GPUs Reaching Broader Set of Developers

- Early Adopters
  - Research
- Universities
- Supercomputing Centers
- Oil & Gas
- CAE
- CFD
- Finance
- Rendering
- Data Analytics
- Life Sciences
- Defense
- Weather
- Climate
- Plasma Physics

1,000,000's
100,000's

2004
Present
3 Ways to Accelerate Applications with GPUs

Applications

Libraries

“Drop-in” Acceleration

Directives

Quickly Accelerate Existing Applications

Programming Languages

Maximum Performance
Directives: Add A Few Lines of Code

### OpenMP

```c
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);
}
```

### OpenACC

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

    #pragma acc parallel
    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.

“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.”

Easy, Fast, Portable

“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.”

http://www.openacc-standard.org/
OpenACC

- Compiler directives to specify parallel regions in C, C++, Fortran
  - OpenACC compilers offload parallel regions from host to accelerator
  - 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

- Programming model allows programmers to start simple
  - Enhance with additional guidance for compiler on loop mappings, data location, and other performance details
OpenACC Specification and Website

- Full OpenACC 1.0 Specification available online
  
  http://www.openacc-standard.org

- Quick reference card also available

- First implementations to appear mid-2012

- Current PGI, Cray, and CAPS compilers all have accelerator directives (precursors to OpenACC)
Small Effort. Real Impact.

Large Oil Company
3x in 7 days
Solving billions of equations iteratively for oil production at world’s largest petroleum reservoirs

Univ. of Houston
Prof. M.A. Kayali
20x in 2 days
Studying magnetic systems for innovations in magnetic storage media and memory, field sensors, and biomagnetism

Uni. Of Melbourne
Prof. Kerry Black
65x in 2 days
Better understand complex reasons by lifecycles of snapper fish in Port Phillip Bay

Ufa State Aviation
Prof. Arthur Yuldashev
7x in 4 Weeks
Generating stochastic geological models of oilfield reservoirs with borehole data

GAMESS-UK
Dr. Wilkinson, Prof. Naidoo
10x
Used for various fields such as investigating biofuel production and molecular sensors.

* Achieved using the PGI Accelerator Compiler
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%
while ( error > tol && iter < iter_max ) {
    error=0.f;

    for( int j = 1; j < n-1; j++ ) {
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1] 
                             + A[j-1][i] + A[j+1][i] );
            error = max(error, 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++;
}
while ( error > tol && iter < iter_max ) {
    error=0.f;
#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++) {
            Anew[j][i] = 0.25f * (A[j][i+1] + A[j][i-1]
                                  + A[j-1][i] + A[j+1][i]);
            error = max(error, 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++;
}
OpenACC GPU Implementation

```c
#pragma acc data copy(A, Anew)
while ( error > tol && iter < iter_max ) {
    error=0.f;
#pragma acc parallel
    for( int j = 1; j < n-1; j++ ) {
        for(int i = 1; i < m-1; i++) {
            error = max(error, abs(Anew[j][i] - A[j][i]));
        }
    }
#pragma acc parallel
    for( int j = 1; j < n-1; j++ ) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }
    iter++;
}
```

- Copy arrays into GPU within region
- Parallelize loops with GPU kernels
- Data automatically copied back to at end of region
Over 2x speed compared to 8-Core OpenMP code with just 3 directives. Note: tested with PGI Accelerator directives.
Jacobi Relaxation (Fortran)

iter = 0
do while ( err .gt tol .and. iter .gt. iter_max )

iter = iter + 1
err = 0.0

do j=1,m
    do i=1,n
        Anew(i,j) = 0.25 * (A(i+1,j) + A(i-1,j) + A(i,j-1) + A(i, j+1)
        err = max( err, abs(Anew(i,j)-A(i,j)) )
    end do
end do

if( mod(iter,100).eq.0 .or. iter.eq.1 ) print*, iter, err
A = Anew
end do
iter = 0
do while ( err .gt tol .and. iter .gt. iter_max )

   iter = iter + 1
   err = 0.0
!$omp parallel do shared(m,n,Anew,A) reduction(max:err)
   do j=1,m
      do i=1,n
         Anew(i,j) = 0.25 * (A(i+1,j) + A(i-1,j) + A(i,j-1) + A(i, j+1)
         err = max( err, abs(Anew(i,j)-A(i,j)) )
      end do
   end do
!$omp end parallel do
if( mod(iter,100).eq.0 ) print*, iter, err
A = Anew
end do
OpenACC GPU Implementation (Fortran)

```fortran
!$acc data copy(A,Anew)
iter = 0
do while ( err .gt tol .and. iter .gt. iter_max )
    iter = iter + 1
    err = 0.0
!$acc parallel reduction( max:err )
    do j=1,m
        do i=1,n
            Anew(i,j) = 0.25 * (A(i+1,j) + A(i-1,j) + A(i,j-1) + A(i,j+1)
            err = max( err, abs(Anew(i,j)-A(i,j)) )
        end do
    end do
!$acc end parallel
    if( mod(iter,100).eq.0 ) print*, iter, err
!$acc parallel
    A = Anew
!$acc end parallel
end do
!$acc end data
```

- **Copy arrays into GPU memory within region**
- **Parallelise code inside region**
- **Close off parallel region**
- **Close off data region, copy data back**
Finding Parallelism in your code

- (Nested) for loops are best for parallelization
- Large loop counts needed to offset GPU/memcpy overhead
- Iterations of loops must be independent of each other
- Compiler must be able to figure out sizes of data regions
  - Can use directives to explicitly control sizes
- Pointer arithmetic should be avoided if possible
  - Use subscripted arrays, rather than pointer-indexed arrays.
- Function calls within accelerated region must be inlineable.
Tips and Tricks

- (PGI) Use time option to learn where time is being spent
  -ta=nvidia,time
- Eliminate pointer arithmetic
- Inline function calls in directives regions
- Use contiguous memory for multi-dimensional arrays
- Use data regions to avoid excessive memory transfers
- Conditional compilation with _OPENACC macro
Basic Concepts

CPU Memory → Transfer data → GPU Memory

CPU → PCI Bus ← GPU

For efficiency, decouple data movement and compute off-load
Directive Syntax

- **Fortran**
  
  ```fortran
  !$acc directive [clause [,] clause] ...
  ```

  Often paired with a matching end directive surrounding a structured code block

  ```fortran
  !$acc end directive
  ```

- **C**

  ```c
  #pragma acc directive [clause [,] clause] ...
  ```

  Often followed by a structured code block
DATA MANAGEMENT
Data Construct

Fortran

```fortran
!$acc data [clause ...]
   structured block
!$acc end data
```

C

```c
#pragma acc data [clause ...]
{ structured block }
```

General Clauses

```c
if( condition )
async( expression )
```

Manage data movement. Data regions may be nested.
**Data Clauses**

`copy ( list )` Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.

`copyin ( list )` Allocates memory on GPU and copies data from host to GPU when entering region.

`copyout ( list )` Allocates memory on GPU and copies data to the host when exiting region.

`create ( list )` Allocates memory on GPU but does not copy.

`present ( list )` Data is already present on GPU from another containing data region.

and `present_or_copy [in|out]`, `present_or_create`, `deviceptr`. 
#pragma acc data copyin(A), create(Anew)
while ( error > tol && iter < iter_max ) {
    error=0.f;
    #pragma acc parallel
    for( int j = 1; j < n-1; j++ ) {
        for(int i = 1; i < m-1; i++) {
            error = max(error, abs(Anew[j][i] - A[j][i]));
        }
    }
    #pragma acc parallel
    for( int j = 1; j < n-1; j++ ) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }
    iter++;
}
Improved OpenACC GPU Implementation

```fortran
!$acc data copyin(A), create(Anew)
iter = 0
do while ( err .gt tol .and. iter .gt. iter_max )

    iter = iter + 1
    err = 0.0
!$acc parallel reduction( max:err )
    do j=1,m
        do i=1,n
            Anew(i,j) = 0.25 * ( A(i+1,j ) + A(i-1,j ) &
                              A(i,  j-1) + A(i,  j+1)
            err = max( err, abs(Anew(i,j)-A(i,j)) )
        end do
    end do
!$acc end parallel
    if( mod(iter,100).eq.0 ) print*, iter, err
    A = Anew
end do
!$acc end data
```
Update Directive

Fortran

!$acc update [clause ...]

Clauses

host( list )
device( list )

C

#pragma acc update [clause ...]

if( expression )
async( expression )

Move data from GPU to host, or host to GPU. Data movement can be conditional, and asynchronous.
WORK MANAGEMENT
Parallel Construct

Fortran

!$acc parallel [clause …]
   structured block
!$acc end parallel

Clauses

if( condition )
async( expression )
num_gangs( expression )
num_workers( expression )
vector_length( expression )

C

#pragma acc parallel [clause …]
   { structured block }

private( list )
firstprivate( list )
reduction( operator:list )

Any data clause
<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>num_gangs (expression)</code></td>
<td>Controls how many parallel gangs are created (CUDA <code>gridDim</code>).</td>
</tr>
<tr>
<td><code>num_workers (expression)</code></td>
<td>Controls how many workers are created in each gang (CUDA <code>blockDim</code>).</td>
</tr>
<tr>
<td><code>vector_length (list)</code></td>
<td>Controls vector length of each worker (SIMD execution).</td>
</tr>
<tr>
<td><code>private(list)</code></td>
<td>A copy of each variable in list is allocated to each gang.</td>
</tr>
<tr>
<td><code>firstprivate(list)</code></td>
<td>Private variables initialized from host.</td>
</tr>
<tr>
<td><code>reduction(operator:list)</code></td>
<td>Private variables combined across gangs.</td>
</tr>
</tbody>
</table>
```c
#pragma acc data copyin(A), create(Anew)
while ( error > tol && iter < iter_max ) {
    error=0.f;
#pragma acc parallel reduction( max:error )
    for( int j = 1; j < n-1; j++ ) {
        for(int i = 1; i < m-1; i++) {
            Anew[j][i] = 0.25f * (A[j][i+1] + A[j][i-1]
                              + A[j-1][i] + A[j+1][i]);
            error = max(error, abs(Anew[j][i] - A[j][i]));
        }
    }
#pragma acc parallel
    for( int j = 1; j < n-1; j++ ) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    } 
    iter++;  }
```
More Parallelism (Fortran)

```fortran
!$acc data copyin(A), create(Anew)
iter = 0
do while ( err .gt tol .and. iter .gt. iter_max )

    iter = iter + 1
    err = 0.0

!$acc parallel reduction( max:err )
    do j=1,m
        do i=1,n
            Anew(i,j) = 0.25 * ( A(i+1,j ) + A(i-1,j ) &
                                A(i,  j-1) + A(i,  j+1)
            err = max( err, abs(Anew(i,j)-A(i,j)) )
        end do
    end do
!$acc end parallel

    if( mod(iter,100).eq.0 ) print*, iter, err
!$acc parallel
    A = Anew
!$acc end parallel
end do
!$acc end data
```

Find maximum over all iterations

Add second parallel region inside data region
Kernels Construct

Fortran
!$acc kernels [clause ...]
   structured block
!$acc end kernels

C
#pragma acc kernels [clause ...]
{ structured block }

Clauses
if( condition )
async( expression )

Any data clause
Kernels Construct

Each loop executed as a separate kernel on the GPU.

```fortran
!$acc kernels
  do i=1,n
    a(i) = 0.0
    b(i) = 1.0
    c(i) = 2.0
  end do

  do i=1,n
    a(i) = b(i) + c(i)
  end do

!$acc end kernels
```

kernel 1

kernel 2
Loop Construct

Fortran

```fortran
!$acc loop [clause ...]
  loop
!$acc end loop
```

Combined directives

```fortran
!$acc parallel loop [clause ...]
!$acc kernels loop [clause ...]
```

C

```c
#pragma acc loop [clause ...]
{ loop }
```

Detailed control of the parallel execution of the following loop.
Loop Clauses

collapse( \( n \) )
Applies directive to the following \( n \) nested loops.

seq
Executes the loop sequentially on the GPU.

private( list )
A copy of each variable in list is created for each iteration of the loop.

reduction( operator:list )
private variables combined across iterations.
Loop Clauses Inside parallel Region

**gang**
Shares iterations across the gangs of the parallel region.

**worker**
Shares iterations across the workers of the gang.

**vector**
Execute the iterations in SIMD mode.
Loop Clauses Inside kernels Region

**gang** [(num_gangs)]  Shares iterations across across at most num_gangs gangs.

**worker** [(num_workers)]  Shares iterations across at most num_workers of a single gang.

**vector** [(vector_length)]  Execute the iterations in SIMD mode with maximum vector_length.

**independent**  Specify that the loop iterations are independent.
#pragma acc data copyin(A), create(Anew)
while ( error > tol && iter < iter_max ) {
    error=0.f;

#pragma acc kernels loop reduction( max:error ), gang(32), worker(16)
    for( int j = 1; j < n-1; j++ ) {
#pragma acc loop gang(16), worker(32)
        for( int i = 1; i < m-1; i++ ) {
            error = max(error, abs(Anew[j][i] - A[j][i]));
        }
    }
#pragma acc kernels loop
    for( int j = 1; j < n-1; j++ ) {
#pragma acc loop gang(16), worker(32)
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    iter++;
}
More Performance (Fortran)

```fortran
!$acc data copyin(A), create(Anew)
iter = 0
do while ( err .gt tol .and. iter .gt. iter_max )

    iter = iter + 1
    err = 0.0
!$acc kernels loop reduction( max:err ), gang(32), worker(8)
    do j=1,m
        do i=1,n
            Anew(i,j) = 0.25 * ( A(i+1,j ) + A(i-1,j ) &
                A(i, j-1) + A(i, j+1)
            err = max( err, abs(Anew(i,j)-A(i,j)) )
        end do
    end do
!$acc end kernels loop
    if( mod(iter,100).eq.0 ) print*, iter, err
!$acc parallel
    A = Anew
!$acc end parallel
end do
!$acc end data
```

30% faster than default schedule
OTHER SYNTAX
<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cache construct</code></td>
<td>Cache data in software managed data cache (CUDA shared memory).</td>
</tr>
<tr>
<td><code>host_data construct</code></td>
<td>Makes the address of device data available on the host.</td>
</tr>
<tr>
<td><code>wait directive</code></td>
<td>Waits for asynchronous GPU activity to complete.</td>
</tr>
<tr>
<td><code>declare directive</code></td>
<td>Specify that data is to allocated in device memory for the duration of an implicit data region created during the execution of a subprogram.</td>
</tr>
</tbody>
</table>
Runtime Library Routines

**Fortran**

```fortran
use openacc
#include "openacc_lib.h"

acc_get_num_devices
acc_set_device_type
acc_get_device_type
acc_set_device_num
acc_get_device_num
acc_async_test
acc_async_test_all
```

**C**

```c
#include "openacc.h"

acc_async_wait
acc_async_wait_all
acc_shutdown
acc_on_device
acc_malloc
acc_free
```
Environment and Conditional Compilation

ACCDEVICE device

Specifies which device type to connect to.

ACCDEVICE_NUM num

Specifies which device number to connect to.

_OPENACC

Preprocessor directive for conditional compilation.