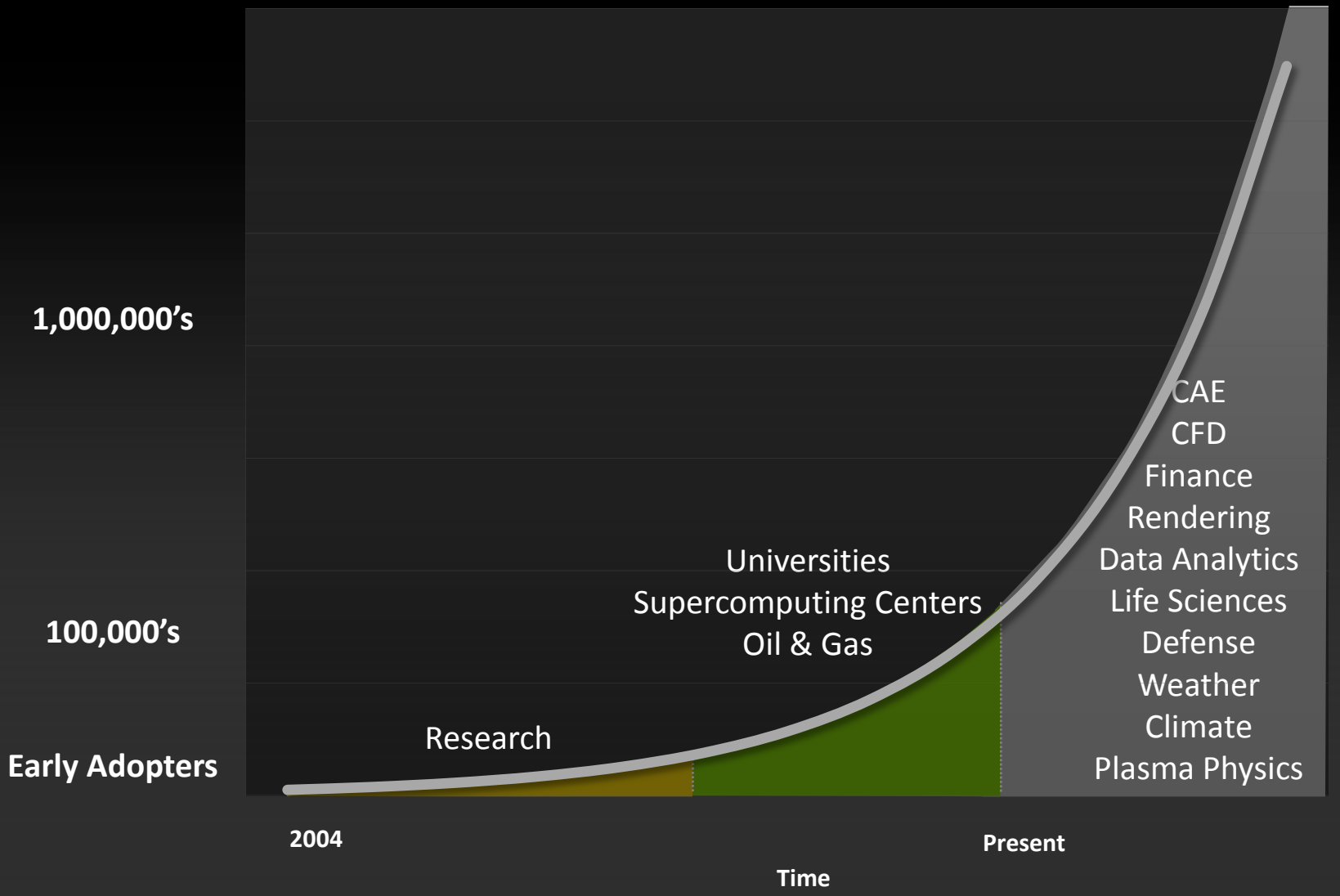
Introduction to OpenACC Directives

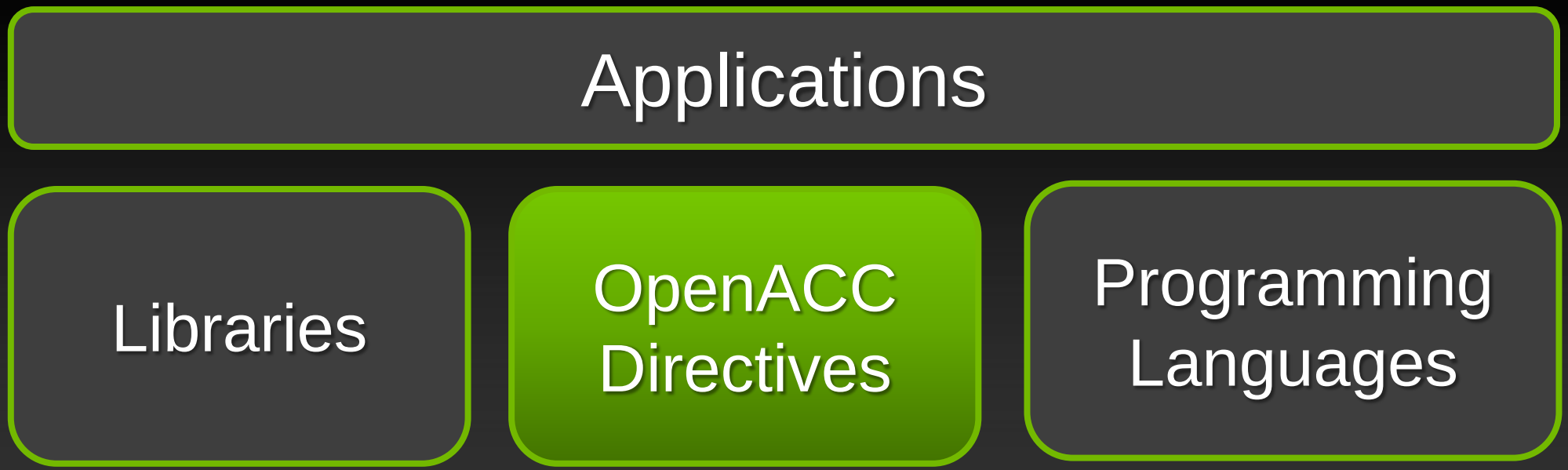
Duncan Poole, NVIDIA
Thomas Bradley, NVIDIA

NVIDIA

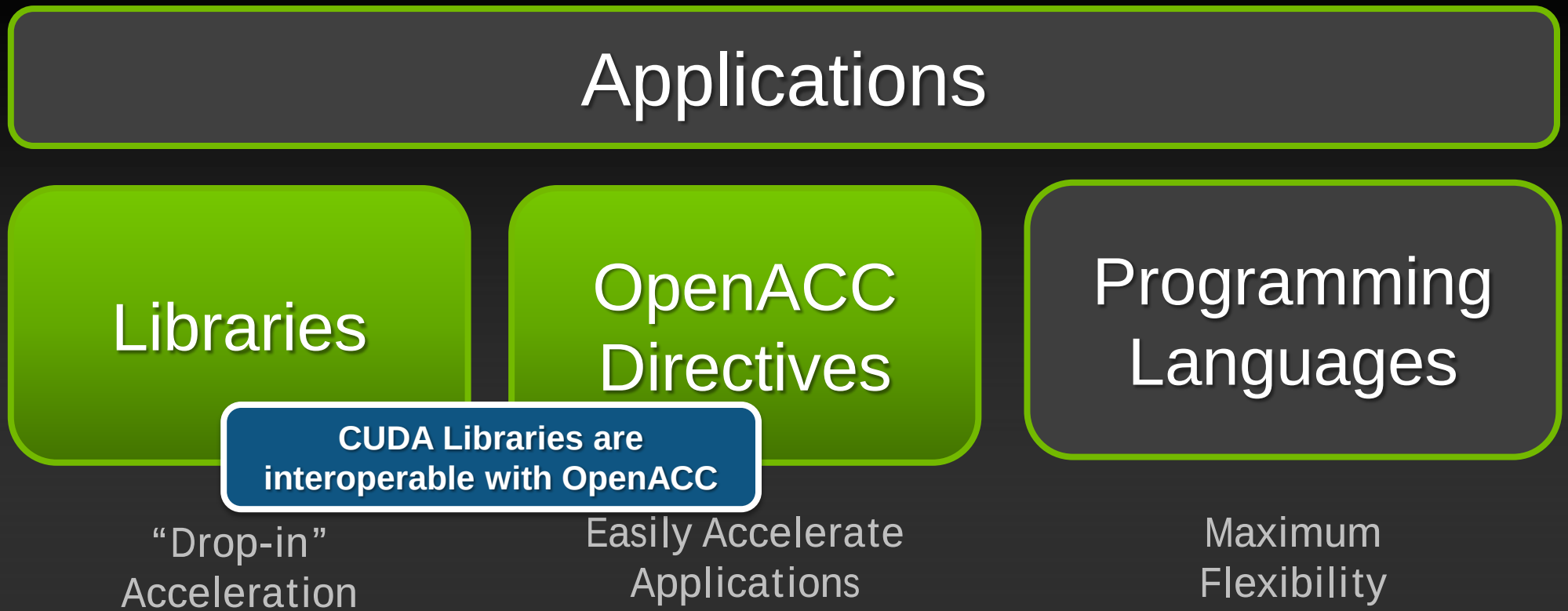
GPUs Reaching Broader Set of Developers



3 Ways to Accelerate Applications



3 Ways to Accelerate Applications



3 Ways to Accelerate Applications

Applications

Libraries

OpenACC Directives

Programming Languages

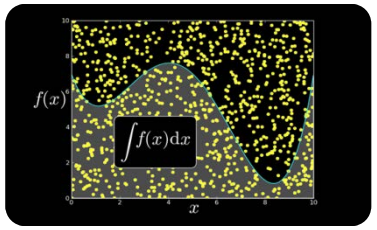
CUDA Languages are interoperable with OpenACC, too!

"Drop-in" Acceleration

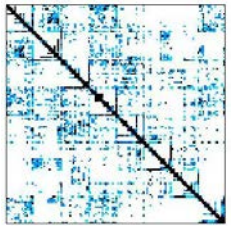
Easily Ac… Applications

…imum Flexibility

NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE

NVIDIA NPP

GPU VSIPL

Vector Signal Image Processing

CULA tools

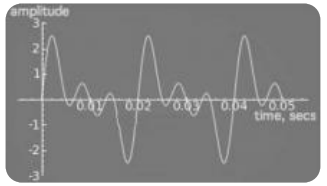
GPU Accelerated Linear Algebra

MAGMA

ICL UT

Matrix Algebra on GPU and Multicore

open source initiative



NVIDIA cuFFT

ROGUE WAVE SOFTWARE

IMSL Library

libjacket

Building-block Algorithms for CUDA

CUSP

Sparse Linear Algebra

open source initiative

Thrust

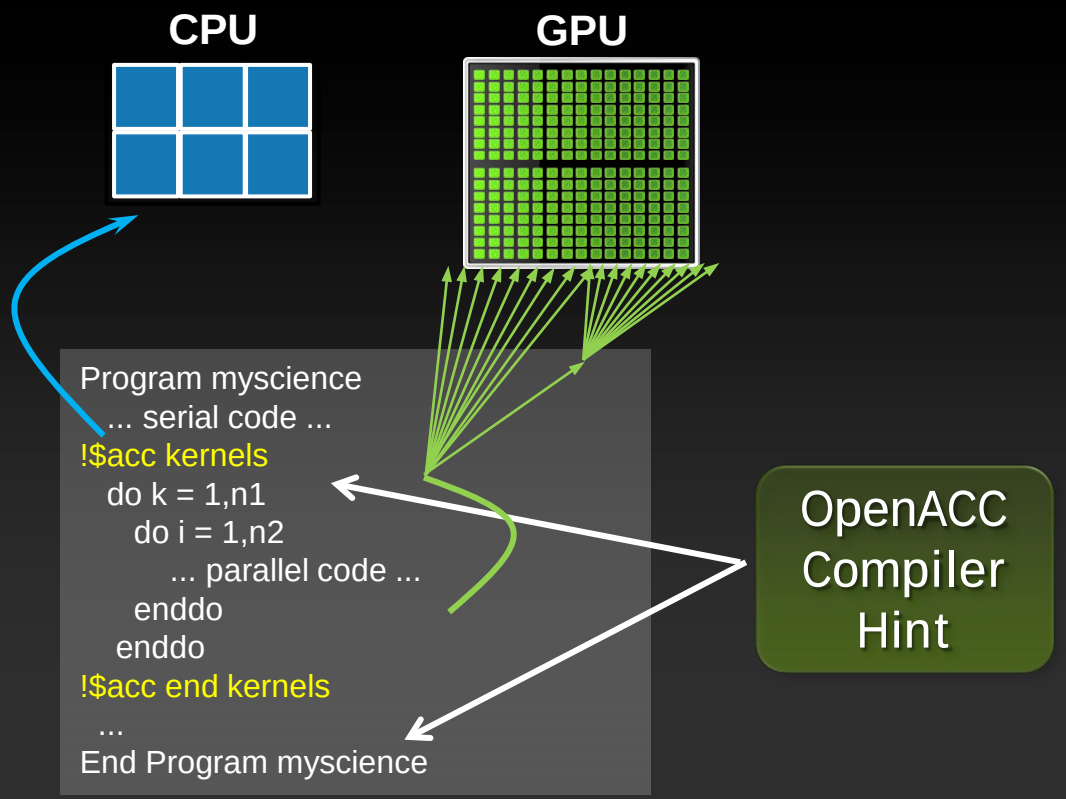
C++ STL Features for CUDA

open source initiative

GPU Accelerated Libraries

"Drop-in" Acceleration for Your Applications

OpenACC Directives



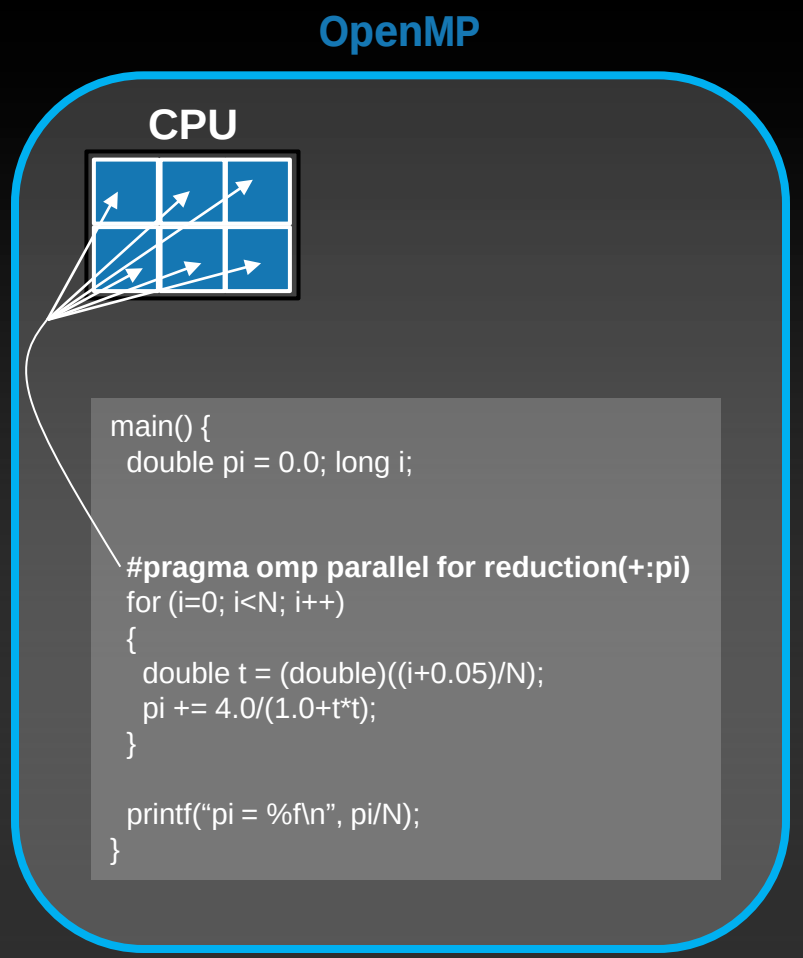
Your original Fortran or C code

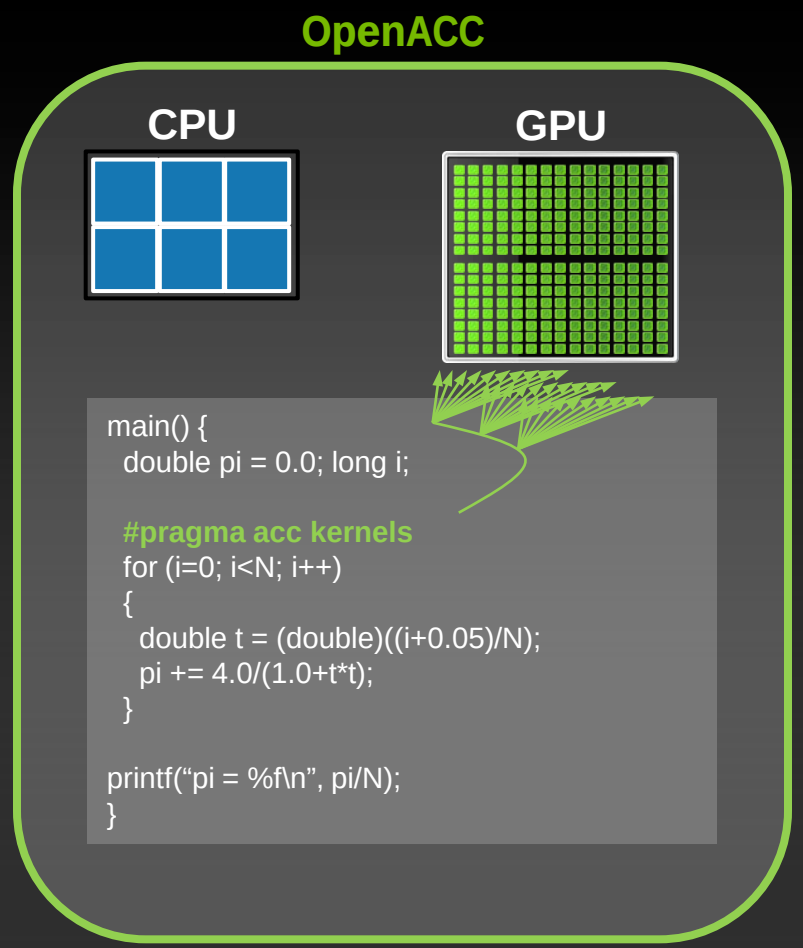
Simple Compiler hints

Compiler Parallelizes code

Works on many-core GPUs & multicore CPUs

Familiar to OpenMP Programmers





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

- **Simple:** 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

High-level

- Compiler directives to specify parallel regions in C & Fortran
  - Offload parallel regions
  - Portable across OSes, host CPUs, accelerators, and compilers
- Create high-level heterogeneous programs
  - Without explicit accelerator initialization
  - Without explicit data or program transfers between host and accelerator

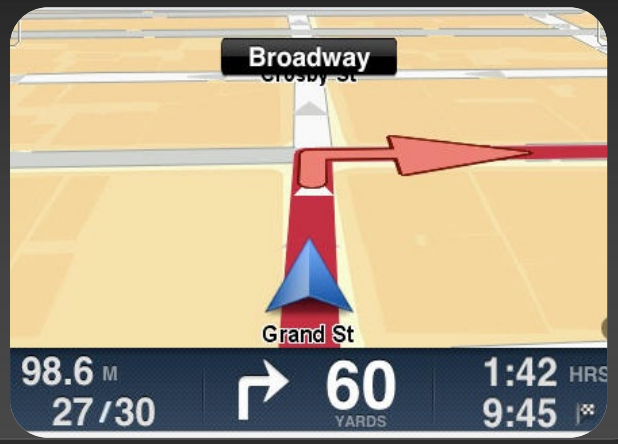
High-level… with low-level access

- Programming model allows programmers to start simple
- Compiler gives additional guidance
  - Loop mappings, data location, and other performance details
- Compatible with other GPU languages and libraries
  - Interoperate between CUDA C/Fortran and GPU libraries
  - e.g. CUFFT, CUBLAS, CUSPARSE, etc.

Directives: Easy & Powerful

Real-Time Object Detection

Global Manufacturer of Navigation Systems



5x in 40 Hours

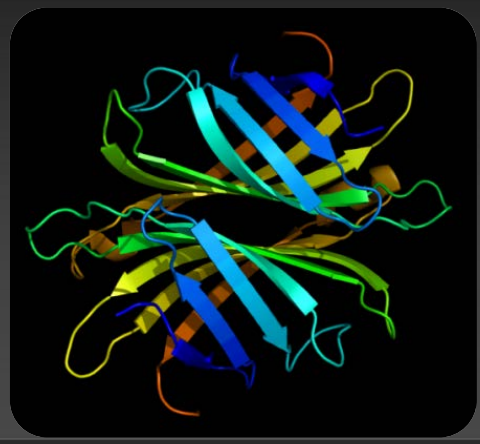
Valuation of Stock Portfolios using Monte Carlo

Global Technology Consulting Company

2x in 4 Hours

Interaction of Solvents and Biomolecules

University of Texas at San Antonio



5x in 8 Hours

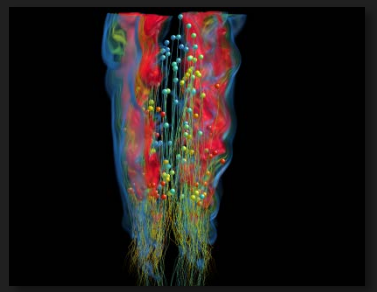
> "Optimizing code with directives is quite easy, especially compared to CPU threads or writing CUDA kernels. The most important thing is avoiding restructuring of existing code for production applications."
>
> *-- Developer at the Global Manufacturer of Navigation Systems*

Focus on Exposing Parallelism

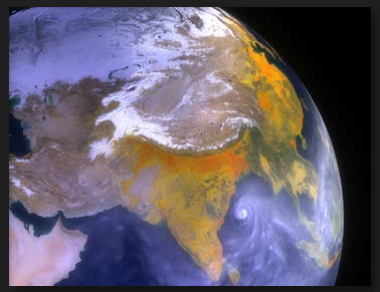
With Directives, tuning work focuses on *exposing parallelism*, which makes codes inherently better

Example: Application tuning work using directives for new Titan system at ORNL

S3D
Research more efficient combustion with next-generation fuels



- Tuning top 3 kernels (90% of runtime)
- *3 to 6x faster on CPU+GPU vs. CPU+CPU*
- But also improved all-CPU version by 50%



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

- Tuning top key kernel (50% of runtime)
- 6.5x faster on CPU+GPU vs. CPU+CPU
- Improved performance of CPU version by 100%

A Very Simple Example: 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)
...
```

Jacobi Iteration: C Code

```
while ( err > tol && iter < iter_max ) {
  err=0.0;


  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }


  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

Iterate until converged

Iterate across matrix elements

Calculate new value from neighbors

Compute max error for convergence

Naïve swap input/output arrays

Jacobi Iteration: OpenMP C Code

```
while ( err > tol && iter < iter_max ) {
  err=0.0;

#pragma omp parallel for shared(m, n, Anew, A) reduction(max:err)
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      err = max(err, abs(Anew[j][i] - A[j][i]);
    }
  }

#pragma omp parallel for shared(m, n, Anew, A)
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

Parallelize loop across CPU threads

Parallelize loop across CPU threads

Jacobi Iteration: OpenACC C Code


```
#pragma acc data copy(A), create(Anew)
while ( err > tol && iter < iter_max ) {
  err=0.0;

#pragma acc kernels reduction(max:err)
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      err = max(err, abs(Anew[j][i] - A[j][i]);
    }
  }

#pragma acc kernels
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

Copy A in at beginning of loop, out at end. Allocate Anew on accelerator

Performance

CPU: Intel Xeon X5680
6 Cores @ 3.33GHz

GPU: NVIDIA Tesla M2090

Execution	Time (s)	Speedup
CPU 1 OpenMP thread	69.80	--
CPU 2 OpenMP threads	44.76	1.56x
CPU 4 OpenMP threads	39.59	1.76x
CPU 6 OpenMP threads	39.71	1.76x
OpenACC GPU	9.78	4.06x (7.14x)

vs. 1 CPU core

vs. 6 CPU cores (1 CPU core)

Further speedups

- OpenACC allows more detailed control over parallelization
  - Using gang, worker, and vector clauses
- By understanding more about OpenACC execution model and GPU hardware organization, we can get higher speedups on this code
- By understanding bottlenecks in the code via profiling and compiler feedback, we can reorganize the code for higher performance

OpenACC Specification and Website

- Full OpenACC 1.0 Specification available online

www.openacc.org

- Quick reference card also available

- Beta implementations available now from PGI, Cray, and CAPS

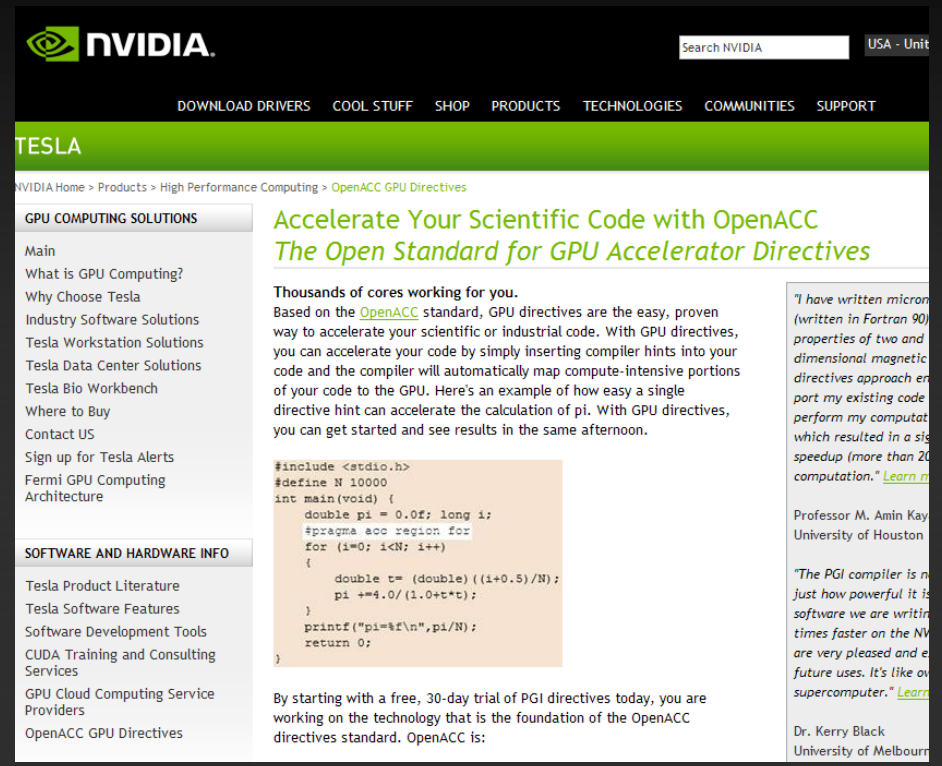
Start Now with OpenACC Directives

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

PGI®

OpenACC Talks on GTC-On-Demand

- Tutorials Monday (CAPS, NVIDIA & PGI)
  - Mark Harris: Getting Started with OpenACC
  - Cliff Wooley: Profiling
  - Michael Wolfe: Advanced Topics
  - Francois Bodin: Programming Many-core Using Directives
- Talks
  - CAPS (Francois Bodin)
  - Cray (James Beyer, Luiz DeRose)
  - PGI (Brent Leback)


Thank you
dpoole@nvidia.com
tbradley@nvidia.com
NVIDIA