Lessons learnt using GPU Direct over RDMA

Filippo SPIGA[1,2] <fs395@cam.ac.uk>

[1] Head Research Software Engineering, Research Computing Services, University of Cambridge
[2] Quantum ESPRESSO Foundation

Wilkes

- **(Still) Biggest GPU system in UK public sector**
- 3631.70 MFlops/W, #2 Green500, June '14
- 128 nodes DELL T620
  - two Intel Ivy Bridge E5-2630 v2 @ 2.60GHz (12 cores in total)
  - two Mellanox Connect-IB
  - two NVIDIA K20c
- (Still) Flexible and effective Testing & Development platform!

Programming heterogeneous systems

- Many GPU-based flagship systems of the near future will exhibit fat GPU node architectures
- Established hybrid programming model MPI+CUDA will continue to exist
- CPUs are getting faster due to larger caches and wider vector registers. These strengths that should not be neglected, it still has a role to play
- CPU can be still used as a processing unit in addition to the GPU, resulting in increased performance
- Overlapping the various data exchanges with computation is a key component in developing efficient and scalable implementations

Case study: 7-point 3D laplace stencil code

- Well-known case study (plus many scientific codes are known to be memory bound!)
- Simple model suggest that a reasonable workload division ratio between the CPU and the GPU
- Simple application with high communication-to-computation ratio makes it well suited for investigating the effect of different data movement strategies

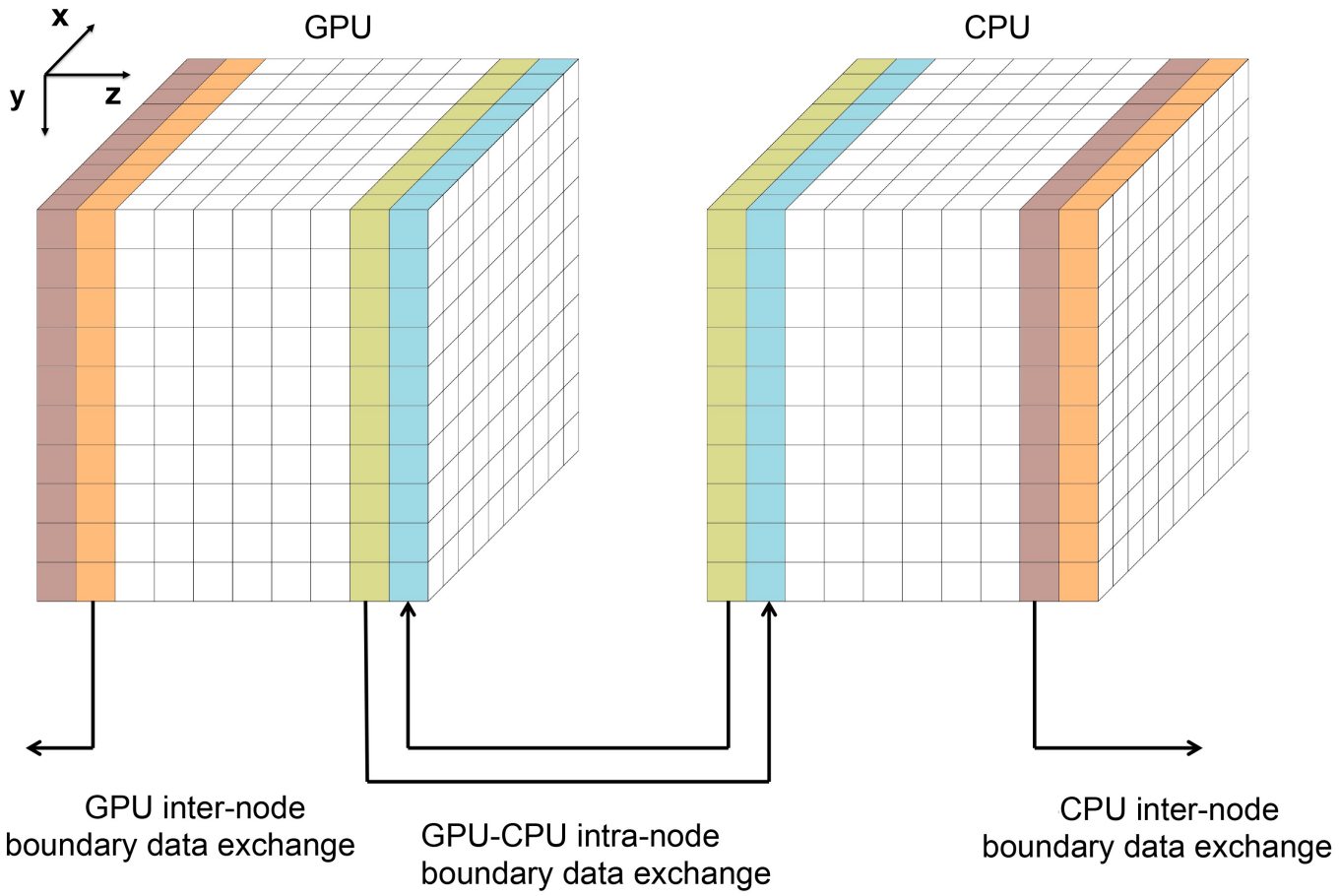
Objective:

- Investigate whether CPU-avoiding GPU ↔ GPU inter-node transfers can alleviate the pressure on the CPU in a concurrent CPU+GPU implementation
- Quantify the impact of CPU-avoiding inter-node GPU↔GPU transfers using GPUDirect RDMA

Implementation

- Sub-domains equal to number of GPU, each sub-domain is assigned to a MPI process
- Computing workload ration GPU/CPU (no silver-bullet but we can get close)
- Overhead of various intra-node and inter-node data exchanges should be masked
  - a *master* thread is dedicated to handling all the MPI calls, which may concern both inter-node CPU ↔ CPU and inter-node GPU ↔ GPU data exchanges (non-blocking fashion). Also responsible for intra-node CPU ↔ GPU data exchanges (async memcpy)
  - The remaining threads computes CPU's halo boundaries, such that these can be finished as soon as possible, consequently initiating the CPU ↔ CPU and CPU ↔ GPU data exchanges

Workload division within a MPI domain



Optimization details

CPU

- Pencil-shaped cache blocking along the Y direction, in combination with non-temporal store instructions

GPU

- Good single-GPU performance (78% of peak attainable) by using pipelined wave-front technique which introduces a *for*-loop to compute values in Z direction column-wise
- The performance was further enhanced using the GPU's read-only cache and GPUs constant memory

MPI

- Computation and communication are overlapped since the computation of interior points is decoupled from the computation of the halo boundary points

Experiments setup

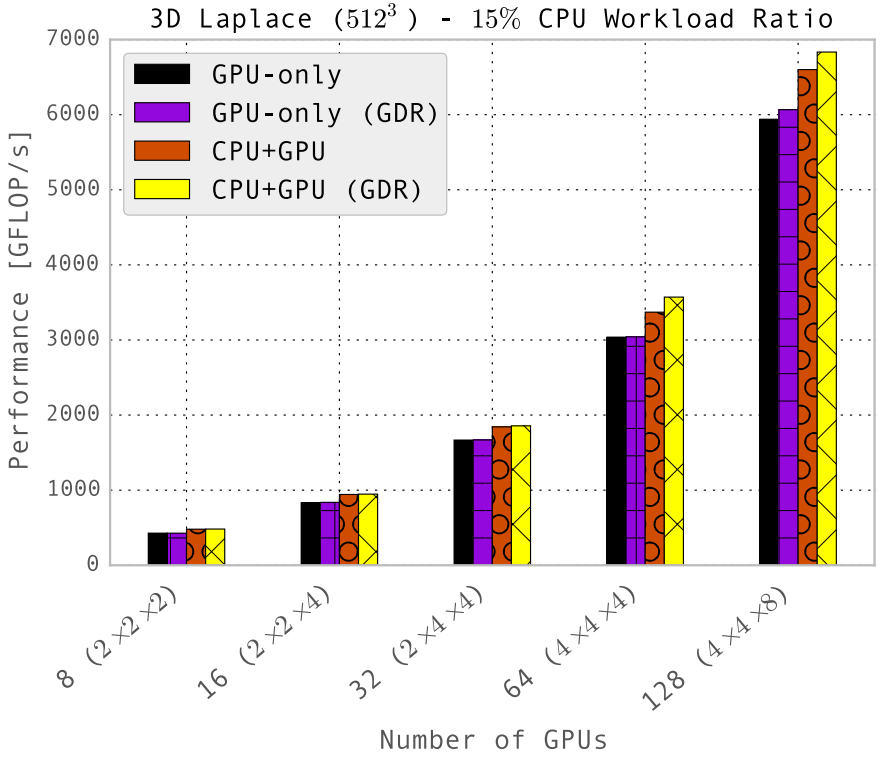
SW stack

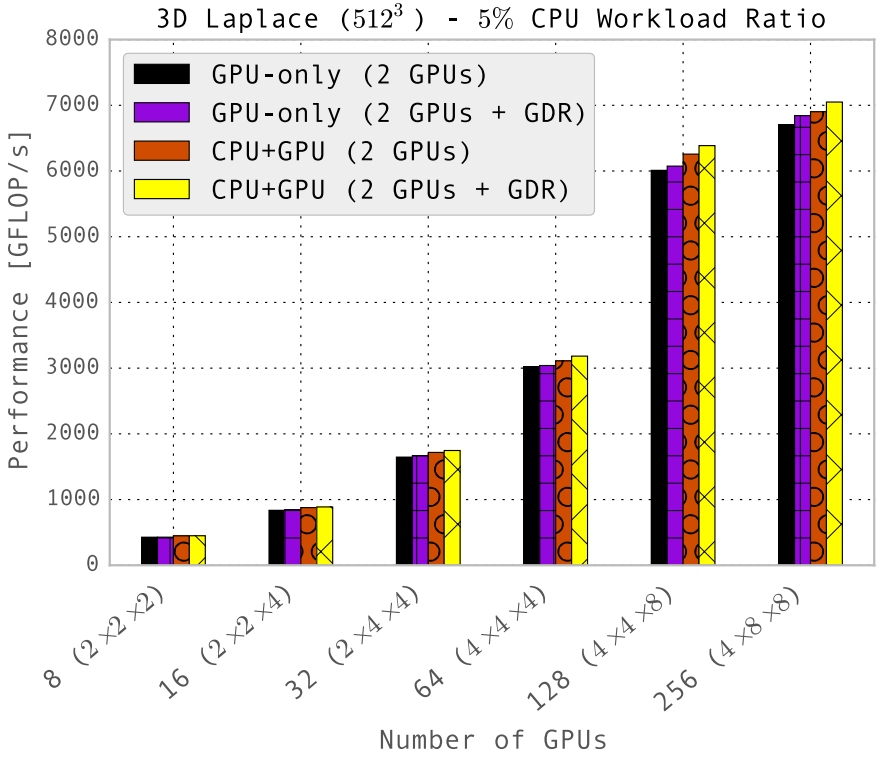
- MVAPICH2-GDR v2.2
- Intel 15
- CUDA SDK 7.0
- MOFED 2.4 + GDR libs

Experiments performed

- Weak scaling (fixed the sub-domain problem size at 512 × 512 × 512), one GPU and two GPU per node using different CPU workload ratio (15% and 5%)
- Strong scaling (overall problem size was set to 512 × 512 × 1024), one GPU and two GPU per node using best workload ratio measured

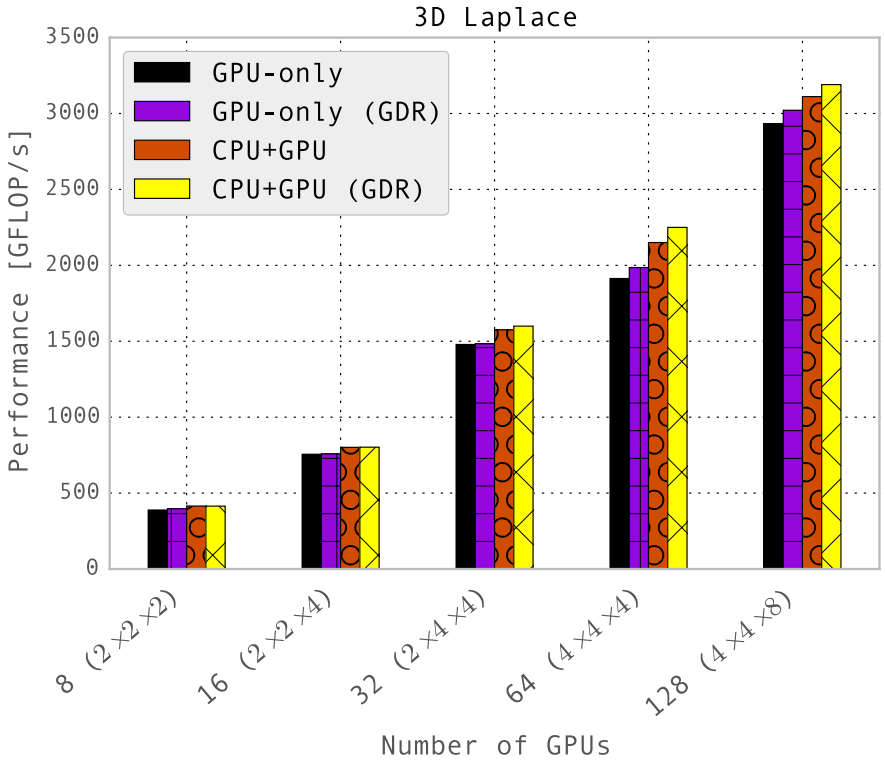
Weak Scaling

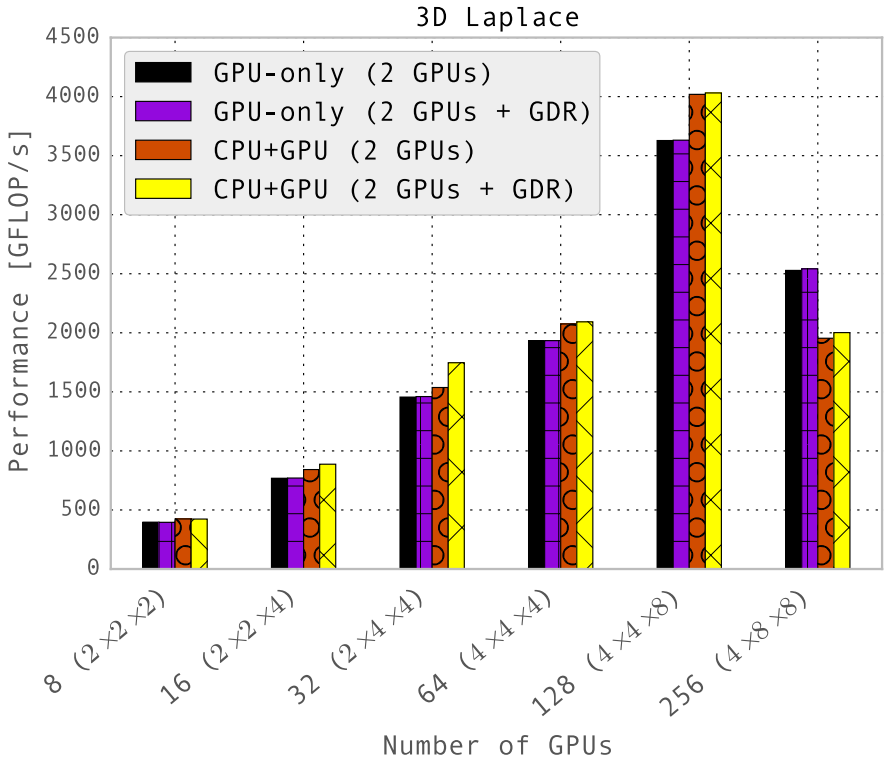




"GPU-generated" MPI message size under our weak scaling experiments is approximately 2MB. The chunking of the messages (for pipelining) also puts some load on the CPU.

Strong scaling





Impact of GDR is on average bigger compared to the weak scaling experiments most likely linked to the fact that the MPI messages become smaller. Intra-node communication plays a more important role as opposed to when only one GPU per node is used.

Personal takeaways …

- GDR is well-suited in applications where latency is crucial or for strong scaling with small problem sizes
- Even at large message sizes (e.g. 2 Mbytes), MVAPICH implementation handles better GPU ↔ GPU transfers than explicit programming
- CUDA-aware MPI programming needs less code
  - 737 originally, with GDR 643 → reduction of 12.75%
- GDR performance sometimes are not exceptional but it is *always* worth using it
- GDR frees the CPU to do something else… but it is not perfect

GDR-as-a-service

Commitments

- Provide latest cutting-edge CUDA software stack
- Provide set of script to handle GPU binding and dual-rail binding for various MPI flavors
- Provide optimal pre-defined environment settings

Challenges

- I need to bother Khaled every time I need a new build of GDR ☺
- Make sure people are using the latest and use scripts correctly

Beyond Wilkes

Cambridge HPC is a Tier-2 facility

- Explosion of new users in the *Long Tail of Science*
- Diverse work-loads (non-parallel single/multi GPU, ML/DL)
- Scaling is overrated

Wilkes 2.0 (in progress)

- NVIDIA Pascal GPU on PCIe, GPU:CPU 4:1 (likely) or 2:2 design
- Mellanox EDR and GDR-enabled node design
- 3x current (sustained) performance target

Acknowledgments

- **Dr Mohammed Sourouri**, NTNU and Simula Research Lab (Norway)
- MVAPICH team
- NVIDIA “GDR” team
- Mellanox “HPC R&D” team

THANK YOU FOR YOU ATTENTION

CPU+GPU Programming of Stencil Computations for Resource-Efficient Use of GPU Clusters
M. Sourouri, J. Langguth, F. Spiga, S. B. Baden, X. Cai
2015 IEEE 18th International Conference on Computational Science and Engineering (CSE'15)

On hybrid MPI+CUDA+OpenMP programming and concurrent CPU+GPU computations
M. Sourouri, J. Langguth, F. Spiga, S. B. Baden, X. Cai
Concurrency and Computation: Practice and Experience (submitted)