OpenMP 4.5 Target Pragma

Learning Objectives

- **Explain** OpenMP pragma target
- **IDENTIFY** which loops are parallelizable with this pragma
- **COMPARE** sequential to GPU accelerated execution times
- **Apply** pragma target to accelerate scientific code example
  - Poisson Equation

Expected Time of the Activity:

Module Approximate Timing

- Pragma Target Introduction clauses: 10 min
- Example: 15 min

Total time for the module: 25 minutes

#pragma omp target

- Introduced in OpenMP 4.5 Standard to Program Heterogeneous Systems
- Heterogeneous System: A general purpose CPU connected to an accelerator device

  Example: Intel CPU connected to a Nvidia GPU
- Programming Steps:
  - Identify the device to offload the computations to
  - Copy Data from Host CPU to Device Memory
  - CPU initializes execution of the code (loop or task function) on the device
  - Device executes the function
  - Data is copy back from the device to the host CPU

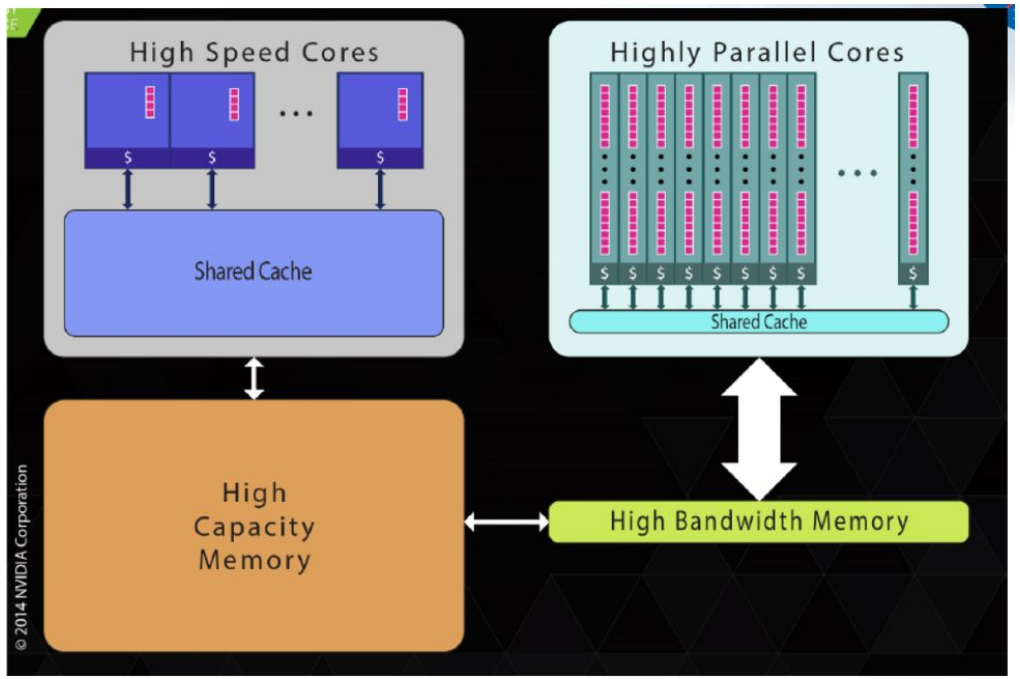
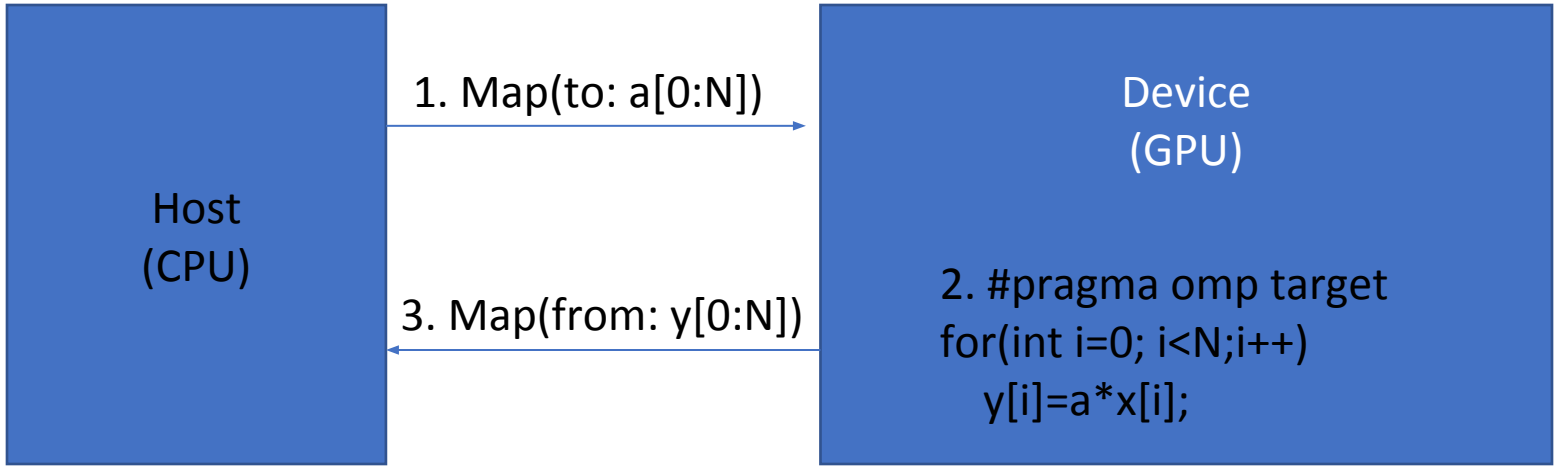


Image courtesy of Nvidia

#pragma omp target

- Creates a device data environment for the code region
- The marked code region is mapped to the device and executed



#pragma omp target clauses:

#pragma omp target teams

- A league of threads teams is created
- The master thread of each team executes the code region

#pragma omp target teams distribute

- Share work across the teams

#pragma omp target map(map-type:list)

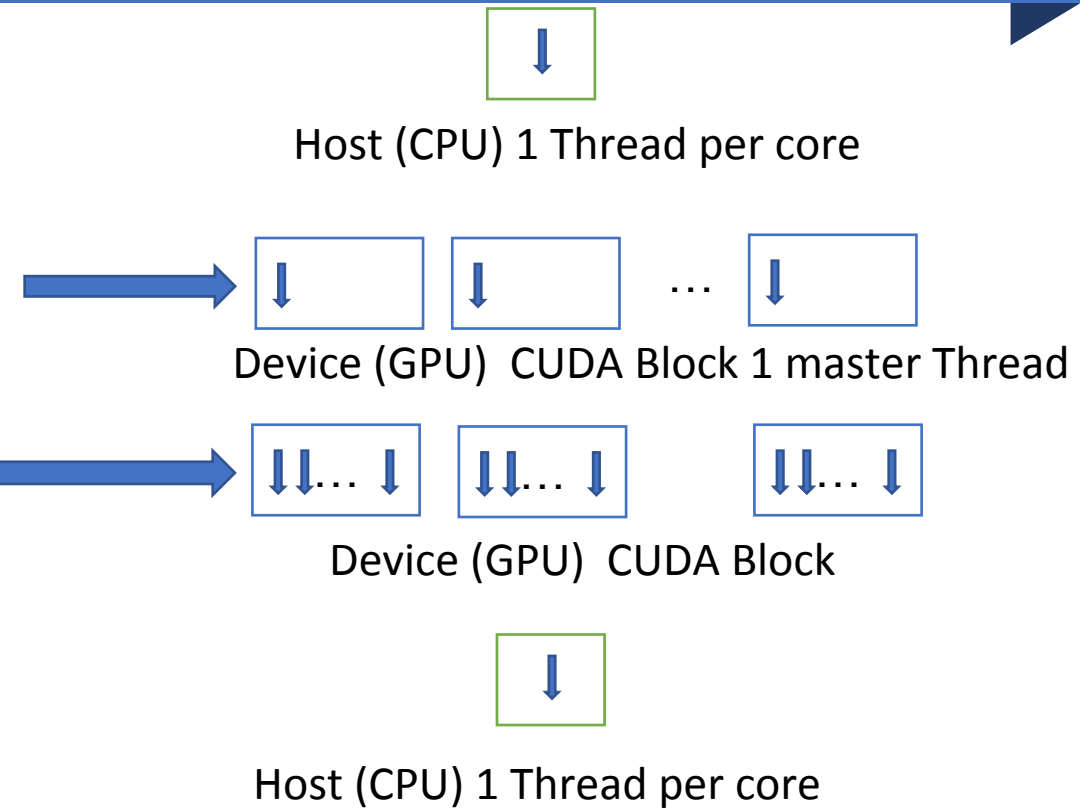
- Map a variable to/from the device data environment
- to (list) allocates memory on the device and copies data in when entering the region, the values are not copied back
- from(list) allocates memory on the device and copies the data to the host when exiting the region
- alloc(list) allocates memory on the device. If the data is already present on the device a reference counter is incremented

Complete set of clauses and examples on OpenMP 4.5 samples

#pragma omp target teams distribute

```
#pragma omp target map(to:x[0:N]), map(y[0:N])
{
   #pragma omp teams num_teams(N), thread_limit(M)
   //some sequential CPU code

   //pragma omp distribute parallel for
   for(int i=0; i<N;i++)
      y[i]=a*x[i];
   //some more sequential CPU code
}
```



Example: 2D Laplace Solver Sequential code

```
While(error >  tol && iter <iter_max){
    error=0.0;


    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]));
        }
    }
    for(int j=1; j<n-1;j++){
        for(int i=1;i<m-1;i++){
            A[j][i]=Anew[j][i];
        }
    }
}
```

1. This loop controls the number of times we calculate the stencil code in the next loops.
To calculate stencil at Anew at iteration t1 we need the values at iteration t0; so there is a clear loop dependency here.

2. Loop calculates for each element of array[row][col] a weighted average with the neighbors to north, south, east, and west
Then calculates the error

3. Copy Anew into A

Example: 2D Laplace Solver

```
#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];
        }
    }
}
```

Compiling Code

Need compiler version that supports OpenMP 4.5 pragma target

OpenMP:
gcc-5 supports offload to MIC (Xeon Phi)
gcc-7 supports offload to NVIDIA-PTX and AMD HSAIL
gcc-above adds more platforms to be offload

Configure the host compiler and device compiler to support offload pragma.
Follow instructions from either of the following if compiler is gcc/g++:

1 OpenMP offload

2 Building GCC with support for Nvidia ptx

3 First Steps with OpenMP 4.5 on Ubuntu and Nvidia GPUs

Compiling Code

Compile as:

gcc –O3 –fopenmp –foffload=nvptx-none –foffload=“-lm” –Wall

    –o solver solver.c

Execution time:

    ~4x sequential code

OpenMP Target offload vs OpenACC

Both directive based methods for programming “accelerators”

OpenMP 4.5 + Cray cc

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	24.12%	2.20986s	6	368.31ms	368.29ms	368.36ms	cffts1_neg
	7.94%	727.67ms	58	12.546ms	1.3440us	131.51ms	[CUDA memcpy DtoH]
	3.40%	311.86ms	56	5.5689ms	928ns	41.782ms	[CUDA memcpy HtD

OpenACC + Cray cc

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	32.83%	258.24ms	6	43.040ms	42.819ms	43.168ms	cffts1_neg
	13.09%	102.99ms	6	17.165ms	928ns	25.769ms	[CUDA memcpy HtoD]
	0.00%	9.9200us	6	1.6530us	1.4720us	1.9200us	[CUDA memcpy DtoH]

Pe
functionality in OpenMP 4.5

Full data Link (courtesy NASA): [OpenMP 4.5]

Conclusion

A. OpenMP 4.5 target offload did not lack a feature/functionality when compared with OpenACC
B. OpenMP 4.5 employs existing functionality for accelerator execution, if possible, e. g. “parallel for”, and “simd”
C. Compiler support for OpenMP would definitely benefit from further improvement
D. OpenMP offload support is getting increasingly stable

Module Student Activity

A. Ask Students to implement the complete Laplace Solver
   A. Using OpenMP parallel for pragma
   B. Using OpenMP target teams distribute for pragma
   C. Compare the results