OpenACC 2.0
versus
OpenMP 4.0 device constructs

James Beyer, Ph.D.
Contents

- Background or “how to” resources
- High level introduction
- Detailed comparison
- Code example comparisons
- Final thoughts
Background or “how to” resources

● Videos
  ● www.openacc.org/Videos
  ● www.cray.com/About/Videos.aspx
    ● Search by keyword: openacc
  ● www.youtube.com
    ● Search OpenMP

● Google search of “OpenMP tutorials” produced 80,000+ hits
● Google search of “OpenACC tutorials” produced 19,000+ hits
High level intro

- **Heritage**
  - OpenMP
    - 15+ years of history
  - OpenACC
    - 2+ years of history

- **Programming model**
  - Directives and function calls
  - Source does not need to change
  - Code still compiles for “host-only” execution

- **Execution model**
  - Host directed

- **Memory model**
  - Weak memory model
  - No synchronization at gang/team level
  - Separate or shared memory

- **Data motion control**
  - `present_or_*`
  - OpenMP
    - Structured
  - OpenACC
    - Structured
    - Unstructured
OpenACC compared to OpenMP

OpenACC
- Parallel (offload)
  - Parallel (multiple “threads”)
- Kernels
- Data
- Loop
- Host data
- Cache
- Update
- Wait
- Declare

OpenMP
- Target
- Team/Parallel
- Target Data
- Distribute/Do/for/Simd
- Target Update
- Declare target
OpenACC compared to OpenMP continued

<table>
<thead>
<tr>
<th>OpenACC</th>
<th>OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>● enter data</td>
<td>●</td>
</tr>
<tr>
<td>● exit data</td>
<td>● declare target</td>
</tr>
<tr>
<td>● data api</td>
<td>● Parallel in parallel or team</td>
</tr>
<tr>
<td>● routine</td>
<td>●</td>
</tr>
<tr>
<td>● async wait</td>
<td>●</td>
</tr>
<tr>
<td>● parallel in parallel</td>
<td>●</td>
</tr>
<tr>
<td>● Tile</td>
<td>●</td>
</tr>
<tr>
<td>● Device_type</td>
<td>●</td>
</tr>
</tbody>
</table>
OpenACC compared to OpenMP continued

<table>
<thead>
<tr>
<th>OpenACC</th>
<th>OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic</td>
<td>Atomic</td>
</tr>
<tr>
<td></td>
<td>Critical sections</td>
</tr>
<tr>
<td></td>
<td>Master</td>
</tr>
<tr>
<td></td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>Tasks</td>
</tr>
<tr>
<td></td>
<td>barrier</td>
</tr>
<tr>
<td></td>
<td>get_thread_num</td>
</tr>
<tr>
<td></td>
<td>get_num_threads</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
OpenACC compared to OpenMP

- **OpenACC 2.0**
  - `acc_copy(in|out)( ptr, bytes )`
  - `acc_create( ptr, bytes )`
  - `acc_delete( ptr, bytes )`
  - `acc_is_present( ptr, bytes )`
  - `acc_update_(device|local)( ptr, bytes )`
  - `acc_deviceptr( ptr )`
  - `acc_hostptr( devptr )`
  - `acc_[un]map_data( devptr, hostptr, bytes )`
  - `acc_memcpy_(to|from)_device`
  - If there is a directive/clause there is likely an API routine
  - All 1.0 environment APIs still present
    - `acc_async_test( id )` ...

- **OpenMP**
  - Most environment APIs contained in OpenACC 1.0
Code Example Comparisons
Device-specific tuning, multiple devices

- **Device_type** (type)
- Similar to an `#if def`
- Compiler can generate code for all targets from single invocation
- Which clauses are allowed to follow clause are limited
  - No data clauses

```c
PROGRAM main
  INTEGER :: a(N), b(N)
  <stuff>
$acc parallel loop &
& device_type(nvidia) num_gangs(200) &
& device_type(host) num_gangs(16)
  DO i = 1,N
    a(i) = a(i) + rhs(i)
  ENDDO
$acc end parallel loop
<stuff>
END PROGRAM main
```

```c
PROGRAM main
  INTEGER :: a(N), b(N)
  <stuff>
$omp target teams distribute &
& num_teams( x )
  DO i = 1,N
    a(i) = a(i) + rhs(i)
  ENDDO
$omp end target parallel do
<stuff>
END PROGRAM main
```
These are not equivalent!!!
- OpenMP workshare construct not defined in the same way as kernels
- OpenMP must insert a “barrier” between statements
- For OpenACC Array a(:) possibly unnecessarily moved from and to GPU between kernels
  - "data sloshing"
- Code still compile-able for CPU
PROGRAM main
    INTEGER :: a(N)
    <stuff>
    call init(a)
    !$acc parallel loop
    DO i = 1,N
       a(i) = i
    ENDDO
    !$acc end parallel loop
    !$acc parallel loop
    !$acc parallel loop
    DO i = 1,N
       a(i) = 2*a(i)
    ENDDO
    !$acc end parallel loop
    call fini(a)
    <stuff>
END PROGRAM main

SUBROUTINE init(b)
    INTEGER :: b(:)
    !$acc enter data pcreate(b)
END SUBROUTINE init

SUBROUTINE fini(b)
    INTEGER :: b(:)
    !$acc exit data copyout(b)
END SUBROUTINE init
Subprograms sharing GPU (Error vs Fix-up)

**OpenACC**

```
PROGRAM main
  INTEGER :: a(N)
  <stuff>
!$acc data copyout(a) if( test )
!$acc parallel loop
  DO i = 1,N
    a(i) = i
  ENDDO
!$acc end parallel loop
  CALL double_array(a)
!$acc end data
  <stuff>
END PROGRAM main

SUBROUTINE double_array(b)
  INTEGER :: b(N)
!$acc parallel loop present(b)
  DO i = 1,N
    b(i) = double_scalar(b(i))
  ENDDO
!$acc end parallel loop
END SUBROUTINE double_array

INTEGER FUNCTION double_scalar(c)
  INTEGER :: c
  double_scalar = 2*c
END FUNCTION double_scalar
```

**OpenMP**

```
PROGRAM main
  INTEGER :: a(N)
  <stuff>
!$omp target data map(out:a) if( test )
!$omp target parallel do
  DO i = 1,N
    a(i) = i
  ENDDO
$omp end target parallel loop
  CALL double_array(a)
$omp end target data
  <stuff>
END PROGRAM main

SUBROUTINE double_array(b)
  INTEGER :: b(N)
!$omp parallel loop map(tofrom:b)
  DO i = 1,N
    b(i) = double_scalar(b(i))
  ENDDO
!$omp end parallel loop
END SUBROUTINE double_array

INTEGER FUNCTION double_scalar(c)
  INTEGER :: c
  double_scalar = 2*c
END FUNCTION double_scalar
```
Interoperability with CUDA

OpenACC

```c
__global__ void dbl_knl(int *c) {
    int i = blockIdx.x*blockDim.x+threadIdx.x;
    if (i < N) c[i] *= 2;
}

extern "C" void dbl_cuda_(int *b_d) {
    cudaMemcpy(b_d, a, N*sizeof(int), cudaMemcpyDeviceToHost);
    cudaThreadSynchronize();
    dbl_knl<<<NBLOCKS,BSIZE>>>(b_d);
    cudaMemcpy(a, b_d, N*sizeof(int), cudaMemcpyHostToDevice);
    cudaThreadSynchronize();
}
```

OpenMP

```c
PROGRAM main
    INTEGER :: a(N)
    <stuff>
$acc data copy(a)
! <Populate a(:) on device
! as before>
$acc host_data use_device(a)
    CALL dbl_cuda_(a)
$acc end host_data
$acc end data
    <stuff>
END PROGRAM main
```
Tile clause

OpenACC

```c
#include <openacc.h>

!$acc loop tile(64,4) gang vector
  do i = 1, n
    do j = 1, m
      a(j,i) = (b(j-1,i)+b(j+1,i)+ &
                b(j,i-1)+b(j,i+1))*0.25
    enddo
  enddo
enddo

!$acc loop collapse(2) gang
  do i = 1, n, 4
    do j = 1, m, 64
      !$acc loop collapse(2) vector
        do ii = i, min(n,i+4)
          do jj = j, min(m,j+64)
            a(jj,ii) = (b(jj-1,ii)+ &
                        b(jj+1,ii)+ &
                        b(jj,ii-1)+ &
                        b(jj,ii+1))*0.25
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```
cache clause examples

OpenACC

 !$acc loop tile( 64, 16, 1 ) gang & worker vector
DO k = 1,N
    DO j = 1,N
        DO i = 1,N
            !$acc cache( A(i,j,k), &
            !$acc B(i-1:i+1,j-1:j+1,k) )

            A(i,j,k) = B(i, j, k) - &
            ( B(i-1,j-1,k) &
            + B(i-1,j+1,k) &
            + B(i+1,j-1,k) &
            + B(i+1,j+1,k) ) / 5

        ENDDO
    ENDDO
ENDDO
ENDDO
ENDDO
 !$acc end parallel
OpenACC Loop selection

OpenACC

PROGRAM main
  INTEGER :: a(N)
!$acc routine( foo ) worker
  <stuff>
!$acc parallel loop
  DO i = 1,N
    CALL foo(a)
  ENDDO
!$acc end parallel loop
  <stuff>
END PROGRAM main

OpenMP

PROGRAM main
  INTEGER :: a(N)
!$omp delcare target ( foo )
  <stuff>
!$omp target parallel do
  DO i = 1,N
    CALL foo(a)
  ENDDO
!$omp end target parallel do
  <stuff>
!$omp target teams distribute
  DO i = 1,N
    CALL foo(a)
  ENDDO
!$omp end target teams distribute
  <stuff>
END PROGRAM main
Declare Create vs Declare Link

Both OpenACC and OpenMP support the declare create concept
Only OpenACC contains the link concept at this time

```c
float a[100000];
#pragma acc declare create( a )
...
#pragma acc routine gang
void foo() {
#pragma acc loop
for(...) 
a[..] = ...
}
...
void bar() {
#pragma acc update device( a )
#pragma acc parallel
foo();
#pragma acc update self( a )
}
```

```c
float a[100000];
#pragma acc declare link(a)
...
#pragma acc routine gang
void foo() {
#pragma acc loop
for(...) 
a[..] = ...
}
...
void bar() {
!! #pragma acc update device( a ) ERROR!!!
#pragma acc parallel copy( a )
foo();
!! #pragma acc update self( a ) ERROR!!!
}
```
OpenACC bind clause examples

File 1

```c
#pragma acc declare create(a)
extern int a[];
#pragma acc routine(foo) bind(foo_nvidia) gang
extern void foo(int i);
#pragma acc parallel loop gang copy(a)
for(i...)
  foo(i);
```

File 2

```c
#pragma acc routine bind(“foo_worker”) worker
void foo( int i) {
  #pragma acc loop worker vector
  for(…)
  a[..] = ...
}
```
OpenACC High-level async example

```c
!$acc parallel loop async(1)
<Kernel A>
!$acc parallel loop async(2)
<Kernel B>

!$acc wait( 1, 2 ) async( 3 )

!$acc parallel loop async(3)
!! wait( 1, 2 )
<Kernel C>

!$acc parallel loop async(4) &
!$acc wait(3)
<Kernel D>

!$acc parallel loop async(5) &
!$acc wait(3)
<Kernel E>

!$acc wait( 1 )
<Kernel F>
```
OpenMP High-level async example

```c
!$omp task depend(inout:a)  
!$omp target parallel do
 <Kernel A>
!$omp task depend(inout:b)  
!$omp target parallel do
 <Kernel B>
!$omp task depend(inout:c)  
  depend(in:a,b)  
!$omp target parallel do
 <Kernel C>
!$omp task depend(inout:d) depend(in:c)  
!$omp target parallel do
 <Kernel D>
!$omp task depend(inout:e) depend(in:c)  
!$omp target parallel do
 <Kernel E>

!$omp task depend(in:a)  
<Kernel F>
```
Nested Parallelism

- **OpenACC 2.0**
  - Actually simply a deletion of two restrictions
    - OpenACC parallel regions may not contain other parallel regions or kernels regions.
    - OpenACC kernels regions may not contain other parallel regions or kernels regions.
  - Other changes were mainly cosmetic
  - Has significant impact on where objects can be placed in memory.

- **OpenMP**
  - Target constructs not allowed inside of target constructs.
  - Teams constructs not allowed inside of teams constructs
  - ...
  - Only parallel inside of target/teams/parallel
  - May come in next release
Manual deep-copy

```c
struct A_t {
    int n;
    int *x;       // dynamic size n
};
...
struct A_t *A;   // dynamic size 2
/* shallow copyin A[0:2] to device_A[0:2] */
struct A_t *dA = acc_copyin( A, 2*sizeof(struct A_t) );
    int i = 0 ; i < 2 ; i++) {
    /* shallow copyin A[i].x[0:A[i].n] to "orphaned" object */
    int *dx = acc_copyin( A[i].x, A[i].n*sizeof(int) );
    /* fix acc pointer device_A[i].x */
    acc_memcpy_to_device( &dA[i].x, &dx, sizeof(int*));
}
```

- Currently works for C/C++
- Portable in OpenACC 2.0, but not usually practical
- Not in OpenMP
Final thoughts

● **OpenMP is …**
  ● Easier to write codes that will not compile for some devices
  ● All historic OpenMP constructs allowed inside of parallel regions on the device

● **OpenACC is**
  ● Harder to write codes that will not compile for some device
  ● OpenMP NOT allowed inside of construct by most vendors

● …
Upcoming GTC Express Webinars

**October 22** - Introduction to SeqAn, an Open-source C++ Template Library

**October 29** - How to Improve Performance using the CUDA Memory Model and Features of the Kepler

**October 30** - OpenACC 2.0 Enhancements for Cray Supercomputers

**November 5** - Accelerating Face-in-the-Crowd Recognition with GPU Technology

**November 6** - Bright Cluster Manager: A CUDA-ready Management Solution for GPU-based HPC

Register at www.gputechconf.com/gtcexpress
GTC 2014 Call for Posters

Posters should describe novel or interesting topics in

- Science and research
- Professional graphics
- Mobile computing
- Automotive applications
- Game development
- Cloud computing

Call opens October 29

www.gputechconf.com