WHAT’S NEW IN OPENACC 2.0 AND OPENMP 4.0
JEFF LARKIN, NVIDIA
OUTLINE

- Background on OpenACC and OpenMP
- New features in OpenACC 2.0
  - Routine Directive
  - Unstructured Data regions
  - Atomics
  - Multiple Device Types
- New features in OpenMP 4.0
  - Target Data Construct
  - Target Construct
  - Teams/Distribute
OPENACC 2.0

- OpenACC is a specification for high-level, compiler directives for expressing parallelism for accelerators.
  - Aims to be performance portable to a wide range of accelerators.
  - Multiple Vendors, Multiple Devices, One Specification

- The OpenACC specification was first released in November 2011.
  - Original members: CAPS, Cray, NVIDIA, Portland Group

- OpenACC 2.0 was released in June 2013, expanding functionality and improving portability

- At the end of 2013, OpenACC had more than 10 member organizations
OPENACC EXAMPLE: SAXPY

**SAXPY in C**

```c
void saxpy(int n,
           float a,
           float *x,
           float *restrict y)
{
    #pragma acc parallel loop
    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**

```fortran
subroutine saxpy(n, a, x, y)
  real :: x(n), y(n), a
  integer :: n, i

  !$acc parallel loop
  do i=1,n
      y(i) = a*x(i)+y(i)
  enddo
  !$acc end parallel loop
end subroutine saxpy
```

! Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x, y)
...
OpenMP 4.0

- OpenMP has existed since 1997 as a specification for compiler directives for shared memory parallelism.
- In 2013, OpenMP 4.0 was released, expanding the focus beyond shared memory parallel computers, including accelerators.
- The OpenMP 4.0 `target` construct provides the means to offload data and computation to accelerators.
- Additional directives were added to support multiple thread teams and simd parallelism.
- OpenMP continues to improve upon its support for offloading to accelerators.
OPENMP’S NEW MISSION STATEMENT

"Standardize directive-based multi-language high-level parallelism that is performant, productive and portable."
openmp target example: saxpy

saxpy in c

```c
void saxpy(int n,
    float a,
    float *x,
    float *restrict y)
{
    #pragma omp target teams \
    distribute parallel for
    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

```fortran
subroutine saxpy(n, a, x, y)
    real :: x(n), y(n), a
    integer :: n, i
    !$omp target teams &
    !$omp& distribute parallel do
    do i=1,n
        y(i) = a*x(i)+y(i)
    enddo
    !$omp end target teams &
    !$omp& distribute parallel do
    end subroutine saxpy

...!
```

// Perform SAXPY on 1M elements
saxpy(2**20, 2.0, x, y)
NEW FEATURES IN OPENACC 2.0
OPENACC **ROUTINE** DIRECTIVE

The routine directive specifies that the compiler should generate a device copy of the function/subroutine in addition to the host copy.

Clauses:

- **gang/worker/vector/seq**
  - Specifies the level of parallelism contained in the routine.
- **bind**
  - Specifies an optional name for the routine, also supplied at call-site
- **no_host**
  - The routine will only be used on the device
- **device_type**
  - Specialize this routine for a particular device type.
OPENACC ROUTINE: BLACK SCHOLES

// Polynomial approximation of cumulative normal distribution function

#pragma acc routine seq
real CND(real d)
{
    const real A1 = (real)0.31938153;
    const real A2 = (real)-0.356563782;
    const real A3 = (real)1.781477937;
    const real A4 = (real)-1.821255978;
    const real A5 = (real)1.330274429;
    const real RSQRT2PI = (real)0.39894228040143267793994605993438;

    real K = (real)1.0 / ((real)1.0 + (real)0.2316419 * FABS(d));

    real cnd = RSQRT2PI * EXP(- (real)0.5 * d * d) * 
               (K * (A1 + K * (A2 + K * (A3 + K * (A4 + K * A5)))));

    if(d > 0)
        cnd = (real)1.0 - cnd;

    return cnd;
}
UNSTRUCTURED DATA REGIONS

OpenACC 2.0 provides a means for beginning and ending a data region in different program scopes.

double a[100];

#pragma acc data copy(a)
{
   // OpenACC code
}

double a[100];

#pragma acc enter data copyin(a)
// OpenACC code
#pragma acc exit data copyout(a)
Unstructured Data Regions enable OpenACC to be used in C++ classes.

Unstructured data regions can be used whenever data is allocated and initialized in a different scope than where it is freed.

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

    private:
        double* v;
        int len;
};
OPENACC **ATOMIC** DIRECTIVE

Ensures a variable is accessed atomically, preventing race conditions and inconsistent results.

```c
#pragma acc parallel loop
for(int i=0; i<N; i++)
{
  if ( x[i] > 0 )
  {
    #pragma acc atomic capture
    {
      cnt++;
    }
  }
}
```

The atomic construct may **read**, **write**, **update**, or **capture** variables in a section of code.

**cnt** can only be accessed by 1 thread at a time.
MULTIPLE DEVICE TYPES

```c
#pragma acc parallel loop \
#ifdef(USE_NVIDIA)
    vector_length(256)
#else defined(USE_AMD)
    vector_length(512)
#endif
for (int i=0; i<n; ++i) {
    y[i] = a*x[i] + y[i];
}
```

```c
#pragma acc parallel loop \
device_type(nvidia) vector_length(256)\ 
device_type(radeon) vector_length(512)
for (int i=0; i<n; ++i) {
    y[i] = a*x[i] + y[i];
}
```
OPENACC 2.0 - HIGHLIGHTS

- Procedure calls, separate compilation (no more routine inlining)
- Nested parallelism (support for Dynamic Parallelism)
- Loop tile clause (multi-dimensional gangs)
- Data management features and global data (device resident globals)
- Device-specific tuning (improved portability)
- Asynchronous behavior additions (improved control flow)
- New API routines (improved C/C++ data API and portability)
- New atomic construct (parallel atomics)
- New default(none) data clause (compile-time debugging)
**OPENMP 4.0 TARGET DATA CONSTRUCT**

*Maps* host data to device data (no execution)
- Useful for mapping data that will be used by multiple TARGETs
- Equivalent to an OpenACC data region

“*Creates a device data environment for the extent of the region.*”

```c
#pragma omp target data
map(to: v1[0:N], v2[:N])
map(from: p[0:N])
{
  #pragma omp target ...
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
    p[i] = v1[i] * v2[i];
}
```

v1, v2, and p are reused
OPENMP 4.0 TARGET CONSTRUCT

Maps host data to device data
And serially executes code on the target device.

- Programmer must use OMP PARALLEL to begin parallel execution.

```
#pragma omp target
#pragma omp parallel for
for (i=0; i<N; i++)
p[i] = v1[i] * v2[i];
```

“Creates a device data environment for the extent of the region and executes on the same device.”
OPENMP 4.0 TEAMS/DISTRIBUTE

- Creates a league of teams and the master thread of each team executes the region.
  - Synchronization not possible between teams
  - Must be tightly nested within a target construct

- The programmer must distribute a loop across these teams.

```c
#pragma omp target teams 
    map(to: B[0:N], C[0:N])
#pragma omp distribute parallel for 
    reduction(+:sum)

for (i=0; i<N; i++)

    sum += B[i] * C[i];
```
OPENMP 4.0 TEAMS DISTRIBUTE PARALLEL FOR

#pragma omp target teams
#pragma omp distribute parallel for

#pragma omp parallel for reduction(+:sum)
for (i=0; i<N; i++)
    sum += B[i] * C[i];

#pragma omp parallel for reduction(+:sum)
for (i=0; i<N; i++)
    sum += B[i] * C[i];

#pragma omp parallel for reduction(+:sum)
for (i=0; i<N; i++)
    sum += B[i] * C[i];

#pragma omp parallel for reduction(+:sum)
for (i=0; i<N; i++)
    sum += B[i] * C[i];
OPENMP 4.0 HIGHLIGHTS

- Support for accelerators with the target directives
- SIMD directive for portable vectorization
- Task dependencies
- User defined reductions
- Thread affinity support
- Error handling
- Fortran 2003 support
- Improved atomics
NEXT STEPS

- OpenACC Website: http://openacc.org
- OpenMP Website: http://openmp.org
- NVIDIA Parallel Forall Blog & CUDACasts

- Go back and watch these GTC sessions:
  - S4167 - Introduction to Accelerated Computing Using Directives
  - S4200 - Advanced Accelerated Computing Using Directives

- Wednesday 4PM, LL20C - S4514 - Panel on Compiler Directives for Accelerated Computing