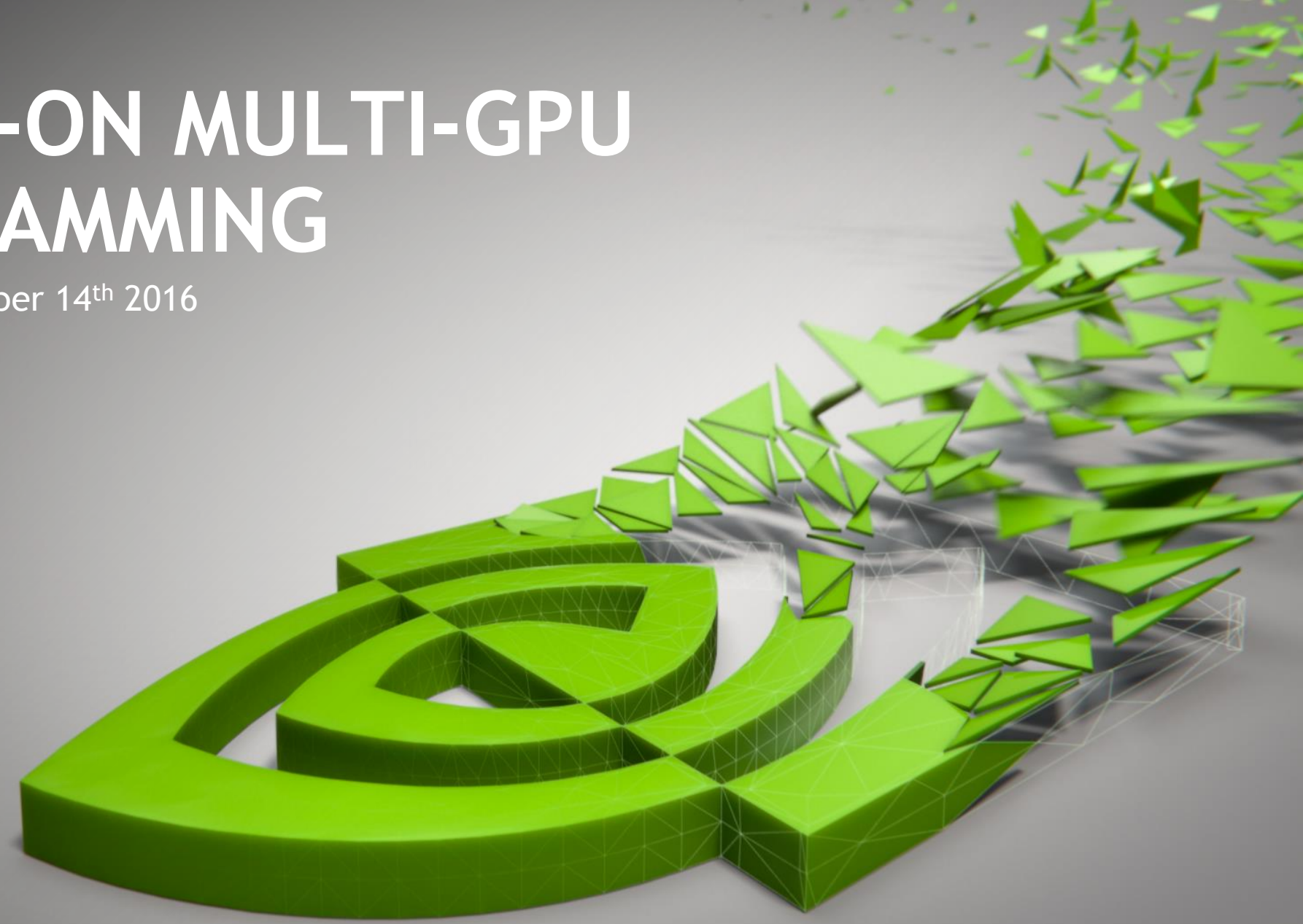
HANDS-ON MULTI-GPU PROGRAMMING

Jiri Kraus, November 14th 2016



HANDS-ON MENU

3 tasks to choose from

Task 0: Accelerate a CPU Jacobi solver with OpenACC relying on Unified Memory for data movement using -ta=tesla:**managed**

Task 1: Use MPI to make OpenACC accelerated Jacobi Solver scale to multiple GPUs

Task 2: Hide MPI communication time by overlapping communication and computation in a MPI+OpenACC multi GPU Jacobi Solver

USING PGPROF

Live Demo

TASK 0 REQUIREMENTS

`-ta=tesla:managed`

Automatically replaces all malloc/new/allocate with cudaMallocManaged

All heap memory is Unified Memory and can be used on the CPU and the GPU

No need for OpenACC data directives

Beta feature in PGI 16.10

TASK 0: USING OPENACC

[C|FORTRAN]/task0

Accelerate Jacobi with OpenACC using Unified Memory for data management

Parallelize Loops with OpenACC parallel loop

Look for TODOs

```
  800, 0.222691
  900, 0.219693
512x512: 1 CPU:    4.4272 s, 1 GPU:    4.4547 s, speedup:      0.99
```

Make Targets:

`run`:	run `poisson2d`
`poisson2d`:	build `poisson2d` bin (default)
`profile`:	profile with `pgprof`
`*.solution`:	same as above with solution (`poisson2d.solution.*`)

http://www.openacc.org/

TASK 1: APPLY DOMAIN DECOMPOSITION

[C|FORTRAN]/task1

Handle GPU affinity

Do Halo Exchange

Look for TODOs

```
Num GPUs: 4.
4096x4096: 1 GPU:   1.6991 s, 4 GPUs:   7.8438 s, speedup: 0.22
MPI time:   0.0000 s, inter GPU BW:  2496.34 GiB/s
```

Make Targets:

`run:`	run `poisson2d`
`poisson2d:`	build `poisson2d` bin (default)
`profile:`	profile with `pgprof`
`*.solution:`	same as above with solution (`poisson2d.solution.*`)

http://www.openacc.org/

https://www.open-mpi.org/doc/v1.10/

TASK 2: HIDE MPI COMMUNICATION TIME

[C|FORTRAN]/task2

Start copy loop asynchronously

Wait for async copy loop after MPI communication has finished

Look for TODOs

```
Num GPUs: 4.
4096x4096: 1 GPU:   1.6904 s, 4 GPUs:   0.6500 s, speedup: 2.60
MPI time:   0.0998 s, inter GPU BW:     1.22 GiB/s
```

Make Targets:

`run:`	run `poisson2d`
`poisson2d:`	build `poisson2d` bin (default)
`profile:`	profile with `pgprof`
`*.solution:`	same as above with solution (`poisson2d.solution.*`)

http://www.openacc.org/

https://www.open-mpi.org/doc/v1.10/

TASK 0: USING OPENACC

Solution

```
#pragma acc parallel loop
for (int iy = iy_start; iy < iy_end; iy++) {
  for( int ix = ix_start; ix < ix_end; ix++ ) {
    Anew[iy*nx+ix] = -0.25 * (rhs[iy*nx+ix]-(A[iy*nx+ix+1]  +A[iy*nx+ix-1]
                                             +A[(iy-1)*nx+ix]+A[(iy+1)*nx+ix]));
    error = fmaxr( error, fabsr(Anew[iy*nx+ix]-A[iy*nx+ix]));
  }
}
```

TASK 1: APPLY DOMAIN DECOMPOSITION

Solution

```
//Initialize MPI and determine rank and size
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

#pragma acc set device_num( rank )

real* restrict const A = (real*) malloc(nx*ny*sizeof(real));
real* restrict const Aref = (real*) malloc(nx*ny*sizeof(real));
real* restrict const Anew = (real*) malloc(nx*ny*sizeof(real));
real* restrict const rhs = (real*) malloc(nx*ny*sizeof(real));
```

TASK 1: APPLY DOMAIN DECOMPOSITION

Solution

```
// Ensure correctness if ny%size != 0
int chunk_size = ceil( (1.0*ny)/size );
int iy_start = rank * chunk_size;
int iy_end = iy_start + chunk_size;
// Do not process boundaries
iy_start = max( iy_start, 1 );
iy_end = min( iy_end, ny - 1 );
```

TASK 1: APPLY DOMAIN DECOMPOSITION

Solution

```
int top = (rank == 0) ? (size-1) : rank-1;
int bottom = (rank == (size-1)) ? 0 : rank+1;
#pragma acc host_data use_device( A ) {
  //1. Sent row iy_start (first modified row) to top receive lower boundary (iy_end)
  //from bottom
  MPI_Sendrecv( A+iy_start*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, top , 0,
                A+iy_end*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, bottom, 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE );
  //2. Sent row (iy_end-1) (last modified row) to bottom receive upper boundary (iy_start-1)
  //from top
  MPI_Sendrecv( A+(iy_end-1)*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, bottom, 0,
                A+(iy_start-1)*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, top , 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}
```

TASK 2: HIDE MPI COMMUNICATION TIME

Solution

```
#pragma acc parallel loop present(A,Anew)
for( int ix = ix_start; ix < ix_end; ix++ ) {
  A[(iy_start)*nx+ix] = Anew[(iy_start)*nx+ix];
  A[(iy_end-1)*nx+ix] = Anew[(iy_end-1)*nx+ix];
}
#pragma acc parallel loop present(A,Anew) async
for (int iy = iy_start+1; iy < iy_end-1; iy++) {
for( int ix = ix_start; ix < ix_end; ix++ ) {
  A[iy*nx+ix] = Anew[iy*nx+ix];
} }
int top = (rank == 0) ? (size-1) : rank-1;
int bottom = (rank == (size-1)) ? 0 : rank+1;
#pragma acc host_data use_device( A )
{
  MPI_Sendrecv( A+iy_start*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, top , 0,
                A+iy_end*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, bottom, 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE );
  MPI_Sendrecv( A+(iy_end-1)*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, bottom, 0,
                A+(iy_start-1)*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, top , 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}
#pragma acc wait
```

TASK 2: HIDE MPI COMMUNICATION TIME

Solution