Parallel programming with CUDA and MPI

Sebastian von Alfthan
Content

• Introduction
• MPI + CUDA programming
• MPI + OpenMP + CUDA programming
• Recent developments
• Summary
Introduction

- **Three levels of hardware parallelism**
  1. GPU - threads on the multiprocessors
  2. Node – Binding together GPU, CPU and NIC (network card)
  3. Machine - Several nodes connected with Infiniband
Introduction

- **Parallelization strategies**
  1. CUDA
  2. Threads (OpenMP, Pthreads) or MPI
  3. MPI
MPI + CUDA
MPI+CUDA topics

- Introduction
- CUDA+MPI strategies: How to share the GPU resource
- Selecting GPU
- Combining MPI_Send and MPI_Receive with CUDA
MPI + CUDA is easy

- **Cuda and MPI can be considered separate entities**
  - CUDA handles parallelization on GPU
  - MPI handles parallelization over nodes
- **Use one MPI process per GPU and accelerate the computational kernels with CUDA**
- **To transfer data between to devices**
  - Sender: Copy data from device to temporary host buffer
  - Sender: Send host buffer data
  - Receiver: Receive data to host buffer
  - Receiver: Copy data to device
MPI + CUDA is difficult: Portability

- Technology is moving forward quickly
  - Different compute capability generations
  - Different levels of support for GPUDirect v1 and v2
  - New MPI libraries with CUDA support are emerging
- Machines differ from each other
  - Number of GPUs and CPUs per node differ
    - 1 GPU per 2 processors to 8 GPUs per 2 processors
  - Selecting active GPU on multi GPU nodes
MPI + CUDA is difficult: Scalability

- More challenging than for a traditional CPU machine
  - Higher computational power per node
  - Higher latency for sending messages from one GPU to another one
- Efficient device-to-device communication is non-trivial
  - Hiding latencies through overlapping computation & communication
  - Pipelined sends and receives
MPI + CUDA is difficult: Scalability

- New developments will make scaling easier
  - Supercomputers with custom interconnects and system software: Cray XK6 with Gemini interconnect
  - GPUDirect v1: Avoids extra data copy in host memory when combining MPI + CUDA
  - GPUDirect v2: Efficient multi-GPU programming using threads; direct inter-node GPU-GPU transfers
  - Direct inter-node GPU-to-GPU transfers are under development
  - Work on GPU aware MPI implementations is ongoing
MPI + CUDA strategies

1. One MPI process per GPU
   - GPU handling is straightforward
   - Wastes the other cores of the processor

2. Many MPI processes per GPU, only one uses it
   - Poses difficult load balancing problems
MPI + CUDA strategies

3. Many MPI processes share a GPU
   - Two processes cannot share the same GPU context, per process memory on GPU
   - Sharing may not always be possible
     - Limited memory on GPU
     - If GPUs are in exclusive mode
   - In some cases this strategy is more efficient as the idle time of the GPU decreases (e.g. NAMD)
Selecting GPU

- To select a GPU for each process one needs to know which ranks are on the same node

- Simple approach
  - You may assume MPI processes are assigned in a linear fashion to the cores
    - Node 1 ranks: 0 1 2 3 4 5
    - Node 2 ranks: 6 7 8 9 10 11 ....
  - You may also assume you know the number of processes per node to compute rank on node
    - int nodeRank=rank%processesPerNode
Selecting GPU

- **The simple approach is not very robust as it assumes the ranks have been placed in a specific fashion**

- **Advanced approach**
  - Get name of node using `MPI_Get_processor_name()`
  - Compute a hash (integer) value from the name
    - See google, e.g., [http://www.cse.yorku.ca/~oz/hash.html](http://www.cse.yorku.ca/~oz/hash.html)
  - Create a intra-node communicator by using the hash value as the color for `MPI_Comm_split()`
  - Get `nodeRank` and `processesPerNode` from the new inter-node communicator using `MPI_Comm_rank()` and `MPI_Comm_size()`
Selecting GPUs

• **The number of active GPUs visible to the rank is**
  • `cudaGetDeviceCount(&deviceCount);`

• **Divide GPUs to processes (strategy 3)**
  ```
  id = nodeRank % deviceCount;
  cudaSetDevice(id);
  ```

• **One GPU per process (strategy 1)**
  ```
  if(processesPerNode == deviceCount){
      id = nodeRank % deviceCount;
      cudaSetDevice(id);
  }
  else //ERROR
  ```
MPI send & receive: Simple version

- **Time for one message is**
  - \( T_{\text{MPI}}(\text{size}) + 2 \ T_{\text{CUDA}}(\text{size}) \)
  - \( T_{\text{CUDA}} \) comparable to MPI (GB/s of transfer speed)
- **Simple to implement, but performance is not good for larger messages**

```c
if (rank==0){
  cudaMemcpy(hBuffer,dBuffer,size,cudaMemcpyDeviceToHost);
  MPI_Send(hBuffer,size,MPI_BYTE,1,100,MPI_COMM_WORLD);
}
else if (rank==1){
  MPI_Recv(hBuffer,size,MPI_BYTE,
           0,100,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  cudaMemcpy(dBuffer,hBuffer,size,cudaMemcpyHostToDevice);
}
```
MPI Send & receive: pipelined

- The message is divided into chunks
- Using streams one can transfer data from GPU one chunk at a time using cudaMemcpyAsync
- Each chunk can be sent using non-blocking MPI_Isend once it has arrived into CPU memory
- At the receiver side the corresponding operation is performed
- Time is reduced as MPI and CUDA transfers are overlapped
- Requires platform dependent tuning of the chunksize, order of magnitude is 100KB
MPI + OPENMP + CUDA
Introduction

• Each MPI process launches host level threads using, e.g., OpenMP

• Can match one MPI process per GPU and still use all cores

• Since CUDA 4 threads of the same process can share a GPU context
  • Can use the same GPU buffers
  • Done simply by selecting the same device ID
MPI + OpenMP + CUDA

- One MPI process per GPU
  - Data parallel: All threads do computation and share the GPU. Since CUDA 4 they can share the same context
  - Task parallel: one handles GPU, one MPI, one IO the rest CPU computation, etc.
- Several GPUs per MPI process
  - Both data and task parallel approaches possible
  - Some performance benefit to be expected from GPU Direct v2, enables direct copies from GPU to GPU
RECENT DEVELOPMENTS
Unified Virtual Address (new in CUDA 4)

No UVA: Multiple Memory Spaces

UVA: Single Address Space

System Memory
GPU0 Memory
GPU1 Memory

System Memory
GPU0 Memory
GPU1 Memory

CPU
GPU0
GPU1

PCI-e

CPU
GPU0
GPU1

PCI-e

0x0000 0xFFFF
0x0000 0xFFFF
0x0000 0xFFFF

0x0000
0xFFFF
0xFFFF

0x0000 0xFFFF
0xFFFF

Ilmatieteen laitos / PowerPoint ohjeistus

23.8.2011
UVA

- Same address space for both GPU and CPU
  - Cannot automatically read data, transfers still needed
- All memory must be allocated through CUDA library
- Enables one to find out where the memory is from the pointer value - easier interfaces possible

Without UVA:
  cudaMemcpy(dArray, hArray, SIZE, cudaMemcpyHostToDevice);

With UVA:
  cudaMemcpy(dArray, hArray, SIZE, cudaMemcpyDefault);
GPU Direct v2 - Peer-to-peer communication

- GPU Direct v2 only available for Tesla Fermi cards and Cuda 4 or later
- Direct access
  - Can directly access memory on another GPU
- Direct transfer
  - Can copy data directly from one to the other
- Works with one MPI process for all GPUs
- Easy to use with UVA
GPU Direct v2 - Peer-to-peer communication

cudaSetDevice(ompThread);
if(ompThread == 0 )
    cudaMalloc((void**)&(dArray1), size);
else if(ompThread == 1 )
    cudaMalloc((void**)&(dArray2), size);
...
if( ompThread==0){
    cudaMemcpy(dArray2, dArray1, size, cudaMemcpyDefault);
    cudaDeviceEnablePeerAccess(1,0);
}
...

- Vuori.csc.fi device-to-device bandwidth
  - With peer-to-peer more than 5GB/s
  - Without less than 4 GB/s
CUDA support in MPI libraries

- MPI libraries where you can directly provide device pointers to the MPI calls
  - MPI_Send(dBuffer, ….)
  - UVA essential for these libraries, enables them to deduce where the buffer is
  - Handle data staging internally

- OpenMPI
  - CUDA support in current trunk version

- MVAPICH2-GPU
  - Will be in future versions
GPUDirect v1

- Infiniband network card and GPU can share the same pinned memory
  - Infiniband uses pinned memory for RDMA transfers
  - CUDA also uses pinned memory for fast DMA transfers to/from card
- Avoids one extra memory copy between GPU and Infiniband pinned memory
- Support by Mellanox and Qlogic
- No changes to user code necessary
GPUDirect v1

Without

With
Summary

• **CUDA+MPI**
  • Naïve approach easy, getting good performance more tricky
  • Selecting GPU on multi-GPU nodes

• **CUDA+OpenMP+MPI**
  • Threads can share same GPU context

• **Recent developments**
  • MPI libraries with CUDA support will simplify things
  • Peer-to-peer access paves way for more efficient on-node multi-GPU parallelisation
references

- Vlasov GPU paper
- CUDA and MPI, Dale Southard Senior Solution Architect NVIDIA
- An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters
- Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark
- An MPI-CUDA implementation of an improved Roe method for two-layer shallow water systems
- GPUs for Molecular Dynamics (MD) Simulations: a user’s perspective
- www2.fz-juelich.de/jsc/files/docs/vortraege/gpu/gpu-5-Multi_GPU.pdf
- people.maths.ox.ac.uk/gilesm/cuda/MultiGPU_Programming.pdf
- http://people.maths.ox.ac.uk/gilesm/cuda/lecs/lec6-2x2.pdf
- www.cse.buffalo.edu/faculty/miller/Courses/CSE710/heavner.pdf