GPU-centric communication for improved efficiency

Benjamin Klenk
University of Heidelberg
Institute of Computer Engineering
Heidelberg, Germany
klenk@uni-hd.de

Lena Oden
Fraunhofer Institute for Industrial Mathematics
Competence Center High Performance Computing
Kaiserslautern, Germany
oden@fhg.itwm.de

Holger Fröning
University of Heidelberg
Institute of Computer Engineering
Heidelberg, Germany
froening@uni-hd.de

***Abstract*—The rise of GPUs as accelerators is reasoned by the claim for power efficient architectures. Despite their remarkable performance while being still power efficient, the main bottleneck of distributed GPU computing remains communication. A hybrid model is required, because the CPU is used to handle communication. Besides performance aspects, this also adds complexity and makes programming even more difficult. Furthermore this affects energy consumption negatively, as either communication models don't match the GPU architecture, or communication is off-loaded to the CPU which prevents it from entering power saving states.**

We have introduced a GPU-centric communication approach that bypasses the CPU entirely. Communication can be triggered on the GPU and there is no need for returning the control flow to the CPU anymore. In this work, we analyze the impact on energy efficiency for this approach and compare against state-of-the-art communication. We will show that GPU-centric communication avoids not only additional PCIe copies and saves time, also less energy is consumed by saving power during the whole communication process. Energy savings of up to 85 % are possible while bandwidth is increased and latency reduced. Furthermore we analyze the energy impact of barrier synchronization. As we will show, heterogeneity in computation also needs heterogeneity in communication to maintain the high energy efficiency of accelerators.

I. Introduction

Since the end of Dennard scaling and the accompanying consequences, power and energy efficiency has become a key aspect in high performance computing. The claim for power efficient architectures has lead to the arise of GPUs as accelerators, although they were actually built for graphics applications. Languages and programming models like OpenCL, CUDA or directive-based approaches like OpenACC make GPUs available for scientific computing.

Besides technical reasons, energy efficiency is absolutely essential with regard to ecology and economy. For example, one Megawatt costs about 1 million USD per year, thus even for medium scaled systems the operational costs can easily exceed the initial expenditures. Modern CPUs like Intel's Xeon E5-2687W Processor (8 cores, 3.4GHz, AVX) feature a peak performance of about 216 GFLOPs with a specified thermal design power (TDP) of 150W. The power efficiency, as performance divided by power consumption, amounts to 1.44 GFLOPS/W. Comparatively, an Nvidia Kepler K20 GPU achieves about 14.08 GFLOPs/W with 3.52 TFLOPs peak performance and 250W TDP.

978-1-4799-6177-1/14/$31.00 © 2014 IEEE

Apart from energy efficiency, also the performance of GPUs is attractive. Applications with enough parallelism benefit from GPUs as accelerators, however, this mostly applies only if the GPU can keep the data in-core. Due to limited memory capacities, currently not more than 12 GB, working sets often exceed this limit and communication becomes necessary. While computation has become heterogeneous, communication is still homogeneous and entirely handled by the CPU. Besides performance aspects like additional PCIe copies, also the programming effort is increased and hybrid programming models become mandatory.

In former work [1], we introduced direct and GPU-centric communication models, so that GPUs can handle communication. This avoids context switches and saves time and power by bypassing the CPU. In this work, we analyze the impact of GPU-centric communication on energy consumption and compare results to state-of-the-art communication methods.

Most methods report energy consumption for communication in pJ/bit, a metric that describes the energy efficiency of serializers and deserializers. Serialization is pervasively used in modern high performance communication to address the limited pin count. What this metric does not cover, are other units involved in sourcing and sinking traffic, for instance communication engines, on- or off-chip memory, or even processors controlling the communication. However, we believe this to be at least equally important, as (at least today) the larger fraction of energy is consumed by the controlling logic and not the data transport itself. Even with a growing fraction of power for serialization in the future, as indicated in [2] [3] [4], a metric that describes the energy for a communication task is mandatory for appropriate optimizations. Thus, we introduce the metric Joules/word, whose naming indicates a higher abstraction. This metric includes all energy required for a certain communication, and is then normalized to the energy per data word to unveil saturation effects respectively inefficiencies due to protocol overheads.

With this work, we provide following contributions:

- We show the concept of GPU-centric communication and highlight the importance of such approaches in today's heterogeneous computing systems.
- A detailed energy analysis is made, based on microbenchmarks to determine bandwidth and latency. We considered communication and synchronization of both CPU and

GPU controlled communication.

- We prove that specialized processors require specialized communication models and methods for performance and energy reasons. Especially energy is a key metric in this work and we show that savings up to 85 % are possible by using GPU-centric communication methods.

The remainder of this work is structured as follows. Section II provides necessary background information on GPU computing and power measurement. This is followed by a review of GPU-centric communication methods. Section IV analyzes energy consumption of a GPU-centric communication method and compares it to state-of-the-art MPI. Next, a discussion section sums up observations and show insights. Related work is shown in section VI while the last section concludes.

II. Background

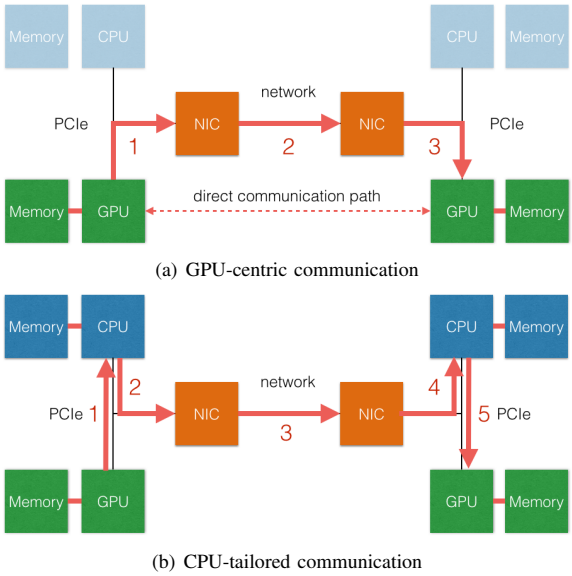
This section focuses on GPU computing, data movements and energy measurement, providing background information for the remainder of this work.

A. GPU Computing

GPUs have experienced a high demand since they stand out in energy efficiency and performance. However, this only applies if applications provide enough parallelism to be exploited by thousands of threads. GPUs are composed of several Streaming Multiprocessors (SMs), each implementing hundreds of so-called CUDA cores. Threads are grouped into blocks, whereby two blocks can be assigned to one SM at the same time. Despite the presence of plenty of cores, not all threads of one block can be executed concurrently, therefore, threads are also grouped into warps, a unit of 32 threads. All threads within a warp follow the same instruction flow, divergences decrease performance and are supposed to be avoided.
Due to the ability to schedule thousands of threads, memory access latencies can be hidden efficiently. This makes large caches unneeded, however, caches are used to reduce memory bus traffic. Furthermore, each SM contains a L1 cache, which is shared with an explicit memory (shared memory) that has to be handled by the programmer. While accesses to the device memory cost about 400-800 cycles, shared memory accesses are extremely fast. Furthermore, all threads within a block can access the memory and communicate with each other. Communication between blocks has to be done using device memory, however, accesses should be coalesced to achieve best memory bandwidth. Synchronization between blocks is not provided, new kernels have to be launched if that becomes necessary. Contrary, threads within blocks can use a hardware-supported barrier. More on CUDA can be found in the excellent book from Nicholas Wilt [5].

B. Power measurement

In general, there are several ways to determine power consumption of applications. First, the power could be measured by external devices and instrument shunts, however, this approach needs special hardware and is cost intensive. Another approach would be to use power models, based on performance counters and linear regression or ANN models [6] [7]. The lack of up-to-date models and less flexibility prohibited the use for this work, though.
In this work, we use a software approach, where power information is gathered by a framework. The Intel Running Average Power Limit (RAPL) [8] registers provide power information for the CPU and DRAM, the same does the Nvidia Management Library (NVML) library for GPUs [9]. Our framework launches an application and periodically polls CPU, DRAM and GPU power consumption during the run time of the application. This approach allows to characterize the power consumption of different components with a sample period of about 200 ms. Intel's RAPL as well as Nvidia's NVML are based on integrated power models. Before applications are launched, the framework measures idle power consumption for 5 seconds. This is necessary to warm up the library that provides power information. The same is done after the application. Therefore, applications should run long enough in order to get accurate results.
Network power consumption is not considered in our framework, since the network consumes 25 W statically, independent on network traffic. Currently, sleep states of network links are not supported due to high transition latencies from active to inactive states and vice versa.



(a) GPU-centric communication

(b) CPU-tailored communication

Fig. 1: GPU-centric communication vs. CPU-tailored communication. The CPU and its memory is inactive for GPU-centric communication during the entire communication process.

III. GPU-centric communication

This section reviews GPU-centric communication approaches. Beginning with state-of-the-art MPI communication for GPUs, also GPU-centric and direct communication methods are presented.

A. State-of-the-art

Most High Performance Computing (HPC) systems are build-up as clusters. Since memory is distributed, communication is required to enable cooperative work of all nodes and processing units. In the past, communication has been tailored for CPUs and communication libraries like MPI have become a standard. This has not changed by the arise of GPUs as accelerators, communication is still handled by the CPU even if computation is offloaded to the GPU. Besides additional latency due to PCIe data copies, this also loads the programmer with additional context switches whenever communication has to take place.

The communication process is shown in Fig. 1. As can be seen, data has to be copied to the host memory before MPI communication can take place. MPI has been extended to support GPUs, but the communication is still handled by the CPU. The only difference is that GPU memory pointers can be used for MPI routines [10] [11]. This has been supported by the GPUDirect RDMA feature, introduced with CUDA 5.0. It enables the Network Interface Controller (NIC) to access GPU memory without interference of the CPU.

A popular CUDA-aware MPI implementation is MVAPICH [12] from Ohio State University that is based on Infiniband. Again, control flow has still to be returned to the CPU in order to perform communication.

While GPUDirect RDMA works very well in theory, there are some limitations in practice. An issue with the PCIe peer-to-peer protocol and read transactions in current Intel chipsets prohibits the use of this feature. Unfortunately, bandwidth is strongly limited and performance decreased remarkably [10] [13]. MPI implementations like MVAPICH use GPUDirect RDMA for rather small messages to keep latency low, but switch to staging copies as the message size increases.

Another example of MPI communication and GPUs is DCGN [14]. It enables to trigger MPI communication within CUDA kernels by detailing a CPU thread that is responsible for communication. If communication becomes necessary, the GPU writes to a special memory region to signal the CPU thread that communication is supposed to be started. Then, standard MPI communication is used to move data and the GPU is signaled when the data transfer has been completed. However, the authors claim for direct GPU communication that avoids detailing a CPU thread and accompanying context switches.

B. GGAS

In former work, we introduced Global GPU Address Space (GGAS) [1] as a GPU-centric communication model. In-line with the execution model of GPUs, communication is done collaboratively by plenty of threads. Each thread copies at least one value and the NIC forwards incoming memory transactions to the destination node, where the instructions are completed. This avoids context switches, because the control flow can be kept on the GPU even for communication. The approach is shown in Fig. 1. As can be seen, the CPU is inactive for the whole communication. GGAS enables a direct communication path between distributed GPUs.

As with accessing device memory, accesses to the global address space have to be coalesced, otherwise performance is decreased dramatically. Another performance aspect are read accesses. Due to issues with the PCIe peer-to-peer protocol in current Intel chipsets, read accesses from a third party device, such as a NIC, perform bad. In order to overcome this limitation, store-only programming is recommended.

Apart from communication, GGAS provides global barrier synchronization, enabling synchronization across distributed GPUs.

C. Put/Get Models

While GGAS communicates collaboratively and occupies more GPU resources, GPU RMA [15] offloads communication to the NIC. A single GPU thread generates work requests and writes them to the NIC. During the time the NIC performs the data movement, the GPU is free to continue with computation. To ensure synchronization, the NIC provides notifications about completed communication requests. However, the notifications are written to system memory and require PCIe accesses from the GPU. This decreases performance in terms of bandwidth respectively increases latency. In order to handle communication, the NIC has to read data from the GPU memory resulting in peer-to-peer read accesses. As mentioned earlier, due to the PCIe peer-to-peer issue, bandwidth is limited and performance decreased.

Another put/get model is the Global Address Space Programming Interface (GPI) [16], which implements a PGAS model for distributed GPUs. Also, OpenSHMEM [17] has been extended to GPUs and is based on put/get communication. However, communication is still CPU-controlled.

For this work, only GGAS is considered for the energy analysis and as example for GPU-centric communication. GPU RMA also bypasses the CPU, therefore power savings are about the same. Regarding bandwidth and latency, GGAS performs superior than GPU RMA, but lacks in overlap capabilities. However, overlap is not relevant in this work.

D. EXTOLL

This work analyzes the efficiency of GPU-centric communication. As mentioned before, GGAS is a perfect example for GPU-centric communication, however, currently only supported by the EXTOLL interconnection network. EXTOLL is high performance interconnect with focus on low latency communication. Among others, it provides a functional unit to support distributed shared memory. The so-called Shared Memory Functional Unit (SMFU) [18] forwards PCIe transactions, such as read or write accesses, through the network. This is used for GGAS, where memory transactions from several threads are forwarded.

The analysis in the work is done by comparing state-of-the-art CPU tailored communication, such as MPI, with GPU-centric communication. We use EXTOLL as interconnect for our experiments, since it supports both communication methods.

		CPU	GPU	DRAM	Total
Bandwidth	GGAS	26.86	47.05	3.99	77.9
	MPI	43.36	48.86	10.11	102.33
Ping-pong	GGAS	29.95	52.58	5.26	87.79
	MPI	43.25	48.45	8.07	99.77
Barrier	GGAS	18.76	56.68	4.33	79.77
	MPI	39.97	51.18	8.48	99.63

TABLE I: Power consumption of CPU, GPU and DRAM during various benchmark executions. All given values are in Watt and refer to single node power consumption. The power is the average of all measurements over several message sizes respectively number of nodes for the barrier benchmark.

Benchmark	Energy savings	Performance Speedup
Bandwidth	50.67 % %	2.42x
Ping-pong	85.01 %	6.96x
Barrier	67.31 %	2.28x

TABLE II: Average energy savings and performance speedups of GGAS and MPI. The savings are averaged over all measured message sizes respectively number of nodes. The same applies to the speedup calculation.

Note, that we use an FPGA implementation with a core frequency of 157MHz and 64 bit wide data paths.

IV. Energy Analysis

This section provides an energy analysis of direct GPU communication. We ran our experiments on a 12 nodes cluster with an Nvidia K20 GPU each. All nodes are dual socket systems, whereby two nodes are equipped with two Intel E5-2609 and the other 10 with two Intel E5-2630 processors. Note that the bandwidth and ping-pong benchmark ran only on two nodes, whereas the barrier benchmark were executed on the whole cluster.

A. Bandwidth

We implemented the bandwidth benchmark for GGAS and MPI. The GGAS version keeps the control flow on the GPU during the whole communication and writes data collaboratively to the target GPU memory. For MPI, the data is first copied from the GPU to the host, then sent to the target node. On the target node, the data is received and copied to the target GPU memory. The results are shown in Fig. 2.

As can be seen, GGAS always needs less energy than MPI, however the gap becomes smaller for larger messages. On the one hand, this is reasoned by the bandwidth, which is higher for GGAS, but also by the power consumption. While the GPU power is the same for both communication methods, the CPU power differs significantly. MPI requires the CPU to handle communication, whereas GGAS enables the CPU to be idle. The average power consumption is shown in Table I. While additional data copies increases DRAM power for MPI, DRAM power is not affected for GGAS, where data is copied directly between the GPUs.

The course of the energy graph follows the bandwidth graph for both MPI and GGAS. As soon as the bandwidth starts to saturate, the energy per word remains the same. This is

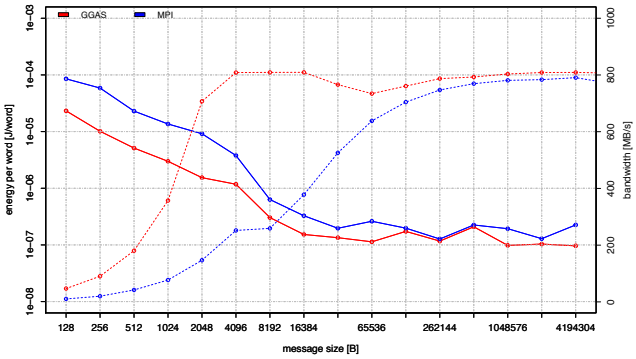


Fig. 2: Energy results of the bandwidth benchmark. The solid lines show the energy per word for given message sizes. Dotted lines show performance in terms of bandwidth.

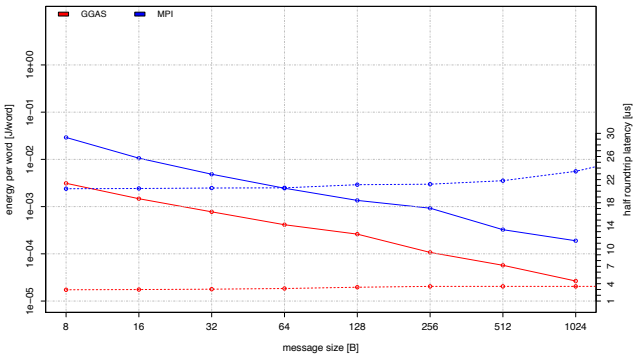


Fig. 3: Energy results of the latency benchmark. The solid lines show the energy per word for given message sizes. Dotted lines show performance in terms of half round trip latency.

reasonable, since the power consumption is independent on the message size and bandwidth remains constant after peak bandwidth has been reached. Regarding Table II, the total savings amount to 55.67% at a average bandwidth speedup of 2.42x.

B. Latency

Besides bandwidth, latency is the second key metric of communication. Again, GGAS keeps control flow on the GPU, therefore the data is written to the target node, received and written back to the source. Because no MPI calls can be made within CUDA kernels, the MPI implementation differs significantly. First, the data is copied from the GPU to the host and then sent via MPI. Next, the source side has to wait until it receives a response from the target. Last, the received data is copied back to the GPU. The energy and latency results are depicted in Fig. 3.

Compared to the bandwidth results, the energy of the latency experiment differs even more. We measured the total energy of the ping-pong benchmark and divided it by two, similar to the half-roundtrip latency for performance. Because the power consumption is static and does not change for different

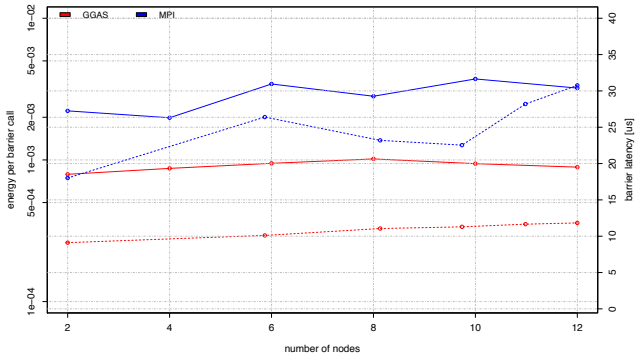


Fig. 4: Energy results of the barrier benchmark. Solid lines show energy per barrier call. Dotted lines show performance in terms of barrier latency.

message sizes, the energy per word decreases for increasing message sizes since latency remains the same. The average power consumption is shown in Table I.

The latency experiment shows a remarkable difference between CPU-tailored and GPU-centric communication. The main reason includes additional PCIe data movements. First, the data has to be copied to the CPU before it can be sent via MPI. The receiving side needs to copy the data to the GPU, but also back again to send the response. Therefore, latency is increased significantly if the CPU is needed to handle communication. With GGAS, data can be directly copied into remote GPU memory and no additional PCIe traffic is required.

Referring to Table II, the savings in energy are about 85% while being almost 7 times faster than MPI. This strongly supports the statement that specialized processors need specialized communication methods.

C. Barrier

GPUs are commonly programmed in a Bulk Synchronous Parallel (BSP) model [19], making barriers inevitably necessary. For MPI, a CUDA kernel is launched, followed by a standard MPI barrier call. The kernel call is necessary to consider context switch latencies. However, no computation is done within the dummy kernel. GGAS provides a global barrier [1], rendering context switches unnecessary. The energy results are shown in Fig. 4.

As can be seen, kernel launch times increases the barrier latency significantly. Furthermore, the global barrier provided by GGAS performs outstandingly and scales better up to 12 nodes. According to Table II, 67% of the energy can be saved. Additionally, the latency of the MPI barrier is over 2x higher than the GGAS barrier.

V. Discussion

In the last section we presented a new metric, Joule/word, to characterize communication while taking the whole processing into account. Both bandwidth and latency experiments show a large difference between CPU and GPU controlled communication. The energy efficiency depends on the message size, whereby larger messages are more energy efficient. However, if messages become too large the latency increases and energy efficiency is eventually decreasing again.

As a result, aggregating messages could improve energy efficiency, however, only within limits since aggregating can increase latency significantly, reducing energy efficiency again.

Barrier synchronization costs about 1 mJ per call for GPU-centric communication like GGAS, but more than twice as much for MPI communication. For MPI, context switches between the GPU and CPU domain increase the barrier latency and therefore increase energy consumption.

In summary, we provide following insights:

- GPU-centric communication is superior in performance and energy efficiency. We show remarkably savings compared to state-of-the-art MPI communication between distributed GPUs
- It is more energy efficient to communicate larger messages instead of very small ones. Aggregating can be an option to improve energy efficiency.
- Synchronization is more energy efficient when it does not require context switches between the CPU and GPU domain.

The insights of this work should be evaluated on application level to see how communication impacts the energy consumption. This is left for future work.

VI. Related Work

A lot of work exists in optimizing application with regard to computation, but also considering energy efficiency. However, communication is usually not part of the analysis.

An optimized energy management for heterogeneous processors and HPC workloads was introduced by Paul et al. [20], however, only intra node optimizations were considered. The impact of clustered GPUs on energy efficiency was investigated by Enos et al. [21], but communication between the GPUs was not part of their work.

In [22], we introduced Infiniband Verbs on GPUs and analyzed performance. However, no power analysis was done. Similar to this work, we investigated performance and power of allreduce operations in [23], but that work focuses on the concept of reduce and allreduce operations using GPU-centric communication.

One of the first works on GPU-centric communication has been done by Owens et al [14], which is described in the background section. With GGAS [1] and GPU RMA [15], we introduced two direct GPU-GPU communication methods. Many work can be found in optimizing MPI with regard to accelerators like GPUs [10] [11] [12]. However, best to our knowledge no analysis on energy efficiency was done regarding the whole communication process.

Several work exists in analyzing energy efficiency of network link controllers or networking hardware. Mentioning all of them would exceed the scope of this work, though.

VII. Conclusion

In this work, we have analyzed the energy impact of GPU-centric communication compared to state-of-the-art MPI communication. For this purpose, we implemented and ran three microbenchmarks: a streaming bandwidth test, a ping-pong experiment to determine latency and a barrier to examine synchronization costs. All benchmark were run with MPI and GGAS as example for GPU-centric communication.

We introduced Joule/Word as metric for energy efficiency and characterized bandwidth and latency in terms if this. For the barrier experiment, we presented Joule per barrier call as metric.

Our results show that GPU-centric communication is superior in performance, but even more in energy consumption. Specialized communication saves both power and time by bypassing the CPU and keeping control flow on the GPU. Additional PCIe data copies and context switches are avoided. For example, savings of up to 85% in energy and a speedup of 7x were achieved for the ping-pong benchmark, but also the bandwidth and barrier benchmark were superior with GPU-centric communication.

Acknowledgment

We gratefully acknowledge the generous support of this research effort by Nvidia, Xilinx Inc, and the EXTOLL Corporation.

References

[1] L. Oden and H. Fröning, "GGAS: Global GPU address spaces for efficient communication in heterogeneous clusters," in *IEEE Cluster*, 2013.

[2] B. Dally, "Gpu computing: To exascale and beyond (keynote)," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2013.

[3] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *Proceedings of the 9th International Conference on High Performance Computing for Computational Science (VECPAR'10)*. Springer-Verlag, 2011, pp. 1–25.

[4] S. Yalamanchili, "Scaling data warehousing applications using gpus," in *Second International Workshop on Performance Analysis of Workload Optimized Systems (FastPath-2013), held with ISPASS-2013.*, 2013.

[5] N. Wilt, *CUDA Handbook: A Comprohensive Guide to GPU Programming*, 1st ed. Addison-Wesley Professional, 2013.

[6] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of gpu kernels using performance counters," in *Green Computing Conference, 2010 International*, 2010, pp. 115–122.

[7] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent gpu architectures," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS '13)*. IEEE Computer Society, 2013.

[8] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," in *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '10. New York, NY, USA: ACM, 2010.

[9] [Online]. Available: developer.nvidia.com/nvidia-management-library-nvml

[10] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. Panda, "Efficient inter-node mpi communication using gpudirect rdma for infiniband clusters with nvidia gpus," in *Parallel Processing (ICPP), 2013 42nd International Conference on*, 2013.

[11] J. Kraus. (2014, march) An introduction to CUDA-aware MPI. [Online]. Available: https://devblogs.nvidia.com/parallelforall/introduction-cuda-aware-mpi/

[12] H. Wang, S. Potluri, M. Luo, A. K. Singh, S. Sur, and D. K. Panda, "Mvapich2-gpu: Optimized gpu to gpu communication for infiniband clusters," *Computer Science - R&D*, vol. 26, pp. 257–266, 2011.

[13] M. Si and Y. Ishikawa, "Direct MPI library for Intel Xeon Phi co-processors," in *Parallel and Distributed Processing Symposium Workshops & PhD Forums (IPDPSW)*, 2013.

[14] J. A. Stuart and J. D. Owens, "Message passing on data-parallel architectures," in *International Symposium on Parallel & Distributed Processing. IPDPS2009*, 2009.

[15] B. Klenk, L. Oden, and H. Fröning, "Analyzing put/get apis for thread-collaborative processors," in *Workshop on Heterogeneous and Unconventional Cluster Architectures and Applications (HUCAA), held with ICPP*, 2014.

[16] L. Oden, "GPI2 for GPUs: A PGAS framework for efficient communication in hybrid clusters," in *Parallel Computing - ParCo2013, International Conference on*. IOS, in press.

[17] S. Potluri, D. Bureddy, H. Wang, H. Subramoni, and D. Panda, "Extending openshmem for gpu computing," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, 2013.

[18] H. Fröning and H. Litz, "Efficient hardware support for the partitioned global address space," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE, 2010.

[19] L. G. Valiant, "A bridging model for parallel computation," vol. 33, no. 8, August 1990, pp. 103–111.

[20] I. Paul, V. Ravi, S. Manne, M. Arora, and S. Yalamanchili, "Coordinated energy management in heterogeneous processors," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. ACM, 2013.

[21] J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J. Stone, and J. Phillips, "Quantifying the impact of GPUs on performance and energy efficiency in HPC clusters," in *Green Computing Conference, 2010 International*, 2010.

[22] L. Oden, H. Fröning, and F.-J. Pfreundt, "Infiniband-verbs on GPU: A case study of controlling an infiniband network device from the GPU," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2014 IEEE 28th International*. IEEE, 2014, in press.

[23] L. Oden, B. Klenk, and H. Fröning, "Energy-efficient collective reduce and allreduce operations on distributed gpus," in *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2014.