GPU Development and Computing Experiences

Kyle FERNANDES\(^1\) <kj333@cam.ac.uk>

\(^1\) Research Computing Services, University of Cambridge
Research Computing Services
Committed to supporting the University of Cambridge's mission to contribute to society through the pursuit of education, learning and research at the highest international levels of excellence

• Formed when High Performance Computing Services (HPCS) merged with University Information Services (UIS) in early 2014

• Four teams with more than 20 staff
  • Platforms
  • HPC Services
  • Bioinformatics (Genomics England/BRIDGE; OpenCB for Hadoop)
  • Research Software Engineering (RSE)

• Support HPC, big data analytics and virtual services for research
Two major HPC systems

- Darwin: 600 nodes with 2 Sandybridge 8-core sockets (9600 cores)
- Wilkes: 128 nodes with 2 Ivybridge 6-core sockets and 2 NVIDIA Tesla K20 GPUs (1536 cores and 256 GPUs)

Wide portfolio of virtual services and storage services for research

- Virtual services based on OpenStack
- Databases and web portals for researchers

http://www.hpc.cam.ac.uk
West Cambridge Data Centre

- £20m investment
- > 1 MW capacity
- Built by ARUP, managed by UIS

- Spectra Logic Tape Library (eventually 15 PiB)
Research Software Engineering

- Officially established November 2015 (unofficially, before then)
- 6 RSEs with various scientific backgrounds and computing skills
  - CFD, Material Science, Molecular Dynamics, Quantum Chemistry
  - Software Engineering, Parallel Programming, GPU Programming, Container DevOps (new), Machine Learning (new)
- Not subsidised by University chest funding
  - Currently fully self-sustained via internal and external (industrial) projects
  - Programme for Simulation Innovation (5-year research project in collaboration with the EPSRC and Jaguar Land Rover, ending 2018)
  - GPU C.o.E., DELL/Intel Solution Centre, DiRAC and SKA
Research Software Engineering

- RSE paid services currently available to all Schools and Departments
  - Internal consultancy (paid directly via P.O.). 3 to 6 months
  - Dedicated staff (RSE technical support paid via research grant). 1 year minimum
- External consultancy (paid directly via P.O.). Short period at higher daily rate
  - RSE paid services will start to be advertised in 2017 (currently at full capacity)
Cambridge National Tier-2 system: “Peta”

- £5m EPSRC funding and ~ £5m University funding
  - Large fraction open to UK EPSRC researchers
- Focus on data-intensive and data-centric HPC and analytics
- 3 major science areas
  - Computational Engineering and Smart Cities Fluid Dynamics
  - Materials Modelling and Computational Chemistry
  - Health Informatics
- RSE will provide support in collaboration with other Tier-2 centres and RSE National Network
Cambridge National Tier-2 system: “Peta”

• New compute capabilities
  • Targeting sustained 1 PFLOPS (Broadwell?) x86 CPUs
  • Targeting sustained 1 PFLOPS (non-fabric KNL?) x86 many-core CPUs
  • Targeting sustained 1 PFLOPS (PCIe Pascal?) GPUs
  • Multi-tier fast I/O capabilities via co-design project with winning bidder

• Targeting “dense” GPU nodes
  • Up to 4 GPUs able to communicate via P2P

• Interconnect will likely be a mixture of (Mellanox IB? and OPA?) technologies
Acknowledgements

• Filippo Spiga
Multi-GPU Collective Communication
Multiple GPUs

• GPU compute nodes tend to contain 4 or more GPUs
  • In principle, good multi-GPU scaling

• Scaling can be poor if large amounts of data needs to be transferred between GPUs
  • Results in significant amount of time being spent on communicating rather than computing
Collective Communication

- Collective communication patterns are quite common
  - Involve multiple senders and/or receivers
  - Patterns include broadcast, scatter, gather, reduce, etc.
Collective Communication

• Collective communication patterns are quite common
  • Involve multiple senders and/or receivers
  • Patterns include broadcast, scatter, gather, reduce, etc.

• Non-collective communication alone can be inefficient and multiple senders and/or receivers compound inefficiencies

• Multi-GPU collective communications difficult to implement efficiently
  • Usually require topology-aware implementation for good performance
Multi-GPU Topologies

- CPU0
  - Switch
    - GPU0
    - GPU1
  - Switch
    - GPU2
    - GPU3

PCIe
Multi-GPU Topologies

CPU0

Switch

GPU0  GPU1  GPU2  GPU3

Switch

CPU0

Switch

GPU0  GPU1  GPU2  GPU3
Multi-GPU Topologies
Topology-Aware Algorithms

- Collectives often implemented using tree algorithms
Collectives often implemented using tree algorithms
Collectives often implemented using tree algorithms
Collectives often implemented using tree algorithms

Most collectives amenable to bandwidth-optimal implementations on rings

- Many topologies can be interpreted as one or more rings
Topology-Aware Algorithms

- Collectives often implemented using tree algorithms
- Most collectives amenable to bandwidth-optimal implementations on rings
  - Many topologies can be interpreted as one or more rings
NCCL

- Pronounced "Nickel"

Easily integrated, topology-aware library of multi-GPU accelerated collective communication routines to improve scalability of multi-GPU applications

- https://github.com/NVIDIA/nccl
- Implemented by NVIDIA
- Supports an arbitrary number of GPUs installed in a single node
- Can be used in single process, multi-threaded (e.g. OpenMP) or multi-process (e.g. MPI) applications
• Broadcast, all-gather, reduce, all-reduce and reduce-scatter available
• Both out-of-place and in-place available where relevant
• Gather, scatter and all-to-all pending
• Supports char, int, half, float, double, long long and unsigned long long types
• Supports sum, product, minimum and maximum reduction operations
NCCL

- Optimised for throughput
  - Optimised to achieve high bandwidth at low occupancy over PCIe
  - Saturates PCIe 3.0 x16 interconnect using a single block of CUDA threads
  - Bulk of GPU free to execute compute tasks concurrently with communication
NCCL

- Implements ring style collectives
- Implicitly indexes GPUs into optimal ring order
- Provides performance portability
  - Same code gives best performance on machines with different GPU topologies
NCCL

- Best performance achieved when all GPUs located on common PCIe root complex
  - Multi-socket configurations also supported
• Best performance achieved when all GPUs located on common PCIe root complex
  
  • Multi-socket configurations also supported
NCCL

- Best performance achieved when all GPUs located on common PCIe root complex
  - Multi-socket configurations also supported
NCCL

• Follows MPI API fairly closely
  • Not MPI standard compliant

```c
int MPI_Reduce_scatter(
    void *sendbuf,
    void *recvbuf,
    int recvcounts[],
    MPI_Datatype datatype,
    MPI_Op op,
    MPI_Comm comm);
```

```c
ncclResult_t ncclReduceScatter(
    void *sendbuf,
    void *recvbuf,
    int recvcount,
    ncclDataType_t datatype,
    ncclRedOp_t op,
    ncclComm_t comm,
    cudaStream_t stream);
```
NCCL

- Follows MPI API fairly closely
  - Not MPI standard compliant

```c
int MPI_Allgather(
    void *sendbuf,
    int sendcount,
    MPI_Datatype sendtype,
    void *recvbuf,
    int recvcount,
    MPI_Datatype recvtype,
    MPI_Comm comm);
```

```c
ncclResult_t ncclAllGather(
    void *sendbuf,
    int sendcount,
    ncclDataType_t sendtype,
    void *recvbuf,
    int recvcount,
    MPI_Datatype recvtype,
    ncclComm_t comm,
    cudaStream_t stream);
```
NCCL Fortran Bindings

- NCCL implemented by NVIDIA in C++
- Fortran bindings consisting of ncclFor module and libncclfor library
- C vs Fortran (single process example)

```c
for (i = 0; i < nDev; ++i) {
    stat = cudaSetDevice(devList[i]);
    res = ncclReduce(sendPtr[i], recvPtr[i], nEl, dataType,
                     redOp, root, comm[i], stream[i]);
}

do i = 1, nDev
    stat = cudaSetDevice(devList(i))
    res = ncclReduce(sendPtr(i), recvPtr(i), nEl, dataType, &
                     redOp, root, comm(i), stream(i))
end do
```
NCCL Fortran Bindings

res = ncclReduce(sendPtr(i), recvPtr(i), nE1, dataType, &redOp, root, comm(i), stream(i))

• NCCL Fortran types and functions accessed by using ncclFor module

  use ncclFor

• NCCL Fortran types

  type(ncclResult) :: res
  type(ncclComm) :: comm(nDev)
  type(ncclDataType), parameter :: datatype = ncclFloat
  type(ncclRedOp), parameter :: redOp = ncclProd
NCCL Fortran Bindings

res = ncclReduce(sendPtr(i), recvPtr(i), nEl, dataType, &redOp, root, comm(i), stream(i))

• Communicator object created for each GPU
  • Identifies set of GPUs that will communicate and maps communication paths between them
    
    res = ncclCommInitAll(comm(:), nDev, devList(:))

• Communicator objects can be destroyed when no longer required
  
  do i = 1, nDev
    call ncclCommDestroy(comm(i))
  end do
NCCL Fortran Bindings

res = ncclReduce(sendPtr(i), recvPtr(i), nEl, dataType, &
redOp, root, comm(i), stream(i))

• NCCL collectives expect pointers to device arrays

  real(real32), allocatable, device :: sendArr(:)
  type(c_devptr)                        :: sendPtr(nDev)

• Device arrays need to be allocated on all participating devices

• Pointers need to be saved within scope of device array allocation

  do i = 1, nDev
    stat = cudaSetDevice(devList(i))
    allocate(sendArr(nEl))
    sendPtr(i) = c_devloc(sendArr(:))
    sendArr(:) = hostBuff(:, i)
  end do
res = ncclReduce(sendPtr(i), recvPtr(i), nEl, dataType, &
redOp, root, comm(i), stream(i))

- Device arrays need to be re-associated in order to use them directly

  do i = 1, nDev
    stat = cudaSetDevice(devList(i))
    call c_f_pointer(recvPtr(i), recvArr(:), [nEl])
    hostBuff(:, i) = recvArr(:)
    deallocate(recvArr(:))
  end do
Resources/Acknowledgements

- https://github.com/NVIDIA/nccl
- Sylvain Jeaugey (NVIDIA)
- Brent Leback (PGI/NVIDIA)
Quantum Supercharger Library
QSL

- Accelerates Gaussian type orbital (GTO) Hartree-Fock (HF - for now) packages
  - ~ 40 packages that do HF with GTO
    - Gaussian, NWChem, GAMESS-US, GAMESS-UK, ORCA, etc.
  - Significant effort to learn, trust and use new package
    - Everything (input files, workflows, “answers” etc.) is the same, just faster
HF Method

1. Guess $D_0$
2. Calculate $D_1$
3. Form $F_0$
4. Converged? (yes/no)
   - yes: Calculate Properties
   - no: Form $F_n$
5. Converged? (yes/no)
   - yes: Calculate Properties
   - no: Calculate $D_{n+1}$
6. $2e$
7. SCF
Performance

- Vinblastine (medium-small, 117 atoms, 647 GTOs, 123 MiB Max GPU memory)

Intel Xeon
E5-2698 v3
3.6 GHz
Haswell

NVIDIA Tesla
K40
875 MHz
GK110B (cc35)

NVIDIA Tesla
P100-PCIe-16GB
1189 MHz
GP100 (cc60)
Performance

- Vinblastine (medium-small, 117 atoms, 647 GTOs, 123 MiB Max GPU memory)

GUS vs GUK (CPU)

GUS vs GUK (GPU)
Performance

- Vinblastine (medium-small, 117 atoms, 647 GTOs, 123 MiB Max GPU memory)
Resources/Acknowledgements


• South African Research Chairs Initiative (SARChI)

• NVIDIA Professor Partnership Program
Thank you