OpenACC 2.0 Highlights

Jeff Larkin, NVIDIA

OpenACC 2.0 - Highlights

- Procedure calls, separate compilation
- Nested parallelism (support for Dynamic Parallelism)
- Loop tile clause
- Data management features and global data
- Device-specific tuning
- Asynchronous behavior additions
- New API routines
- New atomic contstruct
- New default(none) data clause

Procedure calls – OpenACC 1.0

```
#pragma acc parallel loop
for ( int i=0; i<n; ++i) {
  foo(v,i);
  //must inline foo
}
```

Procedure calls – OpenACC 2.0

```
#pragma acc routine worker
extern void foo(float* v, int i);

#pragma acc parallel loop
for ( int i=0; i<n; ++i) {
  foo(v,i);
  //call on the device
}
```

Tell the compiler the level of parallelism in foo

Procedure calls – OpenACC 2.0

```
#pragma acc routine worker
extern void foo(float* v,int i);

#pragma acc parallel loop
for ( int i=0; i<n; ++i) {
  foo(v,i);
  //call on the device
}
```

```
#pragma acc routine worker
void foo(float* v, int i) {
  #pragma acc loop worker
  for ( int j=0; j<n; ++j) {
    v[i*n+j] = 1.0f/(i*j);
  }
}
```

Nested Parallelism

```
#pragma acc routine
extern void foo(float* v,int i);

#pragma acc parallel loop
for ( int i=0; i<n; ++i) {
  foo(v,i);
  //call on the device and span new
  //threads
}
```

```
#pragma acc routine
void foo(float* v, int i) {
  #pragma acc parallel loop
  for ( int j=0; j<n; ++j) {
    v[i*n+j] = 1.0f/(i*j);
  }
}
```

Loop `tile` Clause

Block loops into 8x8 tiles.

- OpenACC 1.0 does not provide a standard way to decompose loops into 2D threadblocks
  - May better exploit data locality

```
#pragma acc loop tile(8,8)
for ( int i=0; i<n; ++i)
{
   for ( int j=0; j<n; ++j)
   {
      v[i*n+j] = 1.0f/(i*j);
   }
}
```

Data management – global data

```
float a[1000000];

extern void foo(float* v,int i);

for ( int i=0; i<n; ++i) {
  foo(v,i);

}
```

```
extern float a[];

void foo(float* v, int i) {

  for ( int j=0; j<n; ++j) {
    v[i*n+j] = a[i]/(i*j);
  }
}
```

Data management – global data

```
float a[1000000];
#pragma acc declare create(a)

#pragma acc routine worker
extern void foo(float* v,int i);

#pragma acc parallel loop
for ( int i=0; i<n; ++i) {
  foo(v,i);
  //call on the device
}
```

```
extern float a[];
#pragma acc declare create(a)

#pragma acc routine worker
void foo(float* v, int i) {
  #pragma acc loop worker
  for ( int j=0; j<n; ++j) {
    v[i*n+j] = a[i]/(i*j);
  }
}
```

Data management – global data

```
float a[1000000];
#pragma acc declare device_resident(a)

#pragma acc routine worker
extern void foo(float* v,int i);

#pragma acc parallel loop
for ( int i=0; i<n; ++i) {
  foo(v,i);
  //call on the device
}
```

```
extern float a[];
#pragma acc declare device_resident(a)

#pragma acc routine worker nohost
void foo(float* v, int i) {
  #pragma acc loop worker
  for ( int j=0; j<n; ++j) {
    v[i*n+j] = a[i]/(i*j);
  }
}
```

Data management – structured data lifetime (OpenACC 1.0)

```
#pragma acc data copyin(x[0:n]) \
                 create(y[0:n])
{
}
```

Data management – unstructured data lifetime

```
class Matrix {
  Matrix() {
    v = new double[n];


  }
  ~Matrix() {


    delete[] v;}
private:
  double* v;
}
```

```
class Matrix {
  Matrix() {
    v = new double[n];
    #pragma acc enter data \
        create(v[0:n])
  }
  ~Matrix() {
    #pragma acc exit data \
        delete(v[0:n])
    delete[] v;}
private:
  double* v;
}
```

Device-specific tuning – device_type

```
#pragma acc routine worker
extern void foo(float* v,int i);

#pragma acc parallel loop \
  num_workers(384)

for ( int i=0; i<n; ++i) {
  foo(v,i);
}
```

```
#pragma acc routine worker
extern void foo(float* v,int i);

#pragma acc parallel loop \
  device_type(nvidia) num_workers(256)\
  device_type(radeon) num_workers(512)
for ( int i=0; i<n; ++i) {
  foo(v,i);
}
```

OpenACC 1.0 - async clause

```
#pragma acc parallel async(1)
{… /*kernel A*/}
do_something_on_host()
#pragma acc parallel async(2)
{…/*Kernel B*/}
#pragma acc parallel async(2)
{…/*Kernel C*/}
```

The async clause is optional on the parallel and kernels constructs; when there is no async clause, the host process will wait until the parallel or kernels region is complete before executing any of the code that follows the construct. When there is an async clause,

Do not wait for kernel completion

Executes concurrently with kernels

(potentially) executes concurrently with kernels

OpenACC 1.0 – `wait` directive

```
#pragma acc parallel async(1)
{… /*kernel A*/}
do_something_on_host()
#pragma acc parallel async(2)
{…/*Kernel B*/}
#pragma acc parallel async(2)
{…/*Kernel C*/}

#pragma acc wait(1)
#pragma acc parallel async(2)
{…}
```

Wait for kernel A in queue (stream) 1 - blocks host

Schedule new work in queue (stream) 2 that depends on queue 1.

OpenACC 2.0 – `wait` directive with `async` clause

```
#pragma acc parallel async(1)
{… /*kernel A*/}
do_something_on_host()
#pragma acc parallel async(2)
{…/*Kernel B*/}
#pragma acc parallel async(2)
{…/*Kernel C*/}

#pragma acc wait(1)
#pragma acc parallel async(2)
{…}
```

```
#pragma acc parallel async(1)
{… /*kernel A*/}
do_something_on_host()
#pragma acc parallel async(2)
{…/*Kernel B*/}
#pragma acc parallel async(2)
{…/*Kernel C*/}

#pragma acc wait(1) async(2)
#pragma acc parallel async(2)
{…}
```

New API routines

- Improved Data management for C/C++
  - acc_copyin
  - acc_present_or_copyin
  - acc_create
  - acc_present_or_create
  - acc_copyout
  - acc_delete
  - acc_map_data
  - acc_unmap_data
  - acc_deviceptr
  - acc_hostptr
  - acc_is_present
  - acc_memcpy_to_device
  - acc_memcpy_from_device
  - acc_update_device
  - acc_update_self
- Expanded Interoperability with CUDA, OpenCL, & Xeon Phi
  - acc_get_cuda_stream
  - acc_get_current_cuda_device
  - …

OpenACC 2.0 - Highlights

- Procedure calls, separate compilation
- Nested parallelism (support for Dynamic Parallelism)
- Loop tile clause
- Data management features and global data
- Device-specific tuning
- Asynchronous behavior additions
- New API routines
- New atomic contstruct
- New default(none) data clause