# 设备间通信

## cparallel\_mpi.c

包含：

MPI初始化（计算进程与IO进程分开）

阻塞式进程数据通信：MPI\_Send, MPI\_Recv，等价于SUNTANS的sendrecv.c程序。

## cparallel\_mpi\_gpu.cu

有如下数据交换函数API，但是都没有被调用过，见csolver\_gpu.cu。多种通信方式，包括6种：

exchange2d\_mpi\_gpu // 一般的MPI-CUDA通信

exchange3d\_mpi\_gpu

exchange2d\_cuda\_aware\_mpi // CUDA-aware MPI通信

exchange3d\_cuda\_aware\_mpi

exchange2d\_cuda\_ipc

exchange3d\_cuda\_ipc

exchange2d\_cudaPeer

exchange3d\_cudaPeer

exchange2d\_cudaPeerAsync

exchange3d\_cudaPeerAsync

exchange2d\_cudaUVA

exchange3d\_cudaUVA

还有一些施加周期边界条件的API，如xperi3d\_mpi\_gpu、yperi3d\_mpi\_gpu、。。。

exchange2d\_mpi\_gpu

float h\_work[j\_size][i\_size]; // 主机上开辟一个临时用于主机和设备间转数据的数组

(1) cudaMemcpy(h\_work, d\_work, nx\*ny\*sizeof(float), cudaMemcpyDeviceToHost); //将设备上的数组d\_work，传输到主机上的h\_work

(2) exchange2d\_mpi(h\_work, nx, ny); // 然后，MPI通信交换数据

(3) checkCudaErrors(cudaMemcpy(d\_work, h\_work, nx\*ny\*sizeof(float), cudaMemcpyHostToDevice)); // 交换后的数组再传输到设备上d\_work

exchange2d\_cuda\_aware\_mpi

gpuPOM采用的是非阻塞式通信

以east方向为例（因为是结构网格，分2个方向）：

float \*d\_send\_to\_east = d\_1d\_ny\_tmp0;

float \*d\_recv\_from\_east = d\_1d\_ny\_tmp3;

MPI\_Request request[2];

MPI\_Status status[2];

// 设备到设备的传输数据，到指针d\_send\_to\_east（待发送到东边的数据）

checkCudaErrors(cudaMemcpy2D(d\_send\_to\_east, sizeof(float),

d\_work+(nx-2), nx\*sizeof(float),

sizeof(float), ny,

cudaMemcpyDeviceToDevice));

MPI\_Isend(d\_send\_to\_east, ny, MPI\_FLOAT, n\_east, my\_task,

pom\_comm, &request[0]); // 发送到east

MPI\_Irecv(d\_recv\_from\_east, ny, MPI\_FLOAT, n\_east, n\_east,

pom\_comm, &request[1]); // 接收来自east

MPI\_Waitall(2, request, status); //同步（非阻塞通信必须的步骤）

// 最后，设备到设备的传输数据，来自east的数据

checkCudaErrors(cudaMemcpy2D(d\_work, nx\*sizeof(float), d\_recv\_from\_east, sizeof(float), sizeof(float), ny, cudaMemcpyDeviceToDevice));

exchange2d\_cuda\_ipc（IPC: Inter Process Communication）

仅支持64位Linux系统。

Using this API, an application can get the IPC handle for a given device memory pointer using cudaIpcGetMemHandle(), pass it to another process using standard IPC mechanisms (e.g., interprocess shared memory or files), and use cudaIpcOpenMemHandle() to retrieve a device pointer from the IPC handle that is a valid pointer within this other process. Event handles can be shared using similar entry points.

请参考cuda\_samples/SimpleIPC的示例代码

要仔细研究！

void exchange2d\_cuda\_ipc(float \*d\_send,

float \*d\_east\_recv,

float \*d\_west\_recv,

cudaStream\_t &stream\_in,

int nx, int ny){

1. MPI\_Barrier(pom\_comm);

2 checkCudaErrors(cudaMemcpy2DAsync(d\_east\_recv, nx\*sizeof(float),

d\_send+(nx-2), nx\*sizeof(float),

sizeof(float), ny,

cudaMemcpyDefault,

stream\_in));

cudaStreamSynchronize

1. MPI\_Barrier(pom\_comm);

gpuPOM模型中使用IPC通信方式的相关函数：

cudaIpcMemHandle\_t handle\_elf

void exchangeMemHandle()： cudaIpcGetMemHandle

void openMemHandle()

exchange2d\_cudaPeer（non-UVA通信方式）

cudaMemcpy3DPeerParms p\_east\_recv={0};

p\_east\_recv.extent = make\_cudaExtent(sizeof(float), ny, nz);

p\_east\_recv.dstDevice = n\_east;

p\_east\_recv.dstPtr = make\_cudaPitchedPtr(d\_east\_recv, nx\*sizeof(float), nx, ny);

p\_east\_recv.srcDevice = my\_task;

p\_east\_recv.srcPtr = make\_cudaPitchedPtr(d\_send+(nx-2), nx\*sizeof(float), nx, ny);

//! send ghost cell data to the east

checkCudaErrors(cudaMemcpy3DPeerAsync(&p\_east\_recv, stream\_in));

checkCudaErrors(cudaStreamSynchronize(stream\_in));

exchange2d\_cudaPeerAsync（non-UVA通信方式）

跟exchange2d\_cudaPeer的代码一样。得仔细研究一下。

exchange3d\_cudaUVA（当前使用的UVA通信方式）

**UVA: Unified Virtual Adress**

(1)MPI\_Barrier(pom\_comm);

//! send ghost cell data to the east

//! recieve ghost cell data from the west

(2) cudaMemcpy2DAsync

//! send ghost cell data to the north

//! recieve ghost cell data from the south

(3) cudaMemcpy2DAsync

(4) checkCudaErrors(cudaStreamSynchronize(stream\_in));

(5) MPI\_Barrier(pom\_comm);

多GPU的P2P复制和访问

Peer-to-peer memcpy，GPU A上的指针A直接数据复制到GPU B上的指针B

（1）使用UVA（统一虚拟地址）

仅使用cudaMemcpy(…, cudaMemcpyDefault)或者

cudaMemcpyAsync(…, cudaMemcpyDefault)

（2）使用non-UVA显式地做P2P复制

cudaError\_t cudaMemcpyPeer( void \* dst, int dstDevice, const void\* src,

int srcDevice, size\_t count )

cudaError\_t cudaMemcpyPeerAsync( void \* dst, int dstDevice,

const void\* src, int srcDevice, size\_t count, cuda\_stream\_t stream = 0 )

示例代码：

（1）当可使用UVA时，可使用cudaMemcpy用于peer-to-peer复制内存，因为CUDA可以推测设备“拥有”它自己的内存。实施代码如下：

//Check for peer access between participating GPUs:

cudaDeviceCanAccessPeer(&can\_access\_peer\_0\_1, gpuid\_0, gpuid\_1);

cudaDeviceCanAccessPeer(&can\_access\_peer\_1\_0, gpuid\_1, gpuid\_0);

//Enable peer access between participating GPUs:

cudaSetDevice(gpuid\_0);

cudaDeviceEnablePeerAccess(gpuid\_1, 0);

cudaSetDevice(gpuid\_1);

cudaDeviceEnablePeerAccess(gpuid\_0, 0);

//UVA memory copy:

cudaMemcpy(gpu0\_buf, gpu1\_buf, buf\_size, cudaMemcpyDefault);

（2）当没有UVA功能时，通过cudaMemcpyPeer执行peer-to-peer复制内存。

// Set device 0 as current

cudaSetDevice(0);

float\* p0;

size\_t size = 1024 \* sizeof(float);

// Allocate memory on device 0

cudaMalloc(&p0, size);

// Set device 1 as current

cudaSetDevice(1);

float\* p1;

// Allocate memory on device 1

cudaMalloc(&p1, size);

// Set device 0 as current

cudaSetDevice(0);

// Launch kernel on device 0

MyKernel<<<1000, 128>>>(p0);

// Set device 1 as current

cudaSetDevice(1);

// Copy p0 to p1

cudaMemcpyPeer(p1, 1, p0, 0, size);

// Launch kernel on device 1

MyKernel<<<1000, 128>>>(p1);

cudaMemcpyAsync：不同Stream之间传输数据使用的API，该函数在主机端是非阻塞的，传输处理后，控制权马上返回给主机线程。

cudaMemcpypeerAsync：cuda提供cudaMemcpyPeerAsync实现显卡间的数据复制，但是该函数需要硬件支持。可以使用cuda安装程序提供的p2pBandwidthLatencyTest测试当前硬件是否支持直接在显卡间实现数据复制。

总结

* 带Async的API函数，都涉及多流之间的数据交换。
* 带Peer的API，就是non-UVA，不需要cudaMallocHost(页锁定内存)；
* UVA方式：不带Peer，一般的cudaMemcpy用于多GPU间数据复制时，需要启动PeerAccess，即cudaDeviceEnablePeerAccess，然后cudaMemcpy使用参数cudaMemcpyDefault或cudaMemcpyDevicetoDevice