Using OpenMP 4.5 Target Offload for Programming Heterogeneous Systems

Mar 20, 2019
NASA Advanced Supercomputing Division
Outline

• Introduction
  - Concepts of Programming Heterogeneous Architectures

• OpenMP Heterogeneity Support Basics
  - Off-loading Work and Data to the GPU
  - Expressing Parallelism and Data Locality

• Using OpenMP 4.5 on Pleiades GPU Nodes

• Learning by Example
  - Laplace Kernel

• More OpenMP 4.5 Constructs and Clauses

• References
Heterogeneous Systems

- A general purpose processor connected to an accelerator device
- An example is an Intel Xeon processor connected to a Nvidia GPU

**CPU**
- Fast clock (2.4-2.9 GHz on Pleiades)
- Multiple cores (16-40 on Pleiades)
- Complex cores
  - Large caches, complex branch prediction, OOO execution, multi-threading
- Parallelism
  - Deep pipelines, multiple cores, vector units, SIMD execution

**Device**
- Slow clock (0.8-1.0 GHz)
- Thousands of cores
  - 2880 SP cores on Pleiades
- Lightweight cores:
  - Small caches, little branch prediction, in-order execution, multi-threading
- Parallelism: Theoretically enormous!
  - In practice limited; SIMT execution

Image courtesy of Nvidia
Programming Heterogeneous Systems

• Necessary steps to be taken
  - Identification and offloading of compute kernels from host to device
  - Expressing parallelism within the kernel
  - Manage data transfer between CPU and Device

• Execution flow
  1. Data copy from main to device memory
  2. CPU initiates kernel for execution on the device
  3. Device executes the kernel using device memory
  4. Data copy from device to main memory

Example: Device is a GPU
Why Use OpenMP?

- **Methods to program heterogeneous systems:**
  - **Vendor specific** library routines
    - Examples: cuBlas, cufftw, …
  - **Vendor specific** frameworks
    - Example: Cuda
  - Public domain frameworks
    - Example: OpenCL
  - Compiler directive based API
    - OpenMP

- **OpenMP:**
  - Compiler directives
  - Runtime library routines
  - Environment variables

- Well established since 1997

- Heterogeneous programming since OpenMP 4.0
  - Seamlessly integrates into existing OpenMP code
  - Supported by many compilers: gcc, Intel icc, IBM xl, Cray cc

- Low programming effort, high performance, **non-portable**
- Medium programming effort with acceptable performance, portable
- High programming effort, high performance, suitable for any kernel, **non-portable**
- High programming effort, high performance, suitable for any kernel, portable
OpenMP Directive Syntax

• Compiler Directive
  - Programmer inserted hint/command for the compiler

• Directive Syntax
  - Fortran
    o Mostly paired with a matching end directive surrounding a structured code block

    ```
    !$omp directive [clause [,] [clause] …]
    code
    !$omp end directive
    ```

  - C
    o No end directive needed as the structured block is bracketed

    ```
    #pragma omp directive [clause [,] [clause] …]
    {
      code
    }
    ```
OpenMP Heterogeneity Basic Support

```c
#pragma omp target or
!$omp target/end target
```
- A device data environment is created for the marked region
- The marked code region is mapped to the device and executed.

```c
#pragma omp target teams or
!$omp teams/end teams
```
- A **league** of thread teams is created
- The master thread of each team executes the code region

```c
#pragma omp target teams distribute
```
- Worksharing construct: Share work across the teams

```c
#pragma omp parallel for or !$omp parallel do
```
- Worksharing across threads within a team

```c
#pragma omp simd
```
- Worksharing across vector length

```c
#pragma omp target map(map-type: list)
```
- Map a variable to/from the device data environment

We have seen those before....
Stencil Kernel in OpenMP 4.5

```c
#pragma omp target teams distribute num_teams(4)
for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++) {
        error = fmax( error, fabs(Anew[j][i] - A[j][i]) );
    }
}
```

- League of teams is started via `target teams`
- All teams execute the target region

Image courtesy of Nvidia
Stencil Kernel in OpenMP 4.5

```
#pragma omp target teams distribute num_teams(4) parallel for thread_limit(4)
for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
        error = fmax( error, fabs(Anew[j][i] - A[j][i]));
    }
}
```

For more details check out the presentation by Jeff Larkin, Nvidia:
Structured Data Management

• The device data region:
  - A region of the program within which data is accessible to the device
  - It can be explicitly defined to reduce data copies
  - The `target data` construct is used to mark such regions

```c
#pragma omp target data map(map-type: list)
```

• Example map types:
  - `to (list)`
    - Allocates memory on the device and copies data in when entering the region, the values are not copied back
  - `from (list)`
    - Allocates memory on the device and copies the data to the host when exiting the region
  - `alloc (list)`
    - Allocates memory on the device
    - If the data is already present on the device a reference counter is incremented
Compiling and Running on Pleiades

• Pleiades GPU nodes:
  - 64 Sandy Bridge nodes with one Tesla K40 GPU

• Compilation:
  - Intel icc/ifort does not generate code for Nvidia GPU execution 😞
  - PGI pgcc/pgf90 does not support the OpenMP target construct 😞
  - Experimental gcc 8.1 with GPU support is available on Pleiades 😊

```bash
module purge
module load cuda
module load /home1/gjost/public/modules/gcc8.1-module
gcc --version
gcc or gfortran -fopenmp -foffload="-lm" test.c or test.f90
```

• Submit to GPU node
  ```bash
  qsub -l select=1:ncpus=16:model=san_gpu -q k40
  ```

• Run the executable
  ```bash
  ./a.out
  ```
Example: 2D Laplace Solver

```c
#pragma omp target data map(to:Anew) map(A)
while ( error > tol && iter < iter_max ) {
    error = 0.0;

#pragma omp target teams distribute parallel for reduction(max:error) map(error)
    for( int j = 1; j < n-1; j++ ){
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                              + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));}
    } // end of while loop

if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);
iter++;} // end of while loop
```

**total: 167.7766 seconds on Pleiades**
Example: 2D Laplace Solver Separated

```c
#pragma omp target data map(to:Anew) map(A)
while ( error > tol && iter < iter_max ) {
    error = 0.0;

#pragma omp target teams distribute reduction(max:error) map(error)
    for( int j = 1; j < n-1; j++ ){
        #pragma omp parallel for reduction(max:error)
            for( int i = 1; i < m-1; i++ ){
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1] 
                                  + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
    #pragma omp target teams distribute
        for( int j = 1; j < n-1; j++ ){
    #pragma omp parallel for
            for( int i = 1; i < m-1; i++ ){
                A[j][i] = Anew[j][i];
            }
        }

    if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);
    iter++;
}

total: 155.659801 seconds on Pleiades
```

- Other possibilities, eg use collapse clause
- Depends on problem sizes and system
Example: 2D Laplace Solver in Fortran

```fortran
!$omp target data map(to:Anew) map(A)
   do while ( error .gt. tol .and. iter .lt. iter_max )
      error=0.0_fp_kind
   !$omp target teams distribute parallel do reduction(max:error) map(error)
      do j=1,m-2
         do i=1,n-2
            Anew(i,j) = 0.25_fp_kind * ( A(i+1,j) + A(i-1,j) + &
                                      A(i ,j-1) + A(i ,j+1) )
            error = max( error, abs(Anew(i,j)-A(i,j)) )
         end do
      end do
   !$omp end target teams distribute parallel do
   if(mod(iter,100).eq.0 ) write(*,'(i5,f10.6)') iter, error
   iter = iter + 1
   !$omp target teams distribute parallel do
      do j=1,m-2
         do i=1,n-2
            A(i,j) = Anew(i,j)
         end do
      end do
   !$omp end target teams distribute parallel do
   !$omp end target data
```
Performance Challenges

• Function calls in inner loops:
  - Challenging for the compiler’s dependence analysis
    o Possibly use !$omp declare target to compile routine for the device (p. 20)
    o manually in-line

• IF-Branches:
  - Can get expensive if threads take different execution paths
    o Move your branches up to a level in your code where all threads go down the same code branch
    o Avoid branches in inner loops

• Structures with complex and/or dynamic components:
  - Might prevent stride 1 memory access
  - Example: NPB FT uses complex data; performance increase via manual linearization
  - Unstructured data management for structures with dynamic components (next slide)

• Loop strides/memory layout
  - Inner loop should be long and move along the fastest dimension
Unstructured Data Management:

Enter/Exit Data Constructs

- Real life applications, just like real life, are not always a nicely structured sequence of code regions
  - Example: C++ Structures or Fortran user defined types with dynamic arrays

- Unstructured data directives
  - `!$omp target enter data map(map_type:list)`
    - Allocate memory on the device for the remainder of the program or until explicitly deleted
    - Possible map types are `to`, `alloc`
  - `!$omp target exit data map(map_type:list)`
    - Deallocation of the memory on the device
    - Possible map types are `from`, `release`, or `delete`

- Multiple `enter/exit data` constructs, branched across different function calls are allowed
module matrix_mod
  implicit none
  type matrix
    integer :: num_rows
    integer :: nnz
    integer, pointer :: rowOffsets(:)
    integer, pointer :: cols(:)
    real(8), pointer :: coefs(:)
  end type matrix
public :: allocate_3d
...
end module

subroutine allocate_3d (a, n)
  implicit none
  type(matrix) :: a
  allocate(a%row_offsets(num_rows+1))
  allocate(a%cols(nnz))
  allocate(a%coefs(nnz))
  arrow_offsets => a%row_offsets
  acols => a%cols
  acoefs => a%coefs
!$omp target enter data map(to:a)
!$omp target enter data map(to:acols(1:nnz)...
end subroutine allocate_3d

subroutine free_3d(a)
  implicit none
  type(matrix) :: a
  arow_offsets => a%row_offsets
  acols => a%cols
  acoefs => a%coefs
!$omp exit data map(delete:acols(1:nnz),...)
!$omp exit data map(delete: a)
dekallocate(arowOffsets)
dekallocate(acols)
dekallocate(acoefs)
end subroutine
dekallocate

Call allocate_3d and free_3d as often as you need to, also across subroutine calls
Routines and global variables

```c
#$omp declare target
- Specifies that variables, functions and subroutines are
mapped to a device for the duration of the program

in LBM.cpp
```

```c
#pragma omp target teams distribute parallel do
for (int index1 = 0; index1 < a_numPts; ++index1)
{
    Real feq[NUMV];
    getFEQ(feq, index1);
}
```

```c
#pragma omp declare target
inline void getFEQ(Real* const __restrict a_feq, const int a_index1)
```

```c
#pragma omp end declare target
```

```c
```

in LBM.H
Synchronize host and device buffers

$!$omp target update clause
- Makes the variables on the device consistent with the original variables
- Possible motion clauses are to or from

```
#pragma omp declare target
void work_on_device_work(int* srcA, int* result);
#pragma omp end declare target

#pragma omp target data map(to:a[0:size)map(from:c[0:size])
{
    #pragma omp target
    {
        work_on_device_work(a, c);
    }
    /* Copy modified target data to the host */
    #pragma omp target update from(c[0:size])
    work_on_host(c);
    #pragma omp target
    {
        work_on_device_work(a, c);
    }
```
Device Pointers

- Pointer variable on the host containing device address
- Device memory routines
  - Support of allocation and management of pointers in the data environment of the target device

```c
u1 = (double*)omp_target_alloc(n3*n2*n1*sizeof(double), omp_get_default_device());
...

#pragma omp target map(tofrom: ou[0:n3*n2*n1]) is_device_ptr(u1)

#pragma omp teams distribute
for (i3 = 1; i3 < n3-1; i3++) {
    #pragma omp parallel for collapse(2)
    for (i2 = 1; i2 < n2-1; i2++) {
        for (i1 = 0; i1 < n1; i1++) {
            I3D(u1, n1, n2, i3, i2, i1) =
            I3D(ou, n1, n2, i3, i2-1, i1) +
            I3D(ou, n1, n2, i3, i2+1, i1) +
            I3D(ou, n1, n2, i3-1, i2, i1) +
            I3D(ou, n1, n2, i3+1, i2, i1); }}
}
```

Note:
#define I3D(array,n1,n2,i3,i2,i1) (array[(i3)*n2*n1 + (i2)*n1 + (i1)])
References

Using GPUs codes on Pleiades HECC KB

https://www.nas.nasa.gov/hecc/support_kb/using-gpu-nodes_298.html

Jeff Larkin Presentation


• OpenMP Home Page and Books

http://www.openmp.org/

http://www.openmp.org/tech/using-openmp-next-step/

https://mitpress.mit.edu/books/using-openmp
module mod_fib
contains
   subroutine fib(N)
      integer :: N
      !$omp declare target
      ! ...
   end subroutine
end module
program my_fib

Integer :: N=8
!$omp declare target
!$omp target
   call fib(N)
!$omp end target
end program

in a module

external

program my_fib

!$omp target if (N > 1000)
   call fib(N)
!$omp end target
end program
subroutine fib(N)
    integer :: N
    !$omp declare target
    print*, "hello from fib"
    ! ...
end subroutine

Question? Use the Webex chat facility to ask the Host
#pragma omp target map(a,b,c,d) nowait
{
  
  #pragma parallel for
  for (i=0; i<N; i++) {
    a[i] = b[i] * c + d;
  }
}
F(b); //Execute in parallel with target task

#pragma omp taskwait //wait for target task to finish
Example: 2D Laplace Solver Collapse

```c
#pragma omp target data map(to:Anew) map(A)
while ( error > tol && iter < iter_max ) {
    error = 0.0;

#pragma omp target teams distribute parallel for reduction(max:error)
    map(error) collapse (2)
    for ( int j = 1; j < n-1; j++ ){
        for( int i = 1; i < m-1; i++ ){
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                               + A[j-1][i] + A[j+1][i] );
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));}

#pragma omp target teams distribute parallel for collapse(2)
    for( int j = 1; j < n-1; j++ ){
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i]; }

    if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);
    iter++;
}
```

total: 167.67 seconds on Pleiades
### Get to know your GPU

```bash
PBS r313i1n2 51> module load cuda
PBS r313i1n2 52> nvidia-smi
Mon Mar 18 14:58:08 2019
```

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.81  Driver Version: 384.81                                   |
+-----------------------------------------------------------------------------+
| GPU_Name  Persistence-M| Bus-Id   | Disp.A | Volatile Uncorr. ECC | Fan | Temp | Perf | Pwr:Usage/Cap | Memory-Usage | GPU-Util | Compute_M. |
+-----------------------------------------------------------------------------+
| 0  Tesla K40m  Off  | 00000000:02:00.0 Off | 0 | N/A  25C  P0  67W / 235W | 0MiB / 11439MiB | 95%     | Default |
+-----------------------------------------------------------------------------+

**Processes:**

```
| GPU   | PID | Type | Process name  | Usage   |
+-------+-----+------|---------------+---------|
| No running processes found                                                 |
+-----------------------------------------------------------------------------+
```
### nvprof textual Profiles

**Sparse Matvec Better**

<table>
<thead>
<tr>
<th>Time(%)</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>97.02%</td>
<td>15.8332s</td>
<td>101</td>
<td>156.76ms</td>
<td>156.50ms</td>
<td>156.90ms</td>
<td>matvec_114_gpu</td>
</tr>
<tr>
<td>1.91%</td>
<td>311.27ms</td>
<td>302</td>
<td>1.0307ms</td>
<td>0.69617ms</td>
<td>1.0377ms</td>
<td>waxpby_60_gpu</td>
</tr>
<tr>
<td>0.89%</td>
<td>145.68ms</td>
<td>200</td>
<td>728.41us</td>
<td>620.77us</td>
<td>843.05us</td>
<td>dot_46_gpu</td>
</tr>
<tr>
<td>0.18%</td>
<td>29.020ms</td>
<td>200</td>
<td>145.10us</td>
<td>144.10us</td>
<td>146.27us</td>
<td>dot_46_gpu_red</td>
</tr>
<tr>
<td>0.00%</td>
<td>422.63us</td>
<td>200</td>
<td>2.1130us</td>
<td>2.0800us</td>
<td>2.7840us</td>
<td>[CUDA memcpy DtoH]</td>
</tr>
<tr>
<td>0.00%</td>
<td>186.50us</td>
<td>200</td>
<td>932ns</td>
<td>896ns</td>
<td>1.6320us</td>
<td>[CUDA memcpy HtoD]</td>
</tr>
</tbody>
</table>

**Sparse Matvec Best**

```bash	nvprof ./cg.exe
```

<table>
<thead>
<tr>
<th>Time(%)</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>87.90%</td>
<td>5.36086s</td>
<td>101</td>
<td>53.078ms</td>
<td>53.044ms</td>
<td>53.107ms</td>
<td>matvec_118_gpu</td>
</tr>
<tr>
<td>5.07%</td>
<td>309.17ms</td>
<td>302</td>
<td>1.0237ms</td>
<td>680.83us</td>
<td>1.0302ms</td>
<td>waxpby_64_gpu</td>
</tr>
<tr>
<td>4.17%</td>
<td>254.03ms</td>
<td>370</td>
<td>686.57us</td>
<td>896ns</td>
<td>1.7418ms</td>
<td>[CUDA memcpy HtoD]</td>
</tr>
<tr>
<td>2.37%</td>
<td>144.67ms</td>
<td>200</td>
<td>723.35us</td>
<td>622.12us</td>
<td>831.88us</td>
<td>dot_50_gpu</td>
</tr>
<tr>
<td>0.47%</td>
<td>28.874ms</td>
<td>200</td>
<td>144.37us</td>
<td>143.39us</td>
<td>145.63us</td>
<td>dot_50_gpu_red</td>
</tr>
<tr>
<td>0.01%</td>
<td>641.54us</td>
<td>2</td>
<td>320.77us</td>
<td>320.55us</td>
<td>320.99us</td>
<td>initialize_vector_26_gpu</td>
</tr>
<tr>
<td>0.01%</td>
<td>414.98us</td>
<td>200</td>
<td>2.0740us</td>
<td>2.0480us</td>
<td>2.8480us</td>
<td>[CUDA memcpy DtoH]</td>
</tr>
</tbody>
</table>
Unstructured Data Management

module matrix_mod
    implicit none
    type matrix
        integer :: num_rows
        integer :: nnz
        integer, pointer :: row_offsets(:)
        integer, pointer :: cols(:)
        real(8), pointer :: coefs(:)
    end type matrix
end public :: allocate_3d
....

subroutine allocate_3d(a, n)
    implicit none
    type(matrix) :: a
    allocate(a%rowOffsets(num_rows+1))
    allocate(a%cols(nnz))
    allocate(a%coefs(nnz))
    arrow_offsets => a%row_offsets
    acols => a%cols
    acoefs => a%coefs
    !$omp target enter data map(to: a)
    !$omp target enter data map(to : a%row_offsets,a%cols,a%coefs)...
end subroutine allocate_3d
end module matrix_mod

Call as often as you need to, also across subroutine calls