RWTH AACHEN UNIVERSITY

OpenMP for Accelerators

as of OpenMP 4.0 RC2, released in 03/2013

Christian Terboven <terboven@rz.rwth-aachen.de>

14.03.2013 / Aachen, Germany

Stand: 13.03.2013

Version 2.3

History

- De-facto standard for Shared-Memory Parallelization.

- 1997: OpenMP 1.0 for FORTRAN
- 1998: OpenMP 1.0 for C and C++
- 1999: OpenMP 1.1 for FORTRAN (errata)

- 2000: OpenMP 2.0 for FORTRAN
- 2002: OpenMP 2.0 for C and C++
- 2005: OpenMP 2.5 now includes both programming languages.

- 05/2008: OpenMP 3.0 release
- 07/2011: OpenMP 3.1 release

- 11/2012: OpenMP 4.0 RC1+TR, 03/2012: RC2

OpenMP™

http://www.OpenMP.org

RWTH Aachen University is a member of the OpenMP Architecture Review Board (ARB) since 2006.

Agenda

- What is an Accelerator in OpenMP?
- Execution Model and Data Model
- Target Construct and Accelerator-specific Constructs
- Example: SAXPY
- Outlook: Asynchronicity

Some content on these slides has been developed by James Beyer (Cray) and Eric Stotzer (TI), the leaders of the OpenMP for Accelerators subcommittee.

What is an Accelerator in OpenMP?

What kind of devices shall be supported?

- **In how differs an accelerator from just another core?**
  - different functionality, i.e. optimized for something special
  - different (possibly limited) instruction set

  → heterogeneous device

- **Assumptions used as design goals for OpenMP 4.0:**
  - every accelerator device is attached to one host device
  - it is probably heterogeneous
  - it may not be programmable in the same language as the host, or it may not implement all operations available on the host
  - it may or may not share memory with the host device
  - some accelerators are specialized for loop nests

Execution Model and Data Model

Execution Model

- **Host-centric: the execution of an OpenMP program starts on the *host device* and it may offload *target regions* to *target devices***
  - In principle, a target region also begins as a single thread of execution: when a target construct is encountered, the target region is executed by the implicit device thread and the encountering thread/task [on the host] waits at the construct until the execution of the region completes
- **If a target device is not present, or not supported, or not available, the target region is executed by the host device**
- **If a construct creates a *data environment*, the data environment is created at the time the construct is encountered**

Data Model

- **When an OpenMP program begins, each device has an initial *device data environment***

- **Directives accepting data-mapping attribute clauses determine how an *original* variable is mapped to a *corresponding* variable in a device data environment**
  - original: the variable on the host
  - corresponding: the variable on the device
  - the corresponding variable in the device data environment may share storage with the original variable (danger of data races)

- **If a corresponding variable is present in the enclosing device data environment, the new device data environment inherits the corresponding variable from the enclosing device**

Example: Execution and Data Model

- **Environment Variable OMP_DEFAULT_DEVICE=<int>: sets the device number to use in target constructs**

```
double B[N] = ...; // some initialization
#pragma omp target device(0) map(tofrom:B)
#pragma omp parallel for
for (i=0; i<N; i++)
   B[i] += sin(B[i]);
```

- map variable B to device, then execute parallel region on device, works probably pretty well on Intel Xeon Phi

```
double B[N] = ...; // some initialization
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
#pragma omp distribute
for (i=0; i<N; i += num_blocks)
   #pragma omp parallel for
   for (b = i; b < i+num_blocks; b++)
      B[b] += sin(B[b]);
```

- same as above, but code probably better optimized for NVIDIA GPGPUs

Comparing OpenMP with OpenACC

RWTH AACHEN UNIVERSITY

- **OpenMP 4.0 – for Intel Xeon Phi:**

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp parallel for
for (i=0; i<N; i++)
   B[i] += sin(B[i]);
```

- **OpenMP 4.0 – for NVIDIA GPGPU:**

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
#pragma omp distribute
for (i=0; i<N; i += num_blocks)
   #pragma omp parallel for
   for (b = i; b < i+num_blocks; b++)
      B[b] += sin(B[b]);
```

- **OpenACC – for NVIDIA GPGPU:**

```
#pragma acc parallel copy(B[0:N]) num_gangs(numblocks)\
            vector_length(bsize)
#pragma acc loop gang vector
  for (i=0; i<N; ++i) {
    B[i] += sin(B[i]);
  }
```

Comparing OpenMP with OpenACC

- **OpenMP 4.0 – for Intel Xeon Phi:**

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp parallel for
for (i=0; i<N; i++)
   B[i] += sin(B[i]);
```

- **OpenMP 4.0 – for NVIDIA GPGPU:**

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
#pragma omp distribute
for (i=0; i<N; i += num_blocks)
   #pragma omp parallel for
   for (b = i; b < i+num_blocks; b++)
      B[b] += sin(B[b]);
```

- **OpenACC for NVIDIA GPGPU:**

```
#pragma acc parallel copy(B[0:N]) num_gangs(numblocks)\
            vector_length(bsize)
#pragma acc loop gang vector
  for (i=0; i<N; ++i) {
    B[i] += sin(B[i]);
  }
```

Current work in the OpenMP Language Committee targeting OpenMP 4.0:

Combined directive

#pragma omp teams distribute parallel for

Target Construct and Accelerator-specific Constructs

target data construct

- **Creates a device data environment for the extent of the region**
  - when a target data construct is encountered, a new device data environment is created, and the encountering task executes the target data region
  - when an if clause is present and the if-expression evaluates to false, the device is the host

- **C/C++:**

The syntax of the **target data** construct is as follows:

#pragma omp target data *[clause[[,] clause],...] new-line*
structured-block

where *clause* is one of the following:

device(*integer-expression* **)**
map(*[map-type* **:** *] list* **)**
if(*scalar-expression* **)**

map clause

- **Map a variable from the current task's data environment to the device data environment associated with the construct**
  - the list items that appear in a map clause may include array sections
  - ***alloc***-type: each new corresponding list item has an undefined initial value
  - ***to***-type: each new corresponding list item is initialized with the original lit item's value
  - ***from***-type: declares that on exit from the region the corresponding list item's value is assigned to the original list item
  - ***tofrom***-type: the default, combination of to and from

- **C/C++:**

The syntax of the **map** clause is as follows:

map(*[map-type* **:** *] list* **)**

target construct

- **Creates a device data environment and execute the construct on the same device**
  - superset of the target data constructs - in addition, the target construct specifies that the region is executed by a device and the encountering task waits for the device to complete the target region

- **C/C++:**

The syntax of the **target** construct is as follows:

#pragma omp target *[clause[[,] clause],...] new-line*
structured-block

where *clause* is one of the following:

device(*integer-expression* **)**
map(*[map-type* **:** *] list* **)**
if(*scalar-expression* **)**

Example: Target Construct

```
#pragma omp target device(0)
#pragma omp parallel for
{
  for (i=0; i<N; i++) ...
}
```

```
#pragma omp target
#pragma omp teams num_teams(8) num_threads(4)
#pragma omp distribute
  for ( k = 0; k < NUM_K; k++ )
  {
    #pragma omp parallel for
    for ( j = 0; j < NUM_J; j++ )
    {
       ...
    }
  }
```

target update construct

- **Makes the corresponding list items in the device data environment consistent with their original list items, according to the specified motion clauses**

- **C/C++:**

The syntax of the **target update** construct is as follows:

#pragma omp target update *motion-clause[, clause[[,] clause],...] new-line*

where *motion-clause* is one of the following:

to(*list* **)**
from(*list* **)**

and where *clause* is one of the following:

device(*integer-expression* **)**
if(*scalar-expression* **)**

declare target directive

- **Specifies that [static] variables, functions (C, C++ and Fortran) and subroutines (Fortran) are mapped to a device**
  - if a list item is a function or subroutine then a device-specific version of the routines is created that can be called from a target region
  - if a list item is a variable then the original variable is mapped to a corresponding variable in the initial device data environment for all devices (if the variable is initialized it is mapped with the same value)
  - all declarations and definitions for a function must have a declare target directive

- **C/C++:**

The syntax of the **declare target** directive is as follows:

#pragma omp declare target *new-line*
declarations-definition-seq
#pragma omp end declare target *new-line*

teams construct (1/2)

- **Creates a league of thread teams where the master thread of each team executes the region**
  - the number of teams is determined by the num_teams clause, the number of threads in each team is determined by the num_threads clause, within a team region team numbers uniquely identify each team (omp_get_team_num())
  - once created, the number of teams remeinas constant for the duration of the teams region
- **The teams region is executed by the master thread of each team**
- **The threads other than the master thread to not begin execution until the master thread encounteres a parallel region**
- **Only the following constructs can be closely nested in the team region: distribute, parallel, parallel loop/for, parallel sections and parallel workshare**

teams construct (2/2)

- A teams construct must be contained within a target construct, which must not contain any statements or directives outside of the teams construct
- After the teams have completed execution of the teams region, the encountering thread resumes execution of the enclosing target region

- C/C++:

The syntax of the **teams** construct is as follows

#pragma omp teams *[clause[[,] clause],...] new-line*
structured-block

where *clause* is one of the following:

num_teams(*integer-expression* **)**
num_threads(*integer-expression* **)**
default(shared | none)
private(*list* **)**
firstprivate(*list* **)**
shared(*list* **)**
reduction(*operator* **:** *list* **)**

distribute construct

- **Specifies that the iteration of one or more loops will be executed by the thread teams, the iterations are distributed across the master threads of all teams**
  - there is no implicit barrier at the end of a distribute construct
  - a distribute construct must be closely nested in a teams region

- **C/C++:**

The syntax of the **distribute** construct is as follows:

#pragma omp distribute *[clause[[,] clause],...] new-line*
for-loops

Where *clause* is one of the following:

private(*list* **)**
firstprivate(*list* **)**
collapse(*n* **)**
dist_schedule(*kind[, chunk_size]* **)**

All associated for-loops must have the canonical form described in Section 2.5.

Example: SAXPY

SAXPY: Serial (Host)

```
int main(int argc, const char* argv[]) {
  int n = 10240; float a = 2.0f; float b = 3.0f;
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Initialize x, y

  // Run SAXPY TWICE

  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  for (int i = 0; i < n; ++i){
        y[i] = b*x[i] + y[i];
  }

  free(x); free(y); return 0;
}
```

SAXPY: OpenACC v2 (NVIDIA GPGPU)

```
int main(int argc, const char* argv[]) {
  int n = 10240; float a = 2.0f; float b = 3.0f;
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Initialize x, y

  // Run SAXPY TWICE
#pragma acc data copyin(x[0:n])
{

#pragma acc parallel copy(y[0:n])
#pragma acc loop
  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

#pragma acc parallel copy(y[0:n])
#pragma acc loop
  for (int i = 0; i < n; ++i){
        y[i] = b*x[i] + y[i];
  }
}
  free(x); free(y); return 0;
}
```

SAXPY: OpenMP 4.0 (NVIDIA GPGPU)

```
int main(int argc, const char* argv[]) {
  int n = 10240; float a = 2.0f; float b = 3.0f;
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Initialize x, y

  // Run SAXPY TWICE
#pragma omp target data map(to:x)
{
#pragma omp target map(tofrom:y)
#pragma omp teams
#pragma omp distribute
#pragma omp parallel for
for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
#pragma omp target map(tofrom:y)
#pragma omp distribute
#pragma omp distribute
#pragma omp parallel for
  for (int i = 0; i < n; ++i){
        y[i] = b*x[i] + y[i];
  }
}
  free(x); free(y); return 0;
```

Outlook: Asynchronicity

Example: Asynchronous Offload via Tasks

- **For asynchronous execution use the task construct and task dependencies:**

```
#pragma omp target data map(alloc:Z)
{
   #pragma omp parallel for
   for (c = 0; c < nchunks; c += chunksz)
   {
      #pragma omp task dep(out:c)
      #pragma omp target update map(to: Z[c:chunksz])

      #pragma omp task dep(in:c)
      #pragma omp target
      #pragma omp parallel for
      for (i = c; i < c + chunksz; i++)
         Z[i] = f(Z[i]);
   }
}
```

The End

Thank you for your attention.