MSA SEMINAR: GPUS

Topics and Talks

• Today (21 January):
  CUDA-aware MPI (Jiri Kraus)
• Next (28 January):
  Arbor (Ben Cumming, CSCS)

→ https://fz-juelich.de/ias/jsc/msa-seminar
→ https://fz-juelich.de/ias/jsc/msa-seminar-slides

• After that:
  • OpenACC
  • SOMA
  • GPU Libraries
  • QUDA
  • QCD
  • PIConGPU
MULTI-GPU PROGRAMMING WITH MPI

Jiri Kraus, Senior Developer Technology Engineer, NVIDIA
MPI+CUDA
MPI+CUDA

Node 0
- MEM
- CPU
- GPU
- PCIe Switch
- IB

Node 1
- MEM
- CPU
- GPU
- PCIe Switch
- IB

Node N-1
- MEM
- CPU
- GPU
- PCIe Switch
- IB

...
/**MPI+CUDA**

//MPI rank 0
MPI_Send(s_buf_d, size, MPI_CHAR, n-1, tag, MPI_COMM_WORLD);

//MPI rank n-1
MPI_Recv(r_buf_d, size, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &stat);
YOU WILL LEARN

What MPI is
How to use MPI for inter GPU communication with CUDA and OpenACC
What Multi Process Service is and how to use it
How to use NVIDIA tools in an MPI environment
How to hide MPI communication times
A SIMPLE EXAMPLE
EXAMPLE: JACOBI SOLVER

Solves the 2D-Laplace Equation on a rectangle

$$\Delta u(x, y) = 0 \ \forall \ (x, y) \in \Omega \setminus \delta \Omega$$

Dirichlet boundary conditions (constant values on boundaries)

$$u(x, y) = f(x, y) \ \forall \ (x, y) \in \delta \Omega$$

2D domain decomposition with n x k domains
EXAMPLE: JACOBI SOLVER

Single GPU

While not converged
Do Jacobi step:

```c
for (int iy=1; iy < ny-1; ++iy)
for (int ix=1; ix < nx-1; ++ix)

    u_new[ix][iy] = 0.0f - 0.25f*( u[ix-1][iy] + u[ix+1][iy]
                                 + u[ix][iy-1] + u[ix][iy+1]);
```

Swap u_new and u

Next iteration
EXAMPLE: JACOBI SOLVER

Multi GPU

While not converged

Do Jacobi step:

```c
for (int iy=1; iy < ny-1; ++iy)
    for (int ix=1; ix < nx-1; ++ix)
        u_new[ix][iy] = 0.0f - 0.25f*(
            u[ix-1][iy] + u[ix+1][iy]
            + u[ix][iy-1] + u[ix][iy+1]);
```

Exchange halo with 1 to 4 neighbors

Swap u_new and u

Next iteration
EXAMPLE JACOBI
Top/Bottom Halo

MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);

MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
#pragma acc host_data use_device ( u_new ) {
MPI_Sendrecv( u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv( u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
             u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}

MPI_Sendrecv( u_new_d+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new_d+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv( u_new_d+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
             u_new_d+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
EXAMPLE: JACOBI

Left/Right Halo

```c
//right neighbor omitted
#pragma acc parallel loop present ( u_new, to_left )
for ( int i=0; i<n-2; ++i )
    to_left[i] = u_new[(i+1)*m+1];

#pragma acc host_data use_device ( from_right, to_left )
{
    MPI_Sendrecv( to_left, n-2, MPI_DOUBLE, l_nb, 0,
                 from_right, n-2, MPI_DOUBLE, r_nb, 0,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}

#pragma acc parallel loop present ( u_new, from_right )
for ( int i=0; i<n-2; ++i )
    u_new[(m-1)+(i+1)*m] = from_right[i];
```
EXAMPLE: JACOBI

Left/Right Halo

//right neighbor omitted
pack<<<gs,bs,0,s>>>(to_left_d, u_new_d, n, m);
cudaStreamSynchronize(s);

MPI_Ssendrecv( to_left_d, n-2, MPI_DOUBLE, l_nb, 0,
from_right_d, n-2, MPI_DOUBLE, r_nb, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE );

unpack<<<gs,bs,0,s>>>(u_new_d, from_right_d, n, m);
CUDA-AWARE MPI ON JUWELS

ParaStation MPI:
GCC+CUDA:
module load GCC/8.3.0 ParaStationMPI/5.4.2-1-CUDA
PGI+CUDA 10.1:
module load PGI/19.3-GCC-8.3.0 ParaStationMPI/5.4.2-1-CUDA

MVAPICH2-GDR:
GCC+CUDA:
module load GCC/8.3.0 MVAPICH2/2.3.3-GDR
PGI+CUDA 10.1:
module load PGI/19.3-GCC-8.3.0 MVAPICH2/2.3.3-GDR
LAUNCH MPI+CUDA/OPENACC PROGRAMS

Launch one process per GPU

ParaStation MPI: $ PSP_CUDA=1 mpirun -np ${np} ./myapp <args>

MVAPICH: $ MV2_USE_CUDA=1 mpirun -np ${np} ./myapp <args>

Open MPI: CUDA-aware features are enabled per default

Cray: MPICH_RDMA_ENABLED_CUDA

IBM Spectrum MPI: $ mpirun -gpu -np ${np} ./myapp <args>

set by module on JUWELS
HANDLING MULTIPLE MULTI GPU NODES
HANDLING MULTIPLE MULTI GPU NODES
How to determine the local rank? - MPI-3

```c
MPI_Comm loc_comm;

MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, rank, MPI_INFO_NULL, &loc_comm);

int local_rank = -1;

MPI_Comm_rank(loc_comm,&local_rank);

MPI_Comm_free(&loc_comm);
```
HANDLING MULTIPLE MULTI GPU NODES
HANDLING MULTIPLE MULTI GPU NODES

GPU-affinity

Use local rank:

```c
int local_rank = -1;
MPI_Comm_rank(local_comm,&local_rank);
int num_devices = 0;
cudaGetDeviceCount(&num_devices);
cudaSetDevice(local_rank % num_devices);
```
EXAMPLE JACOBI
Top/Bottom Halo

```c
#pragma acc update host(u_new[offset_first_row:m-2], u_new[offset_last_row:m-2])
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
             u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
#pragma acc update device(u_new[offset_top_boundary:m-2], u_new[offset_bottom_boundary:m-2])

//send to bottom and receive from top top bottom bottom omitted

cudaMemcpy(  u_new+offset_first_row,
             u_new_d+offset_first_row, (m-2)*sizeof(double), cudaMemcpyDeviceToHost);
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
cudaMemcpy(  u_new_d+offset_bottom_boundary,
             u_new+offset_bottom_boundary, (m-2)*sizeof(double), cudaMemcpyDeviceToHost);
```

without CUDA-aware MPI
THE DETAILS
UNIFIED VIRTUAL ADDRESSING

No UVA: Separate Address Spaces

UVA: Single Address Space

System Memory

GPU Memory

CPU

GPU

PCI-e

System Memory

GPU Memory

CPU

GPU

PCI-e
UNIFIED VIRTUAL ADDRESSING

One address space for all CPU and GPU memory

Determine physical memory location from a pointer value

Enable libraries to simplify their interfaces (e.g. MPI and cudaMemcpy)

Supported on devices with compute capability 2.0+ for

64-bit applications on Linux and Windows (+TCC)
NVIDIA GPUDIRECT™
Peer to Peer Transfers
NVIDIA GPDIRECT™

Support for RDMA
CUDA-AWARE MPI

Example:

MPI Rank 0 `MPI_Send` from GPU Buffer

MPI Rank 1 `MPI_Recv` to GPU Buffer

Show how CUDA+MPI works in principle

Depending on the MPI implementation, message size, system setup, ... situation might be different

Two GPUs in two nodes
MPI_GPU_TO_REMOTE_GPU

Support for RDMA

MPI_Rank_0

MPI_Rank_1

GPU

Host

MPI_Send(s_buf_d,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

MPI_Recv(r_buf_d,size,MPI_CHAR,0,tag,MPI_COMM_WORLD,&stat);
MPI GPU TO REMOTE GPU

Support for RDMA

MPI_Sendrecv

Time
cudaMemcpy(s_buf_h, s_buf_d, size, cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h, size, MPICHAR, 1, tag, MPI_COMM_WORLD);

MPI_Recv(r_buf_h, size, MPICHAR, 0, tag, MPI_COMM_WORLD, &stat);
cudaMemcpy(r_buf_d, r_buf_h, size, cudaMemcpyHostToDevice);
REGULAR MPI GPU TO REMOTE GPU

memcpy D→H | MPI_Sendrecv | memcpy H→D

Time
MPI_GPU_TO_REMOTE_GPU

without GPUDirect

MPI_Send(s_buf_h, size, MPI_CHAR, 1, tag, MPI_COMM_WORLD);

MPI_Recv(r_buf_h, size, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &stat);
MPI GPU TO REMOTE GPU
without GPUDirect

MPI_Sendrecv

Time
PERFORMANCE RESULTS GPUDIRECT RDMA

OSU BW ParaStation MPI 5.4.2-1 JUWELS - Tesla V100

Latency (1 Byte) 20.16 us 19.59 us 3.12 us

* UCX_RNDV_SCHEME=get_zcopy
* UCX_RNDV_THRESH=131072
PERFORMANCE RESULTS GPUDIRECT P2P

OSU BW ParaStation MPI 5.4.2-1 JUWELS - Tesla V100

Message Size (Byte)

Latency (1 Byte)  19.55 us  18.54 us  2.27 us
MULTI PROCESS SERVICE (MPS) FOR MPI APPLICATIONS
GPU ACCELERATION OF LEGACY MPI APPS

Typical legacy application

- MPI parallel
- Single or few threads per MPI rank (e.g. OpenMP)

Running with multiple MPI ranks per node

GPU acceleration in phases

- Proof of concept prototype, ...
- Great speedup at kernel level

Application performance misses expectations
**MULTI PROCESS SERVICE (MPS)**

For Legacy MPI Applications

With MPS
Available on Tesla/Quadro since CC 3.5

Multicore CPU only

GPU-accelerated
PROCESSES SHARING GPU WITHOUT MPS

No Overlap

Process A

Context A

Process B

Context B

GPU
PROCESSES SHARING GPU WITHOUT MPS

Context Switch Overhead

Context Switch

Context Switch
PROCESSES SHARING GPU WITH MPS

Maximum Overlap

Process A
  Context A

Process B
  Context B

MPS

GPU

Kernels from Process A

Kernels from Process B
PROCESSES SHARING GPU WITH MPS

No Context Switch Overhead
MPS CASE STUDY: RELION

Enables overlap between copy and compute of different processes

GPU sharing between MPI ranks increases utilization
USING MPS

No application modifications necessary

Not limited to MPI applications

MPS control daemon

Spawn MPS server upon CUDA application startup

#With Slurm 19.05+

? 

#On Cray systems

export CRAY_CUDA_MPS=1

#Manually

nvidia-smi -c EXCLUSIVE_PROCESS

nvidia-cuda-mps-control -d
MPS: IMPROVEMENTS WITH VOLTA

More MPS clients per GPU: 48 instead of 16

Less overhead: Volta MPS clients submit work directly to the GPU without passing through the MPS server.

More security: Each Volta MPS client owns its own GPU address space instead of sharing GPU address space with all other MPS clients.

More control: Volta MPS supports limited execution resource provisioning for Quality of Service (QoS). -> CUDA_MPS_ACTIVE_THREAD_PERCENTAGE
MPS SUMMARY

Easy path to get GPU acceleration for legacy applications
Enables overlapping of memory copies and compute between different MPI ranks
Remark: MPS adds some overhead!
DEBUGGING AND PROFILING
TOOLS FOR MPI+CUDA APPLICATIONS

Memory checking: cuda-memcheck

Debugging: cuda-gdb

Profiling: NVIDIA Nsight Systems
MEMORY CHECKING WITH CUDA-MEMCHECK

cuda-memcheck is a tool similar to Valgrind’s memcheck

Can be used in a MPI environment

```
mpiexec -np 2 cuda-memcheck ./myapp <args>
```

Problem: Output of different processes is interleaved

Solution: Use save or log-file command line options

```
mpirun -np 2 cuda-memcheck \
   --log-file name.%q{OMPI_COMM_WORLD_RANK}.log \
   --save name.%q{OMPI_COMM_WORLD_RANK}.memcheck \
   ./myapp <args>
```

OpenMPI: OMPI_COMM_WORLD_RANK
MVAPICH2: MV2_COMM_WORLD_RANK
MEMORY CHECKING WITH CUDA-MEMCHECK
MEMORY CHECKING WITH CUDA-MEMCHECK

Read Output Files with `cuda-memcheck --read`
DEBUGGING MPI+CUDA APPLICATIONS

Using `cuda-gdb` with MPI Applications

Use `cuda-gdb` just like `gdb`

For smaller applications, just launch `xterms` and `cuda-gdb`

```
mpiexec -x -np 2 xterm -e cuda-gdb ./myapp <args>
```
if ( rank == 0 ) {
    int i=0;
    printf("rank %d: pid %d on %s ready for attach\n.", rank, getpid(), name);
    while (0 == i) { sleep(5); }
}

> mpiexec -np 2 ./jacobi_mpi+cuda
Jacobi relaxation Calculation: 4096 x 4096 mesh with 2 processes and one Tesla M2070 for each process (2049 rows per process).
rank 0: pid 30034 on judge107 ready for attach
> ssh judge107
jkraus@judge107:~> cuda-gdb --pid 30034
DEBUGGING MPI+CUDA APPLICATIONS

CUDA_DEVICE_WAITS_ON_EXCEPTION

The application encountered a device error and CUDA
may now attach a debugger to the application.

CUDA Exception: Device Illegal Address
The exception was triggered in device 3.

Program received signal CUDA_EXCEPTION_10, Device Illegal Address.

```
0x0000000000000000 in checkError (cudaKernel=0x2300000000000000, devId=0x2300000000000000, err=0x2300000000000000, devId=0x2300000000000000, devRes=0x2300000000000000, flags=0x1)
```

(cuda-gdb) br
0x0000000000000000 in checkError (cudaKernel=0x2300000000000000, devId=0x2300000000000000, err=0x2300000000000000, devRes=0x2300000000000000, flags=0x1)
(cuda-gdb) s
DEBUGGING MPI+CUDA APPLICATIONS

With `CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1` core dumps are generated in case of an exception:

- Can be used for offline debugging
- Helpful if live debugging is not possible

`CUDA_ENABLE_CPU_COREDUMP_ON_EXCEPTION`: Enable/Disable CPU part of core dump (enabled by default)

`CUDA_COREDUMP_FILE`: Specify name of core dump file

Open GPU:  
```
(cuda-gdb) target cudacore core.cuda
```

Open CPU+GPU:  
```
(cuda-gdb) target core core.cpu core.cuda
```
DEBUGGING MPI+CUDA APPLICATIONS

CUDA_ENABLE_COREDUMP_ON_EXCEPTION

```bash
[jkraus@ivb114 solutions]$ CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1 mpirun -x CUDA_ENABLE_COREDUMP_ON.Exception -np 2 ./jacobi_mpi+cuda
Jacobi relaxation Calculation: 4096 x 4096 mesh with 2 processes and one Tesla K40m for each process (2049 rows per process).

mpirun noticed that process rank 1 with PID 28722 on node ivb114 exited on signal 11 (Segmentation fault).
```

```bash
[jkraus@ivb114 solutions]$ ls core.*
core.cuda.ivb114.28722 core.cuda.ivb114.28723
[jkraus@ivb114 solutions]$`
```
DEBUGGING MPI+CUDA APPLICATIONS

CUDA_ENABLE_COREDUMP_ON_EXCEPTION

![CUDA Debugger screenshot showing an exception in a CUDA application](image.png)
DEBUGGING MPI+CUDA APPLICATIONS

Third Party Tools

Allinea DDT debugger

Rogue Wave TotalView
PROFILING MPI+CUDA APPLICATIONS
Using Nsight Systems

Trace MPI and embed MPI rank in output filename (OpenMPI)

```
mpirun -np $np nsys profile -o profile.%q{OMPI_COMM_WORLD_RANK} \n  --trace=mpi,cuda --mpi-impl openmpi
```

MVAPICH2:

```
MVAPICH2: MV2_COMM_WORLD_RANK
  --mpi-impl mpich
```
PROFILING MPI+CUDA APPLICATIONS
Using Nsight Systems
PROFILING MPI+CUDA APPLICATIONS

Using Nsight Systems
Multiple parallel profiling tools are CUDA-aware

Score-P
Vampir
Tau

These tools are good for discovering MPI issues as well as basic CUDA performance inhibitors.
ADVANCED MPI ON GPUS
BEST PRACTICE: USE NON-BLOCKING MPI

```c
#pragma acc host_data use_device ( u_new ) {
    MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
                  u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
                  MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
                  u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,
                  MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}

MPI_Request t_b_req[4];
#pragma acc host_data use_device ( u_new ) {
    MPI_Irecv(u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 0,
               MPI_COMM_WORLD, t_b_req[0]);
    MPI_Irecv(u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 1,
               MPI_COMM_WORLD, t_b_req[1]);
    MPI_Isend(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 0,
              MPI_COMM_WORLD, t_b_req[2]);
    MPI_Isend(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 1,
              MPI_COMM_WORLD, t_b_req[3]);
}
MPI_Waitall(4, t_b_req, MPI_STATUSES_IGNORE);
```

Gives MPI more opportunities to build efficient pipelines
COMMUNICATION + COMPUTATION OVERLAP
ParaStation MPI 5.4.2-1 JUWELS - Tesla V100 - Jacobi on 18432x18432

Source: https://github.com/NVIDIA/multi-gpu-programming-models/
COMMUNICATION + COMPUTATION OVERLAP

No Overlap

Process Whole Domain

Overlap

Boundary and inner domain processing can overlap

Process inner domain

Process boundary domain

Dependency

Possible gain

MPI
process_boundary_and_pack<<<gs_b,bs_b,0,s1>>>(u_new_d,u_d,to_left_d,to_right_d,n,m);

process_inner_domain<<<gs_id,bs_id,0,s2>>>(u_new_d,u_d,to_left_d,to_right_d,n,m);

cudaStreamSynchronize(s1); //wait for boundary
MPI_Request req[8];

//Exchange halo with left, right, top and bottom neighbor

MPI_Waitall(8, req, MPI_STATUSES_IGNORE);
unpack<<<gs_s,bs_s,0,s2>>>(u_new_d, from_left_d, from_right_d, n, m);

cudaDeviceSynchronize(); //wait for iteration to finish
COMMUNICATION + COMPUTATION OVERLAP
OpenACC with Async Queues

```c
#pragma acc parallel loop present ( u_new, u, to_left, to_right ) async(1)
for ( ... )
    //Process boundary and pack to_left and to_right
#pragma acc parallel loop present ( u_new, u ) async(2)
for ( ... )
    //Process inner domain
#pragma acc wait(1) //wait for boundary
MPI_Request req[8];
#pragma acc host_data use_device ( from_left, to_left, form_right, to_right, u_new ) {
    //Exchange halo with left, right, top and bottom neighbor
}
MPI_Waitall(8, req, MPI_STATUSES_IGNORE);
#pragma acc parallel loop present ( u_new, from_left, from_right ) async(2)
for ( ... )
    //unpack from_left and from_right
#pragma acc wait //wait for iteration to finish
```
COMMUNICATION + COMPUTATION OVERLAP
ParaStation MPI 5.4.2-1 JUWELS - Tesla V100 - Jacobi on 18432x18432

Source: https://github.com/NVIDIA/multi-gpu-programming-models/
Improve scalability with high priority streams

cudaStreamCreateWithPriority

Use-case: MD Simulations

Stream 1
- Comp. Local Forces

Stream 2

Stream 1 (LP)
- Comp. Local Forces

Stream 2 (HP)

Possible gain
MPI AND UNIFIED MEMORY

CAVEAT

Using Unified Memory with a non Unified Memory-aware MPI might fail with errors or even worse silently produce wrong results, e.g. when registering Unified Memory for RDMA.

Use a Unified Memory-aware MPI,

e.g. UCX based MPI (ParaStation, OpenMPI,...) or MVAPICH2-GDR since 2.2b

Unified Memory-aware: CUDA-aware MPI with support for Unified Memory
Unified Memory can be used by any processor in the system.

Memory pages of a Unified Memory allocation may migrate between processors memories to ensure coherence and maximize performance.

Different data paths are optimal for performance depending on where the data is: e.g. NVLink between peer GPUs.

The MPI implementation needs to know where the data is, but it can’t!
MPI AND UNIFIED MEMORY
Performance Implications - Simple Example

cudaMallocManaged( &array, n*sizeof(double), cudaMemAttachGlobal );

while( ... ) {
    foo(array,n);
    MPI_Send(array,...);
    foo(array,n);
    foo(array,n);
}

MPI AND UNIFIED MEMORY

Performance Implications - Simple Example

- If foo is a CPU function pages of array might migrate to System Memory
- If foo is a GPU function pages of array might migrate to GPU Memory
- The MPI implementation is not aware of the application and thus doesn't know where array is and what's optimal

```c
while(...) {
    foo(array, n);
    MPI_Send(array, ...);
    foo(array, n);
}
```
MPI AND UNIFIED MEMORY
The Future with Data Usage Hints

Tell where the application intends to use the data

cudaMallocManaged( &array, n*sizeof(double), cudaMemAttachGlobal );

cudaMemAdvise(array,n*sizeof(double),cudaMemAdviseSetPreferredLocation,device);

while( ... ) {
    foo(array,n);
    MPI_Send(array,...);
    foo(array,n);
}

Remark: Data Usage Hints are available since CUDA 8, but currently not evaluated by any Unified Memory-aware MPI implementation.
MPI AND UNIFIED MEMORY

The Future with Data Usage Hints

Tell where the application intends to use the data

cudaMallocManaged( &array, n*sizeof(double), cudaMemAttachGlobal );

cudaMemAdvise(array,n*sizeof(double),cudaMemAdviseSetPreferredLocation, cudaCpuDeviceId);

while( ... ) {

    foo(array,n);

    MPI_Send(array,...);

    foo(array,n);

}

Remark: Data Usage Hints are available since CUDA 8, but currently not evaluated by any Unified Memory-aware MPI implementation.
Data usage hints can be queried by the MPI Implementation and allow it to take the optimal data path

If the application lies about the data usage hints it will run correctly but performance will be affected

Performance tools help to identify missing or wrong data usage hints

Data usage hints are general useful for the Unified Memory system and can improve application performance.

Remark: Data Usage Hints are only hints to guide the data usage policies of the Unified Memory system. The Unified Memory system might ignore them, e.g. to ensure coherence or in oversubscription scenarios.
MPI AND UNIFIED MEMORY

Current Status

Available Unified Memory-aware MPI implementations

- UCX-based MPIs, e.g. OpenMPI and ParaStation MPI
- MVAPICH2-GDR (since 2.2b)

Currently both don’t evaluate Data Usage Hints, i.e. all Unified Memory is treated as Device Memory

Potential performance issues if not all buffers used in MPI are touched mainly on the GPU.
MPI AND UNIFIED MEMORY
Without Unified Memory-aware MPI

Only use non Unified Memory Buffers for MPI: cudaMalloc, cudaMallocHost or malloc

Application managed non Unified Memory Buffers also allow to work around current missing cases in Unified Memory-aware MPI Implementations.
DETECTING CUDA-AWARENESS

ParaStation MPI and OpenMPI (since 2.0.0) via mpi-ext.h

Macro:

```
MPIX_CUDA_AWARE_SUPPORT
```

Function for runtime decisions

```
MPIX_Query_cuda_support()
```

See [http://www.open-mpi.org/faq/?category=runcuda#mpi-cuda-aware-support](http://www.open-mpi.org/faq/?category=runcuda#mpi-cuda-aware-support)

ParaStation MPI: MPI_INFO_ENV

```
MPI_Info_get(MPI_INFO_ENV, "cuda_aware",
    sizeof(is_cuda_aware)-1, is_cuda_aware,
    &api_available);
```