to math.h
  - Bessel functions: j0, j1, jn, y0, y1, yn
  - Scaled complementary error function: erfcx
  - Average and rounded average: __{u}hadd, __{u}rhadd

Productivity improvements in 4.1

- Boost-style placeholders added to Thrust
  - These allow functors to be concisely defined inline
  - Example: *saxpy* in 1 line:

```
thrust::transform(x.begin(), x.end(), y.begin(), y.begin(), a * _1 + _2);
```

Performance improvements in 4.1 (for Fermi)

- “Batched” GEMM API in cuBLAS performs many small matrix multiplies
- Sparse matrix-vector multiply in cuSPARSE is up to 2x faster using ELL/HYB format
- Significant optimization to IEEE floating-point operations implemented in software (i.e., divide, reciprocal, and square-root)
- Many other improvements are documented in the release notes