MultiGPU programming

Presenter: Jiri Kraus (NVIDIA)
Suraj Prabhakaran | April 21, 2015

German Research School for Simulation Sciences GmbH
Laboratory for Parallel Programming

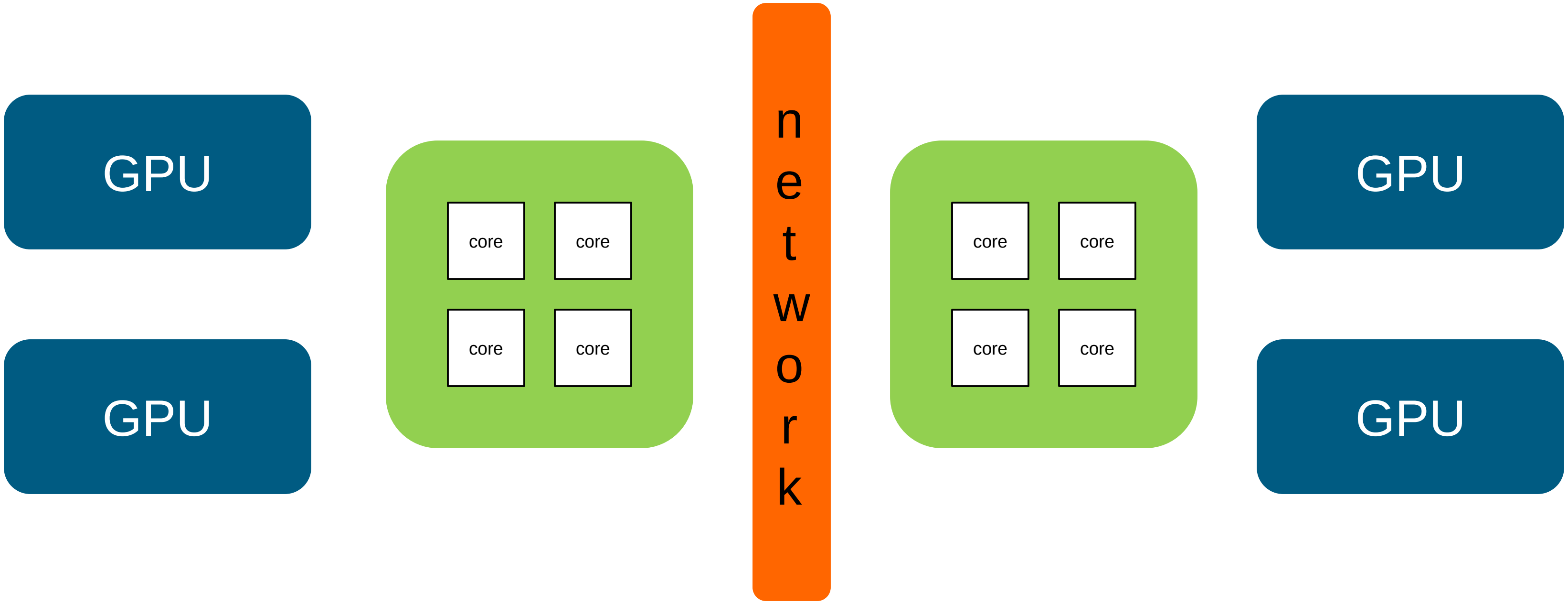
Using Multi GPUs

- Further speedup computations
- Single GPU memory not sufficient
- Increases performance/W

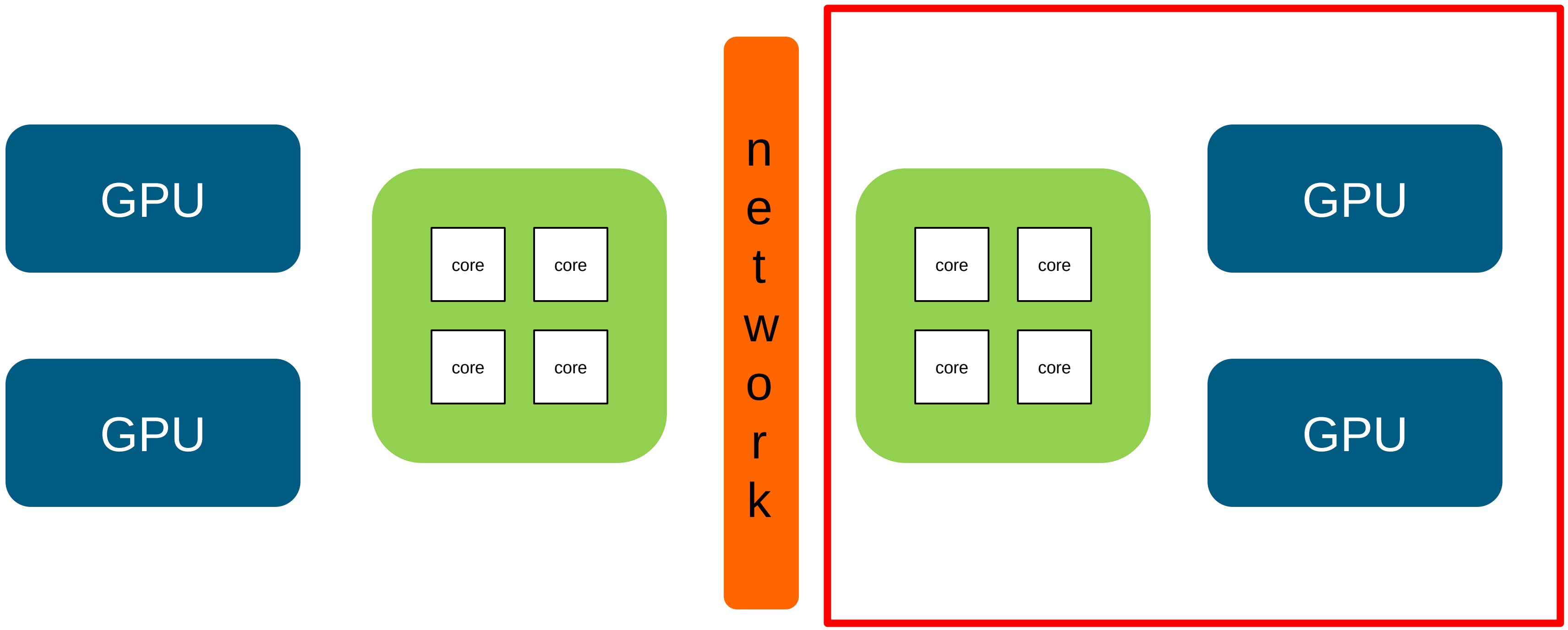
- Intra-node Multi-GPU
  - Easy-to-use, directly use the CUDA API

- Inter-node Multi-GPU
  - Network communication with MPI

Application scenario



Application scenario



Intra-node Multi-GPU

- Single CPU thread access Multiple GPUs
- CUDA calls issued to *current* GPU
- cudaSetDevice(x) sets the current GPU.
- Example

```
cudaSetDevice(0);
cudaMalloc(dst_0,…);
cudaMemcpy(dst_0, …);
cudaSetDevice(1);
cudaMalloc(dst_1,…);
cudaMemcpy(dst_1, …);
```

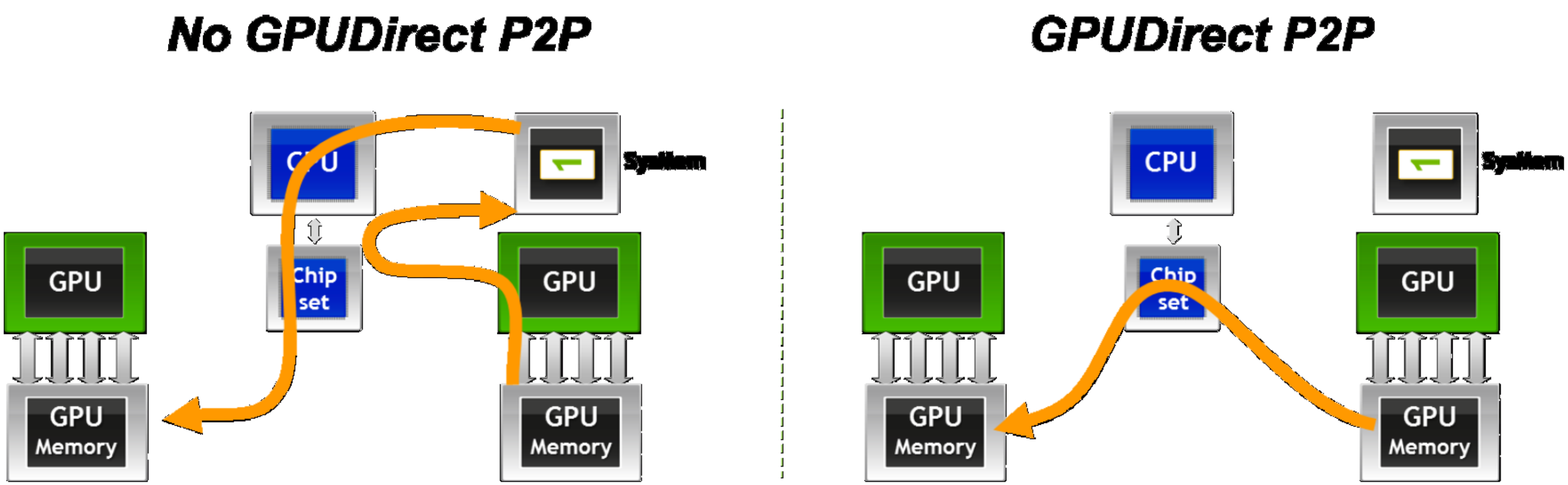
Intra-node Multi-GPU

- Current GPU can be changed even when async calls (kernels, async memcopies) are running
- Example

```
cudaSetDevice(0);
kernel<<<…>>>(…);
cudaSetDevice(1);
cudaMemcpyAsync(…);
```

Intra-node Multi-GPU Communication

- One GPU has to access data from another GPU
- Traditional method: Go about it through the CPU/Main Memory
- Due to UVA: Peer-to-peer memcopies (GPUDirect P2P)



Intra-node Multi-GPU Communication

- Check if the GPU can access Peer device

```
cudaDeviceCanAccessPeer(&canAccessPeer, devx, devy);
```

- First enable Peer-to-peer communication

```
cudaSetDevice(devx);
cudaDeviceEnablePeerAccess(devy,0);
```

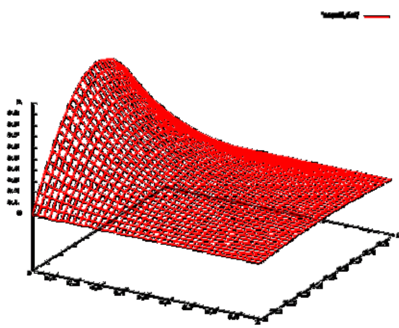
- Transfer data between two devices

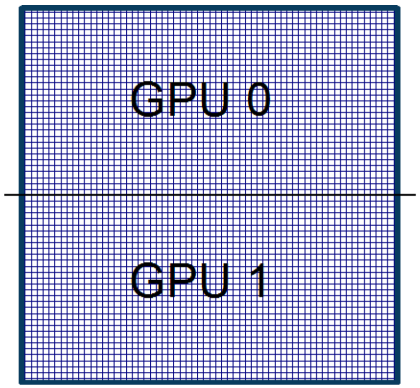
```
cudaMemcpy(dst, src, size, cudaMemcpyDeviceToDevice);
```

  - *Also works if peer access is not possible or not enabled (fall back with host memory staging)*

Hands-on Example: Jacobi

- Solves the 2D-Poission equation on a square
  - Dirichlet boundary conditions
- 1D domain decompostion with two domains





Hands-on Example: Jacobi

While not converged

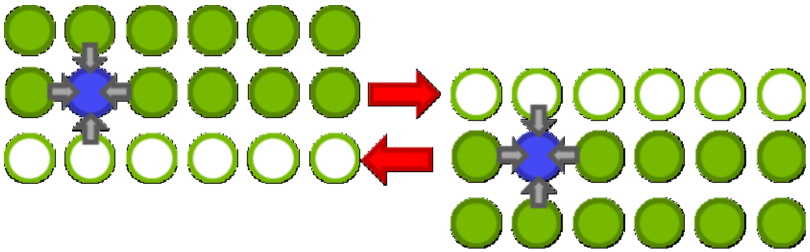
do Jacobi step on each GPU

```
for (int i=1; i < n-1; i++)
  for (int j=1; j < m-1; j++)
    u_new[i][j] = 0.0f - 0.25f*(u[i-1][j] + u[i+1][j]
                               + u[i][j-1] + u[i][j+1])
```

exchange halo between GPUs

copy `u_new` and `u` on each GPU

next iteration



Task: Modify the provided MPI+CUDA Jacobi to utilize CUDA-aware MPI

- TODOs in `MultiGPU/exercises/tasks/Jacobi/jacobi_cuda.c`
  - Add cudaSetDevice were necessary
  - Use cudaMemcpy to update halos
  - Enable Peer access
- Solution in

  `MultiGPU/exercises/solutions/Jacobi/jacobi_cuda.c`
- Slides are in

  `MultiGPU/slides/multigpu_20042015.pdf`