Advanced OpenACC

Jeff Larkin, NVIDIA

How might we do transpose in OpenACC?


```
void time_kernel_acc(char* label, void
(*fptr)(int, int, float*, float*),  int r
cols, float * in, float * out) {
  ...

#pragma acc data
  {
  start = omp_get_wtime();
  for(int i=0; i<nReps; i++)
  {
    fptr(rows, cols, in, out);
  }
  #pragma acc wait
  end = omp_get_wtime();
  }
  ...
}
```

Copy data outside of timers, as in CUDA version.

Ensure work is complete before timing.

```
void openACCTranspose(int rows, int cols, float *
in, float * out)
{
  int i,j;

  #pragma acc parallel loop
  for(i=0; i<rows; i++)
  {
   for(j=0; j<cols; j++)
    {
      out[i*rows + j] = in[j*cols + i];
    }
  }

}
```

Parallelize loop nest

Something went wrong!

- Attempted to compile and it failed

```
$ CC -hgnu -hlist=md   -c main.c

      out[i*rows + j] = in[j*cols + i];
CC-7060 crayc++: ERROR File = main.c, Line = 40
  Unsupported OpenACC construct Unshaped C pointer -- in
CC-7060 crayc++: ERROR File = main.c, Line = 40
  Unsupported OpenACC construct Unshaped C pointer -- out

Total errors detected in main.c: 2
```

Compiler can't determine array bounds and gives up.

- Also visible in main.lst

How might we do transpose in OpenACC?


```
void time_kernel_acc(char* label, voi
(*fptr)(int, int, float*, float*),  i
cols, float * in, float * out) {
  ...

#pragma acc data copyin(in[0:rows*cols])
copyout(out[0:rows*cols])
  {
  start = omp_get_wtime();
  for(int i=0; i<nReps; i++)
  {
    fptr(rows, cols, in, out);
  }
  #pragma acc wait
  end = omp_get_wtime();
  }
  ...
}
```

Shape the Arrays

```
void openACCTranspose(int rows, int cols, float *
in, float * out)
{
  int i,j;

  #pragma acc parallel loop
present(in[:rows*cols],out[:rows*cols])
  for(i=0; i<rows; i++)
  {
   for(j=0; j<cols; j++)
    {
      out[i*rows + j] = in[j*cols + i];
    }
  }

}
```

Shape the Arrays

Now it builds and runs

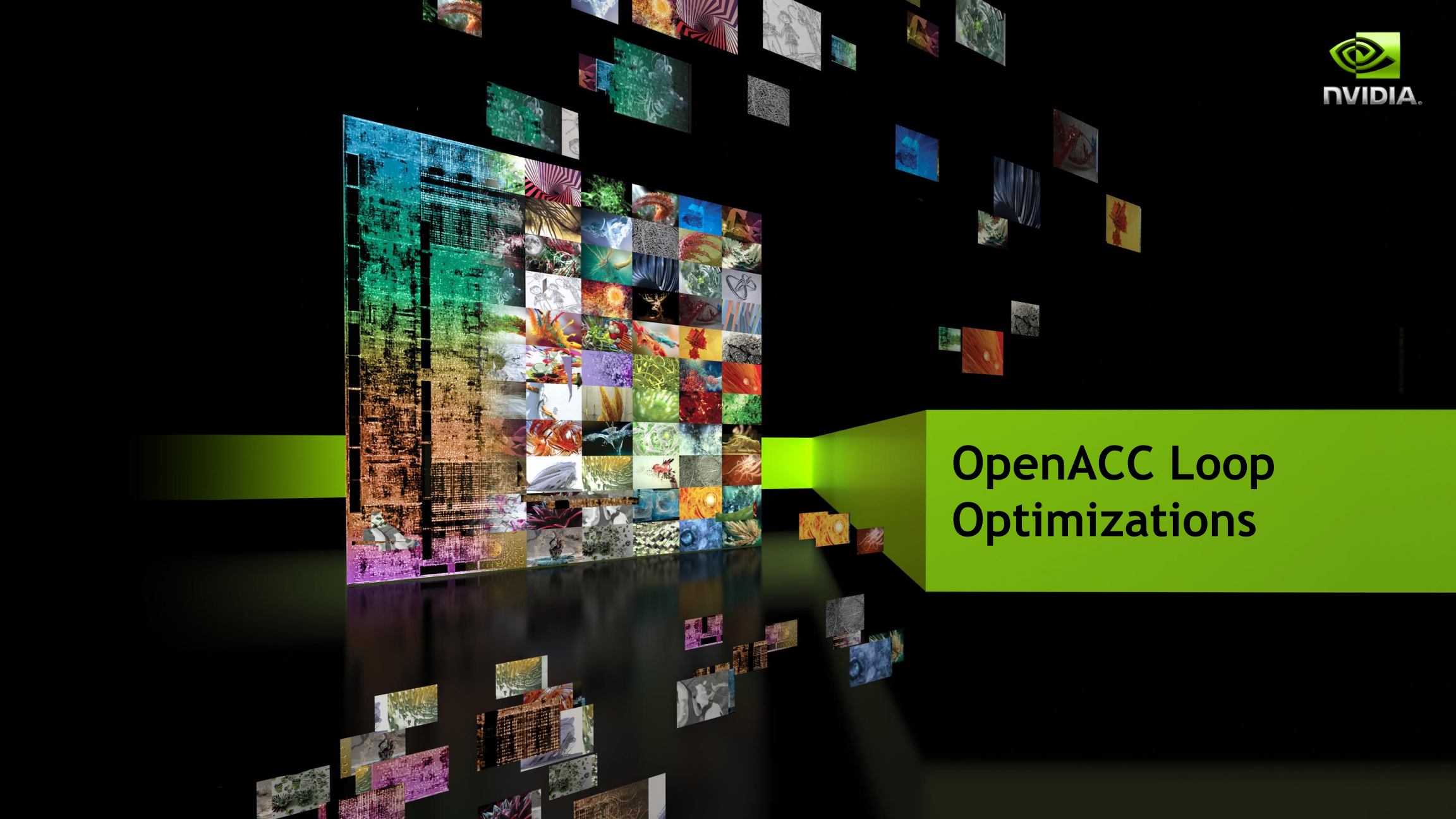
```
30.              void openACCTranspose(int rows,
int cols, float * in, float * out)
31.              {
32.                int i,j;
33.              #pragma acc parallel loop
present(in[rows*cols],out[rows*cols])
34.     gG-----<   for(i=0; i<rows; i++)
35.     gG         {
36.     gG g---<    for(j=0; j<cols; j++)
37.     gG g         {
38.     gG g           out[i*rows + j] = in[j*cols
+ i];
39.     gG g--->     }
40.     gG----->   }
41.              }
```

These loops are parallelized for the GPU.

```
(CRAY)
$ aprun -d 16 ./transpose
OpenMPI Processors: 16
CPU+OMP: Kernel bandwidth: 4.733409 gb/sec
CPU+OpenACC: Kernel bandwidth: 83.527385 gb/sec
CUDA-1D: Kernel bandwidth: 7.090239 gb/sec
CUDA-2D: Kernel bandwidth: 58.978922 gb/sec
CUDA-shared: Kernel bandwidth: 72.277753 gb/sec
CUDA-no-conflicts: Kernel bandwidth: 113.238345
gb/sec
CUDA-multi-element: Kernel bandwidth: 174.213759
gb/sec
```

```
(PGI)

CPU+OpenACC: Kernel bandwidth: 28.690365 gb/sec
```



OpenACC Loop Optimizations

OpenACC loop Directive

Programmer provides additional information to the compiler about the loop that immediately follows, such as decomposition, private variables, and reductions.

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

Affected Loop

*The loop directive does not require an “end” in Fortran.

3 Levels of Parallelism

`gang`	"Loose" parallelism, where gangs can work independently without synchronization. Roughly equivalent to a CUDA "block"
`worker`	"Tighter" parallelism, where workers may share data and/or coordinate within a common gang. Roughly equivalent to CUDA "threads"
`vector`	"Tightest" parallelism, where a vector instruction may be used across multiple data. Should be multiple of 32 on an Nvidia GPU.

On Nvidia GPUs, "worker" and "vector" are essentially interchangable.

Additional `loop` Clauses

`independent`	Loops within a kernels region are independent and may be overlapped.
`collapse(n)`	This loop should be merged with the next n loops to expose additional parallelism.
`reduction()`	Loop contains a parallel reduction, care must be taken by the compiler
`private()`	Each iteration needs a private copy of the listed variables.

Jacobi Iteration: OpenACC C Code


```
#pragma acc data copy(A), create(Anew)
while ( err > tol && iter < iter_max ) {
  err=0.0;

#pragma acc parallel loop reduction(max:err) vector_length(512)
  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]);

      err = max(err, abs(Anew[j][i] - A[j][i]);
    }
  }

#pragma acc parallel loop vector_length(512)
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

Increase from 128 threads to 512 threads.

Performance

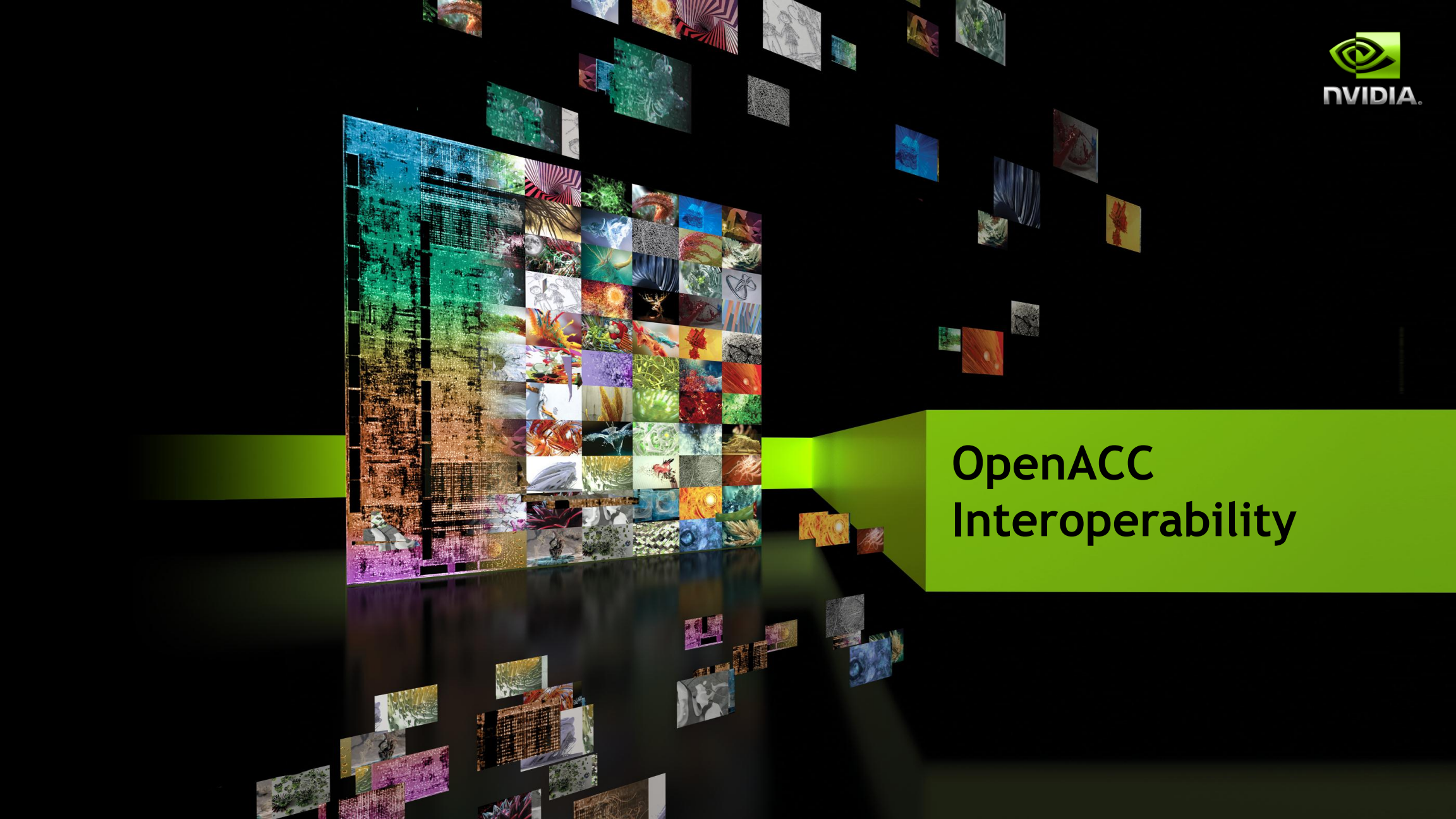
CPU: AMD IL-16
@ 2.2 GHz

Execution	Time (s)	Speedup
CPU 1 OpenMP thread	109.7	--
CPU 2 OpenMP threads	71.6	1.5x
CPU 4 OpenMP threads	53.7	2.0x
CPU 8 OpenMP threads	65.5	1.7x
CPU 16 OpenMP threads	66.7	1.6x
OpenACC GPU	4.92	10.9x

GPU: NVIDIA Tesla K20X

Speedup vs. 1 CPU core

Speedup vs. 4 OpenMP Threads

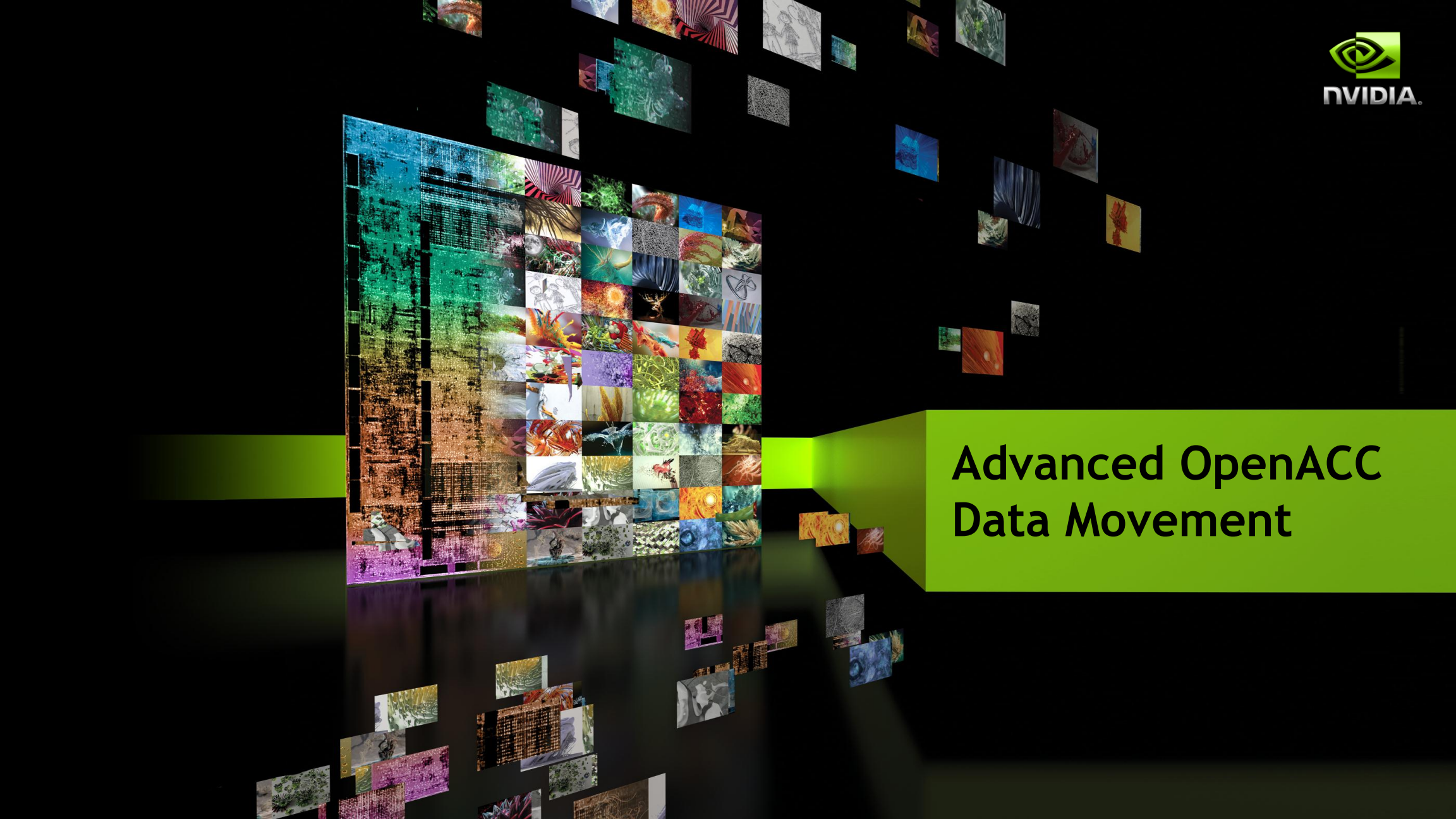

NVIDIA
OpenACC
Interoperability

OpenACC host_data Directive

Programmer differentiates between Host and Device copies for a given array within a data region. This is used mostly for CUDA/Library interoperability.

```
#pragma acc host_data use_device(A,B,C)
{
  cublasDgemm('N','N',N,N,N,1.0,A,N,B,N,1.0,C,N);
}
```

The function cublasDgemm is a host function that accepts device pointers A, B, and C.


NVIDIA
Advanced OpenACC
Data Movement

OpenACC update Directive

Programmer specifies an array (or partial array) that should be refreshed within a data region.

```
do_something_on_device()

!$acc update host(a)

do_something_on_host()

!$acc update device(a)
```

Copy “a” from GPU to CPU

Copy “a” from CPU to GPU

The programmer may choose to specify only part of the array to update.


Thank you
NVIDIA