Optimizing Amber for Device-to-Device GPU Communication

Samuel Khuvis
skhuvis@osc.edu
Ohio Supercomputer Center
Columbus, Ohio, USA

Karen Tomko
ktomko@osc.edu
Ohio Supercomputer Center
Columbus, Ohio, USA

Scott R. Brozell
srb@osc.edu
Ohio Supercomputer Center
Columbus, Ohio, USA

Chen-Chun Chen
chen.10252@buckeyemail.osu.edu
Ohio State University
Columbus, Ohio, USA

Hari Subramoni
subramon@cse.ohio-state.edu
Ohio State University
Columbus, Ohio, USA

Dhabaleswar K. Panda
panda@cse.ohio-state.edu
Ohio State University
Columbus, Ohio, USA

ABSTRACT

Although direct GPU-to-GPU communication has been possible in MPI libraries for over a decade, the limited availability of compatible hardware at academic HPC centers has discouraged the development of algorithms in scientific applications that take advantage of this capability. In this paper, we take Amber, a molecular dynamics code used to simulate proteins and nucleic acids, as a test case. We demonstrate the modifications necessary to implement GPU-to-GPU communication. Compared to the previous implementation, these modifications show an average of approximately 36% improvement in performance overall and 84% for the important explicit solvent subset of the benchmarks.

CCS CONCEPTS

• **Computing methodologies** → **Parallel computing methodologies**.

KEYWORDS

MPI, GPUs, Amber

ACM Reference Format:
Samuel Khuvis, Karen Tomko, Scott R. Brozell, Chen-Chun Chen, Hari Subramoni, and Dhabaleswar K. Panda. 2023. Optimizing Amber for Device-to-Device GPU Communication. In *Practice and Experience in Advanced Research Computing (PEARC '23), July 23–27, 2023, Portland, OR, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3569951.3597553

1 INTRODUCTION

AMBER is a widely-used software suite for molecular dynamics (MD) simulations of proteins and nucleic acids. It has been in active development since 1975 [2]. As recently as 2019 AMBER had more modeling papers citing it than any of the other popular biomolecular software packages [12, Figure 2e]. Several solvation models are supported including explicit solvent implemented with particle-mesh Ewald (PME) and various Generalized Born (GB) implicit solvent approaches. AMBER added GPU support in version 11 [4] and has continued to improve its performance and capabilities [7, 11] in later releases. The initial GPU implementation targeted NVIDIA GPUs and computed most of the MD algorithm on the GPU. Later updates added additional features, such as implementation of sophisticated free energy methods on the GPU in Amber18 and support for AMD GPUs in Amber22. When we refer to Amber in this paper we mean the main CUDA MD program of AMBER version 20.

Traditionally, multi-GPU support has been achieved with MPI by passing data from the GPU buffer to host buffer, performing MPI communication, and then passing data back to GPU buffers. The capability to specify device buffers in MPI calls was added to MVAPICH2 in 2011 and to HPC-X OpenMPI in v1.7.0 in 2013 [14]. If appropriate hardware and device software is available on the system, then communication is performed directly between GPUs without transfer back to hosts. If not available, then the library will perform inter-process communication via host-to-device and device-to-host communication internally, although the additional data copying will have some impact on performance. Although this technology was first presented in 2011 [5], the hardware capabilities have only started to become available at most academic HPC centers in the last few years. As a result, many scientific applications have been slow to take advantage of these capabilities. For instance, GROMACS, another MD application, only added support for direct GPU-to-GPU communication in its 2022 release [6].

In this paper, we discuss the modifications required to enable device-to-device GPU communication in Amber20 and demonstrate performance benefits on a set of benchmark runs.

2 EXPERIMENTAL SETUP

All results in this paper were performed on the following benchmarks in the Amber20 Benchmark Suite to better understand the performance bottlenecks of typical use cases:

- Particle-mesh Ewald (PME):
  - Cellulose production (NPT and NVE)
  - FactorIX production (NPT and NVE)
  - JAC production (NPT and NVE)
  - STMV production (NPT and NVE)
- Generalized Born (GB):
  - myoglobin
  - nucleosome

where the statistical ensembles NPT and NVE specify the fixed quantities: Namely, the total number of particles (N) is constant in

PEARC '23, July 23–27, 2023, Portland, OR, USA

ACM ISBN 978-1-4503-9985-2/23/07…$15.00
https://doi.org/10.1145/3569951.3597553

both, and in NPT pressure (P) and temperature (T) are conserved quantities, whereas total energy (E) and volume (V) are conserved in NVE. The computational cost of pressure and temperature control in NPT MD is nonzero but modest in comparison to the algorithmically more straightforward NVE MD [1]. Both ensembles are important for biomolecular simulations [10].

Experiments were performed on the Pitzer system at OSC [3]. Tests were performed on 48-core Cascade Lake nodes each with 2 NVIDIA V100 GPUs.

Tests were performed with three MPI implementations:

(1) MVAPICH2 2.3.6 [9]
(2) MVAPICH2-GDR 2.3.7
(3) HPC-X OpenMPI 3.1.6

Note that MVAPICH2-GDR and HPC-X are both CUDA-aware MPI implementations, meaning that GPU buffers can be passed directly to MPI functions, while MVAPICH2 is not CUDA-aware.

3 PERFORMANCE ANALYSIS

To better understand the performance characteristics of Amber, we began by profiling a run of Amber with, JAC production (NPT), one of the benchmarks in the Amber20 Benchmark Suite, on 2 nodes with 1 process and 1 GPU per node.

Profiling with ARM MAP [8] shows that a large portion of time was being spent in MPI communication and device-to-host communication. Figure 1 shows that the most expensive functions for this benchmark are `MPI_Allreduce` and `cudaMemcpy`. Figure 1 (b) shows the call stack; we can see that these functions are being called from the `gpu_allreduce` function.

Listing 1 shows pseudocode for the relevant part of the `gpu_allreduce` function. The allreduce is performed in three steps. The data is initially on the device, so it is copied to the host. Then, an allreduce is performed among all of the host devices. Once it has completed, the data is copied back to the device.

Listing 1: gpu_allreduce pseudocode

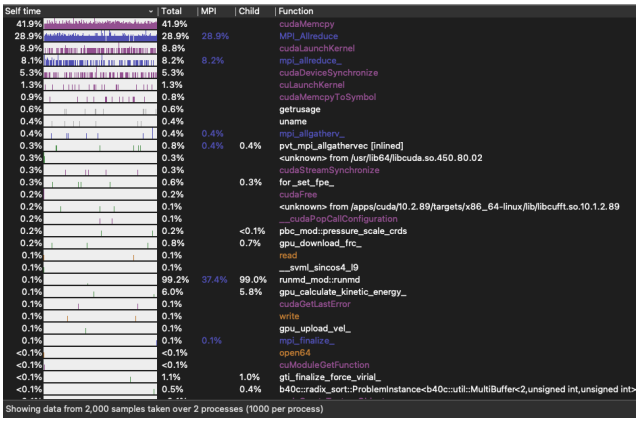
```
cudaMemcpy Device-to-Host
MPI_Allreduce Host-to-Host
cudaMemcpy Host-to-Device
```

4 MODIFICATIONS

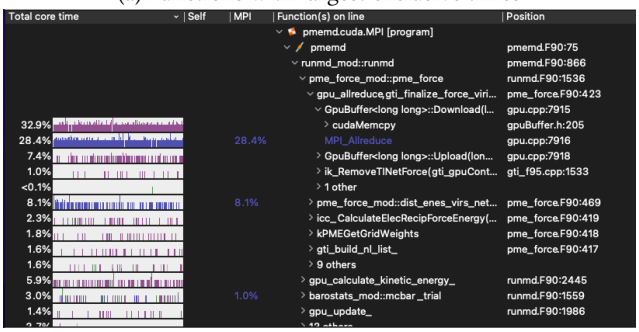
As discussed in Section 3, the most expensive routines in our testing were the Host ↔ Device communication and `MPI_Allreduce` called from `gpu_allreduce`.

Since the data is stored on the device, we can reduce the time spent in `gpu_allreduce`, by communicating directly between devices on different hosts, rather than by copying from device to host, then host to host, and back to device.

Listing 2 shows the modified algorithm for `gpu_allreduce`. First, we ensure that all of the devices have reached the same point in the code. Then, we perform an allreduce directly between buffers on the devices. Finally, Amber requires that the buffer be up-to-date on the host so we still perform the copy from the device to the host. This modified algorithm reduces the amount of data that needs to be transferred between the device and the host.



(a) Functions with largest exclusive times



(b) Call stack

Figure 1: MAP profile of JAC_production_NPT_4fs benchmark on 2 nodes, 1 process per node.

Listing 2: Modified gpu_allreduce pseudocode

```
cudaDeviceSynchronize
MPI_Allreduce Device-to-Device
cudaMemcpy Device-to-Host
```

The algorithmic modification is straightforward and, in fact, involved only five lines of Amber C++ code. To put this in context, the concomitant changes for the program output and the installation recipe, although conceptually trivial in comparison, were three times the size in Fortran90 and CMake code. We reiterate that the initial Amber GPU program and the direct GPU-to-GPU communication technology were both circa 2011. However marrying these capabilities was not achieved until this framework provided the software middleware to facilitate the necessary algorithmic revision. Furthermore, it is thus available for MVAPICH2-GDR, HPC-X OpenMPI, and future MPIs.

5 RESULTS

We expect that the use of a CUDA-aware MPI implementation can reduce the time spent in Amber by reducing the amount of host ↔ device communication.

Figures 2–6 show the performance of each of the benchmarks from Section 2. For each benchmark, throughput is given in nanoseconds of MD simulation time per day of wallclock time (ns/day) on

2, 4, and 8 nodes. For each of these configurations, we show results for MVAPICH2 with the original code, and then HPC-X OpenMPI, MVAPICH2, and MVAPICH2-GDR with the modified code.

We find that for most benchmarks MVAPICH2 gives the lowest throughput, but optimal throughput is achieved with MVAPICH2-GDR. For instance, Figure 2 (a) shows performance of the Cellulose production (NPT) benchmark. Running on 8 Cascade Lake nodes, MVAPICH2 gives a throughput of 28.51 ns/day with the original code and 24.92 ns/day with the modified code. Using the CUDA-aware MPI implementations gives an increased throughput of 52.4 ns/day with HPC-X OpenMPI and 64.28 ns/day with MVAPICH2-GDR. As expected, Figures 2–5 show slightly higher throughput for NVE relative to NPT.

To summarize our results, below is the mean throughput across all the benchmarks for each implementation:

- MVAPICH2 (original code): 192.5 ns/day
- MVAPICH2 (modified code): 201.4 ns/day
- HPC-X OpenMPI (modified code): 234.7 ns/day
- MVAPICH2-GDR (modified code): 262.9 ns/day

On average across all benchmarks we see a ~36% improvement in performance from the original code to the modified code with device-to-device communication using MVAPICH2-GDR.

In particular, we see a large improvement in the eight PME benchmarks where the average throughput for each implementation is:

- MVAPICH2 (original code): 118.2 ns/day
- MVAPICH2 (modified code): 120.3 ns/day
- HPC-X OpenMPI (modified code): 197.2 ns/day
- MVAPICH2-GDR (modified code): 217.8 ns/day

The average performance improvement from the original code with MVAPICH2 to the modified code with MVAPICH2-GDR across all the PME benchmarks was ~84%.

As is evident from the Figures, Amber does not scale well. All the PME results show either no significant performance gain or some loss. Only for the modestly sized, 25095 atoms, GB nucleosome benchmark is the scaling greater than one; see Figure 6 (b). And for the small, 2492 atoms, GB myoglobin benchmark, the scaling is significantly less than one. Although Amber recommends using multiple GPUs only for enhanced sampling (e.g., replica exchange where each GPU in a multiple GPU job executes a single computationally independent simulation from a pool of methodologically coupled simulations), many other MD techniques would benefit from a scalable multi-GPU capability, such as long time-scale MD, free energy calculations (a field with especially intense activity in the Amber community [13]), conformational sampling, and drug discovery in general. The poor scaling is partly a consequence of the limited availability of both hardware and software that supports the development of multi-GPU algorithms as presented and discussed in Section 4, in particular this framework.

6 CONCLUSIONS

By profiling Amber20 benchmarks, we found that the most expensive functions in the main CUDA molecular dynamics program of the AMBER software suite were `MPI_Allreduce` and `cudaMemcpy` called from `gpu_allreduce`.

Modifying the `gpu_allreduce` function so that `MPI_allreduce` communicates directly between GPU buffers reduces the amount of host ↔ device communication with CUDA-aware MPI implementations.

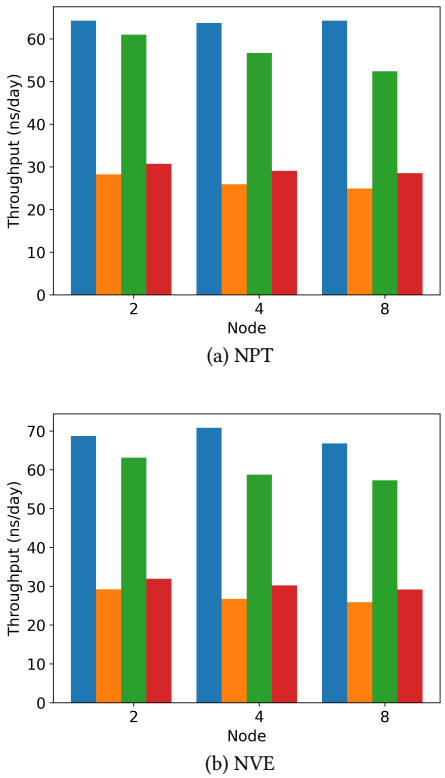


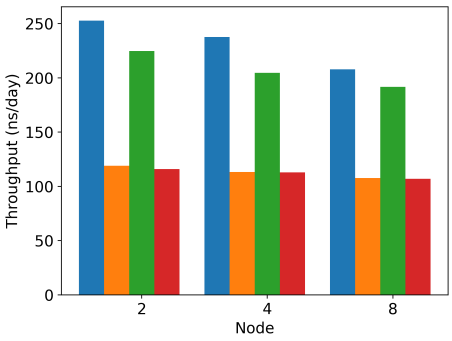
Figure 2: Performance comparison of Cellulose production benchmarks.
MV-GDR (modified code) MV2 (modified code) HPC-X (modified code) MV2 (original code)

As we observed in Section 5, modifying the `gpu_allreduce` and using MVAPICH2-GDR, increases the throughput of Amber by an average of approximately 36% over the whole set of ten benchmarks and by 84% for the PME subset of eight benchmarks.

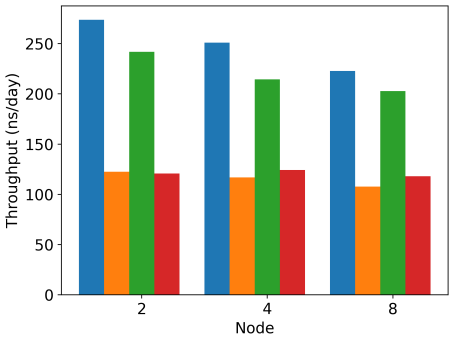
The algorithmic modification involved five lines of application code. This underscores the power of frameworks and the collaborative approach: a substantial performance improvement was enabled via the layered paradigm and implemented with a surgical procedure.

ACKNOWLEDGMENTS

The authors would like to thank Amber developer Thomas E. Cheatham, III for his willingness to collaborate on the project. This work is supported by the National Science Foundation grant #1931537.

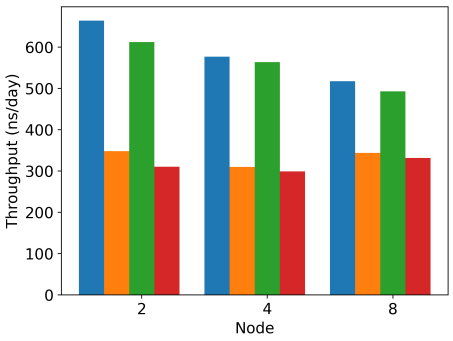


(a) NPT

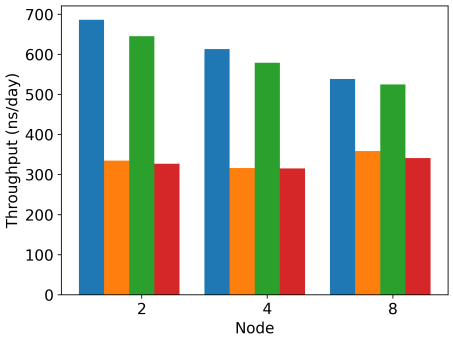


(b) NVE

Figure 3: Performance comparison of FactorIX production benchmarks.
MV-GDR (modified code) MV2 (modified code) HPC-X (modified code) MV2 (original code)

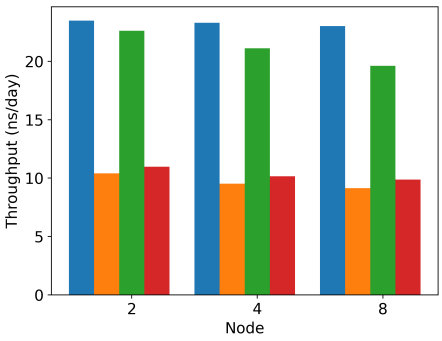


(a) NPT

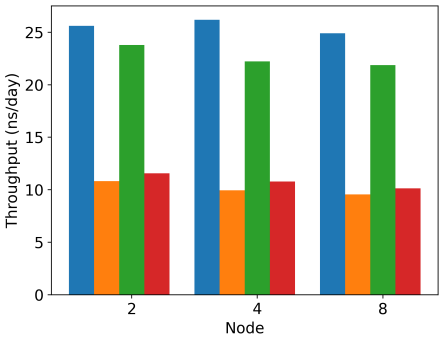


(b) NVE

Figure 4: Performance comparison of JAC production benchmarks.
MV-GDR (modified code) MV2 (modified code) HPC-X (modified code) MV2 (original code)

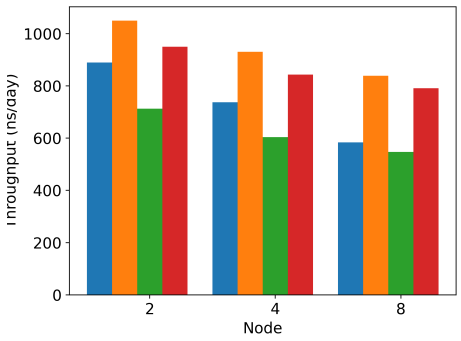


(a) NPT

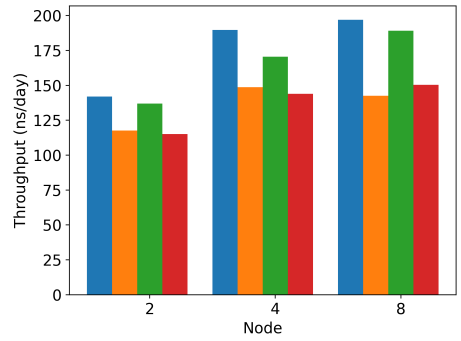


(b) NVE

Figure 5: Performance comparison of STMV production benchmarks.
MV-GDR (modified code) MV2 (modified code) HPC-X (modified code) MV2 (original code)



(a) myoglobin



(b) nucleosome

Figure 6: Performance comparison of GB benchmarks.
MV-GDR (modified code) MV2 (modified code) HPC-X (modified code) MV2 (original code)

REFERENCES

[1] M. P. Allen and D. J. Tildesley. 1987. *Computer Simulation of Liquids.* Clarendon Press, Oxford.

[2] D.A. Case, H.M. Aktulga, K. Belfon, I.Y. Ben-Shalom, J.T. Berryman, S.R. Brozell, D.S. Cerutti, T.E. Cheatham, III, G.A. Cisneros, V.W.D. Cruzeiro, T.A. Darden, R.E. Duke, G. Giambasu, M.K. Gilson, H. Gohlke, A.W. Goetz, R. Harris, S. Izadi, S.A. Izmailov, K. Kasavajhala, M.C. Kaymak, E. King, A. Kovalenko, T. Kurtzman, T.S. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, M. Machado, V. Man, M. Manathunga, K.M. Merz, Y. Miao, O. Mikhailovskii, G. Monard, H. Nguyen, K.A. O'Hearn, A. Onufriev, F. Pan, S. Pantano, R. Qi, A. Rahnamoun, D.R. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, A. Shajan, J. Shen, C.L. Simmerling, N.R. Skrynnikov, J. Smith, J. Swails, R.C. Walker, J. Wang, J. Wang, H. Wei, R.M. Wolf, X. Wu, Y. Xiong, Y. Xue, D.M. York, S. Zhao, and P.A. Kollman. 2020. Amber 2020. https://ambermd.org. University of California, San Francisco.

[3] Ohio Supercomputer Center. 1987. Ohio Supercomputer Center. http://osc.edu/ark:/19495/f5s1ph73

[4] Amber Project Contributors. 2023. *GPU overview and brief history.* Amber Project. Retrieved February 27, 2023 from https://ambermd.org/GPUSupport.php

[5] Paul Stewart Crozier, Gilad Shainer, Ali Ayoub, Pak Lui, Tong Liu, Christian Robert Trott, and Greg Scantlen. 2011. The Development of Mellanox - NVIDIA GPUDirect over InfiniBand - a New Model for GPU to GPU Communications. (January 2011). https://www.osti.gov/biblio/1120826

[6] GROMACS development team. 2022. *Highlights.* GROMACS Project. Retrieved February 27, 2023 from https://manual.gromacs.org/documentation/2022/release-notes/2022/major/highlights.html

[7] Andreas W. Götz, Mark J. Williamson, Dong Xu, Duncan Poole, Scott Le Grand, and Ross C. Walker. 2012. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 1. Generalized Born. *Journal of Chemical Theory and Computation* 8, 5 (2012), 1542–1555. https://doi.org/10.1021/ct200909j arXiv:https://doi.org/10.1021/ct200909j PMID: 22582031.

[8] ARM Ltd. 2022. *ARM MAP.* https://www.linaroforge.com

[9] Dhabaleswar Kumar Panda, Hari Subramoni, Ching-Hsiang Chu, and Mohammadreza Bayatpour. 2021. The MVAPICH project: Transforming research into high-performance MPI library for HPC community. *Journal of Computational Science* 52 (2021), 101208. https://doi.org/10.1016/j.jocs.2020.101208 Case Studies in Translational Computer Science.

[10] Daniel R. Roe and Bernard R. Brooks. 2020. A Protocol for Preparing Explicitly Solvated Systems for Stable Molecular Dynamics Simulations. *The Journal of Chemical Physics* 153 (2020), 054123. https://doi.org/10.1063/5.0013849

[11] Romelia Salomon-Ferrer, Andreas W. Götz, Duncan Poole, Scott Le Grand, and Ross C. Walker. 2013. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald. *Journal of Chemical Theory and Computation* 9, 9 (2013), 3878–3888. https://doi.org/10.1021/ct400314y arXiv:https://doi.org/10.1021/ct400314y PMID: 26592383.

[12] Tamar Schlick, Stephanie Portillo-Ledesma, Christopher G. Myers, Lauren Beljak, Justin Chen, Sami Dakhel, Daniel Darling, Sayak Ghosh, Joseph Hall, Mikaeel Jan, Emily Liang, Sera Saju, Mackenzie Vohr, Chris Wu, Yifan Xu, and Eva Xue. 2021. Biomolecular Modeling and Simulation: A Prospering Multidisciplinary Field. *Annual Review of Biophysics* 50, 1 (2021), 267–301. https://doi.org/10.1146/annurev-biophys-091720-102019 arXiv:https://doi.org/10.1146/annurev-biophys-091720-102019 PMID: 33606945.

[13] Hsu-Chun Tsai, Tai-Sung Lee, Abir Ganguly, Timothy J. Giese, Maximilian CCJC Ebert, Paul Labute, Kenneth M. Jr. Merz, and Darrin M. York. 2023. AMBER Free Energy Tools: A New Framework for the Design of Optimized Alchemical Transformation Pathways. *Journal of Chemical Theory and Computation* 19, 2 (2023), 640–658. https://doi.org/10.1021/acs.jctc.2c00725 arXiv:https://doi.org/10.1021/acs.jctc.2c00725 PMID: 36622640.

[14] Hao Wang, Sreeram Potluri, Miao Luo, Ashish Kumar Singh, Sayantan Sur, and Dhabaleswar K. Panda. 2011. MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters. *Comput Sci Res Dev* 26 (2011), 257–266. https://doi.org/10.1007/s00450-011-0171-3