OpenMP accelerator model

James Beyer and Eric Stotzer

OpenMP BOF SC'12

Nov 2012

OpenMP Design Challenges

- A model that is portable to different accelerator (device) ISAs
  - How to manage data motion
    - Host and device share memory

      OR
    - device and host do not share memory
  - How to partition code among host and device(s)
  - How to express parallelism and asynchronous execution
- A model that *fits* in OpenMP
- A model that is *productive* and offers *performance*

Overview

- Host-centric model with a *host device* and multiple *target devices* of the same type
  - **device** A logical execution engine with local storage.
  - **device data environment** A data environment associated with a **target data** or **target** region.
- target constructs control how data and code is offloaded to a device.
- Data is *mapped* from a host data environment to a device data environment.

Summary

- New directives
  - target
  - target data
  - target update
  - target mirror
  - target linkable
- New runtime functions
  - omp_get_device_num
  - omp_set_device_num
- New environment variable
  - OMP_DEVICE_NUM

target

Create a device data environment and execute the construct on the same device.

C/C++
#pragma omp target *[clause[[,] clause],...] new-line* *parallel-loop-construct* | *parallel-sections-construct*
C/C++
Fortran
!$omp target** *[clause[[,] clause],...]* *parallel-loop-construct* \| *parallel-sections-construct* **!$omp end target
Fortran

Clauses

device(*integer-expression* **)**
map(*list* **)**
mapto(*list* **)**
mapfrom(*list* **)**
scratch(*list* **)**
num_threads(*list* **)**
if(*scalar-expression* **)**

```
sum = 0;
#pragma omp target device(acc0) map(B,C)
#pragma omp parallel for reduction(+:sum)
for (i=0; i<N; i++)
    sum += B[i] * C[i]
```

target data

Create a device data environment for the extent of the region.

C/C++
#pragma omp target data *[clause[[,] clause],...] new-line* *structured-block*
C/C++
Fortran
!$omp target data *[clause[[,] clause],...]* *structured-block* **!$omp end target data**

Clauses

device(*integer-expression* **)**
map(*list* **)**
mapto(*list* **)**
mapfrom(*list* **)**
scratch(*list* **)**
if(*scalar-expression* **)**

```
void gramSchmidt(restrict float Q[][COLS], const int rows,
const int cols)
{
#pragma omp target data map(Q[0:rows][0:cols])
  for(int k=0; k < cols; k++) {
    double tmp = 0.;

#pragma omp target
#pragma omp parallel for reduction(+:tmp)
for(int i=0; i < rows; i++) tmp +=  (Q[i][k] * Q[i][k]);
    tmp = sqrt(tmp);

#pragma omp target
#pragma omp parallel for
for(int i=0; i < rows; i++) Q[i][k] /= tmp;

…
}}}
```

target update

Update(to) a variable from the data environment of the current task to the enclosing device data environment, or update(from) a variable from the enclosing device data environment to the data environment of the current task.

C/C++
#pragma omp target update *[clause[[,] clause],...] new-line*
C/C++
!$omp target update *[cl*

Clauses

device(*integer-expressio*
mapto(*list* **)**
mapfrom(*list* **)**
if(*scalar-expression* **)**

```
!$omp target data map(grad,recv_w,recv_e,send_e,send_w,recv_n,recv_s,send_n,send_s)
!$omp target
!$omp parallel do
      do k=-1,lz
        do j=-1,local_ly
          send_e(j,k) = grad(local_lx-1,j        ,k)
          send_w(j,k) = grad(0         ,j        ,k)
        end do
      end do
!$omp end parallel do
!$omp end target
!$omp target update mapfrom(send_e,send_w)
      call mpi_irecv(recv_w, bufsize(2),mpi_double_precision,w_id, tag(25),&
                     mpi_comm_world,irequest_in(25),ierr)
          o  o  o
      call mpi_isend(send_w, bufsize(2),mpi_double_precision,w_id, tag(26),&
                     mpi_comm_world,irequest_out(26),ierr)
      call mpi_waitall(2,irequest_in(25),istatus_req,ierr)
      call mpi_waitall(2,irequest_out(25),istatus_req,ierr)
!$omp target update mapto(recv_e,recv_w)
!$omp target
!$omp parallel do
      do k=-1,lz
        do j=-1,local_ly
          grad(local_lx  ,j        ,k) = recv_e(j,k)
          grad(-1        ,j        ,k) = recv_w(j,k)
```

declare target

The **declare target** construct can be applied to a function (C, C++ and Fortran) or a subroutine (Fortran) to enable the creation of a device specific version that can be called from a target region.

C/C++
#pragma omp declare target *new-line* *function-definition-or-declaration*
C/C++
Fortran
!$omp declare target *(subroutine-or-function-name)*
Fortran

```
#pragma omp target declare
void P(restrict float Q[][COLS], const int i, const int k)
{ return Q[i][k] * Q[i][k]; }

#pragma omp target data map(Q[0:rows][0:cols])
#pragma omp parallel for reduction(+:tmp)
for(int i=0; i < rows; i++) tmp += P(Q,i,k);
```

declare target mirror

Map a global variable to a device for the duration of the program

C/C++
#pragma omp declare target mirror(*list* **)** *new-line*
C/C++

Fortran
!$omp declare target mirror(*list* **)**
Fortran

```
#pragma omp target declare mirror(Q)
float Q[ROWS][COLS];

#pragma omp target declare
void P(const int i, const int k)
{ return Q[i][k] * Q[i][k]; }

#pragma omp target data
#pragma omp parallel for reduction(+:tmp)
for(int i=0; i < rows; i++) tmp += P(i,k);
```

OpenMP declare target linkable

Assert that the user has mapped a global variable to a device

C/C++
#pragma omp declare target linkable(list) *new-line*
C/C++
Fortran
!$omp declare target linkable(*list*)
Fortran

```
extern int Y;
#pragma omp declare target linkable(Y)

#pragms omp declare target
void F(void);

#pragma omp target map(Y)
#pragma omp parallel sections
{ F() };

void F(void) { Use Y; }
```

Asynchronous execution

Use the task construct and upcoming task dependencies

```
#pragma omp target data scratch(Z)
{
   #pragma parallel section
    for (C=0; C<NCHUNKS; C+=CHUNKSZ)
    {
        #pragma omp task source(C)
        #pragma omp target update mapto(Z[C:CHUNKSZ])

        #pragma omp task sink(C)
        #pragma omp target
        #pragma omp parallel for
        for (i=C; i<C+CHUNKSZ; i++) Z[i] = F(Z[i]);
    }
}
```

Why only a TR?

- ✓ Data environment
  - ✓ Compute region
  - ✓ Multiple compute regions
- ✓ Data transfers
  - ✓ Map, mapto, mapfrom, scratch
  - ✓ Update
- ✓ Subroutines
- ✓ Linking support
- Execution model
  - Works for many devices
  - Does not work on all devices

What is missing from TR?

- MIC/PHI – nothing
- Convey – nothing
- DSPs – nothing
- APUs -- nothing
- GPUs – several issues
- ???

What is missing for GPUs?

- OpenMP has a very rich set of synchronizations
- GPUs almost no synchronization capability at some levels
  - Example architecture
  - Nvidia TB
    - No support for barriers
    - No support for locks
    - Atomics cannot be used to build locks or barriers
  - Nvidia warp
    - Synchronization is back!

What is next?

- Do we restrict functionality for GPU's or do we add new constructs?

- Goals
  - Portable
  - Productive
  - Performance

- Need to define an execution model that works everywhere!