Analyzing Communication Models for Distributed Thread-Collaborative Processors in Terms of Energy and Time

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@itwm.fhg.de

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

***Abstract*—Accelerated computing has become pervasive for increasing the computational power and energy efficiency in terms of GFLOPs/Watt. For application areas with highest demands, for instance high performance computing, data warehousing and high performance analytics, accelerators like GPUs or Intel's MICs are distributed throughout the cluster. Since current analyses and predictions show that data movement will be the main contributor to energy consumption, we are entering an era of communication-centric heterogeneous systems that are operating with hard power constraints. In this work, we analyze data movement optimizations for distributed heterogeneous systems based on CPUs and GPUs. Thread-collaborative processors like GPUs differ significantly in their execution model from general-purpose processors like CPUs, but available communication models are still designed and optimized for CPUs. Similar to heterogeneity in processing, heterogeneity in communication can have a huge impact on energy and time. To analyze this impact, we use multiple workloads with distinct properties regarding computational intensity and communication characteristics. We show for which workloads tailored communication models are essential, not only reducing execution time but also saving energy. Exposing the impact in terms of energy and time for communication-centric heterogeneous systems is crucial for future optimizations, and this work is a first step in this direction.**

I. Introduction

The end of Dennard scaling has led to the transition to multi-core and many-core, since the increasing transistor count described by Moore's Law can no longer be kept within a fixed power budget. In general, power consumption will be the main limitation for future computing systems, from handheld devices to ultra-large scale data centers, which increases the importance of optimizing energy efficiency.

Energy efficiency is one of the main reasons to employ accelerators like GPUs or Intel's Many-Integrated-Cores (MICs) in high performance computing systems. Such specialized processors offer much higher performance per Watt compared to general-purpose processors like CPUs. Large-scale clusters currently consume up to 17.8MWatts [1], and as a rule of thumb one MWatt costs about one Million USD per year. Thus, the operational expenditures over the lifetime of such systems can easily exceed the initial capital expenditures. Besides such economic reasons, other good reasons for improved energy efficiency include ecological and technical ones.

We are now facing an era where most of the energy is not spent on computation; instead communication tasks will dominate power consumption [2] [3] [4]. Predictions show that this trend will continue and actually intensify in the future. As a result, we are now observing a fundamental transition to communication-centric systems composed of heterogeneous computing units. Hard power constraints will make energy efficiency a key metric.

GPUs are a prominent example of application-specific accelerators. They differ substantially in their execution model from CPUs. GPUs only excel in performance and thus energy efficiency for *in-core* calculations. Most memory resources are scarce, and Big Data amplifies this problem. Any off-device data access suffers from the huge performance disparity between on-device memory and the PCIe interface. On the other hand, from a peripheral device's point of view, the performance disparity between local and remote accesses is diminishing. Recent networking technologies like Infiniband FDR or 100G Ethernet are approaching the performance levels of intra-host technologies like PCIe. This diminishing disparity provides an opportunity to overcome limitations by sharing resources. However, it is essential to expose the costs in terms of energy and time to users and software layers. Otherwise, optimizations are hindered, or made impossible due to the lack of information.

In this work, we use GPU clusters as an example for a heterogeneous communication-centric system and explore the impact of different communication paradigms in terms of energy and time. As we will see, most communication models are optimized for general-purpose CPUs and poorly match the requirements of GPUs. Similar to heterogeneity in processing, communication models have to be tailored, resulting in more diversity. In particular, we provide the following contributions:

- Reviewing communication models for distributed thread-collaborative processors like GPUs, including standard models like MPI but also specialized models.

- An implementation of multiple workloads using different communication models, covering different communication and computation characteristics.
- A quantitative and comparative assessment of different communication models. Among others, for our 12-node test system observations include a performance increase by an average of almost 1.5x and energy savings of 22%, both for 12 nodes and compared to state-of-the-art communication methods.
- An analysis of the impact of different communication models on energy consumption, indicating future optimizations. Insights include that (1) specialized models offer substantial advantages for a variety of workloads, (2) thread-collaborative models only seem to be limited by reduced overlap possibilities, and (3) a CPU-bypass can significantly reduce energy consumption.

The remainder of this work is structured as follows. Section 2 reviews the required background. Section 3 describes the methodology of our approach. Section 4 reveals the different workloads. Sections 5 and 6 provide the results in terms of execution time and energy, respectively. Section 7 discusses the observations and summarizes key findings. Related work is presented in section 8, while the last section concludes.

II. Background

This section provides detailed information about GPUs and our custom network interface and reviews current communication methods for thread-collaborative processors like GPUs.

A. GPU computing

Modern GPUs are composed of multiple *Symmetric Multiprocessors (SMs)* that consist of hundreds of so-called CUDA cores. This large number of physical cores enables thousands of threads to run in parallel. These threads are organized in blocks that are tightly bound to the SMs. However, the entire block is not scheduled at the same time, instead the block is divided into so-called warps consisting of 32 threads. Schedulers in recent GPUs are able to concurrently schedule two independent instructions of up to four warps. Thus, control flow should be identical within warps, otherwise branch divergence will decrease the utilization. The large number of threads allows to efficiently hide memory access latencies, rendering large cache structures unnecessary. In addition to small L1 and L2 caches, each SM features a fast scratchpad memory that is completely administrated by the programmer. It serves to keep data closer to the cores, increasing the bandwidth significantly. Accesses to off-chip memory have to be coalesced to achieve best memory bandwidth, which is about one order of magnitude higher than what a CPU can achieve with its system memory.

GPUs are programmed with CUDA, for instance, a computing platform introduced by Nvidia in 2007. In order to run instructions on the GPU, the CPU has to launch a parallel kernel. Data transfers between the host system and the GPU have to be explicitly marked as well. The GPU instructions are compiled for the given hardware at run time from a virtual instruction set called PTX. Alternatives like OpenCL or OpenACC are possible, but not used in this work.

B. GPU communication models

The key aspect of this work is the analysis of the impact of different GPU communication models. These communication models are now examined in more detail.

a) Data transfer and communication control: When analyzing communication models, it is essential to distinguish between data transfer and communication control. This is particularly true for heterogeneous systems, as the question then arises which task is done by which unit.

A data transfer between two GPUs can either be staged, which requires additional copies in host memory, or direct, where data is written to the NIC directly. The latter one has been enabled by GPUDirect RDMA for inter-node GPU data transfers, introduced with CUDA 5.0 (2012). This technique brings benefits for small messages, but due to the bad PCIe peer-to-peer read performance, which is caused by a protocol issue in current Intel chipsets [5],, the performance of larger messages is worse then for staged transfers.

The communication control describes where the communication requests are issued and synchronized. The GPU as well as the CPU can control communication. If the CPU controls the communication, every communication request requires a context switch between the GPU and host domains. On the other hand, if communication is controlled by the GPU, the control flow can be kept entirely on the GPU. However, this adds additional work to the GPU, while in the first case this work can be offloaded to the CPU.

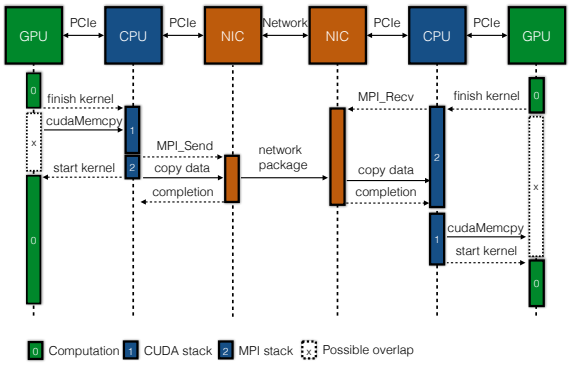
In the following we will elaborate on how different communication models implement these two approaches.

b) Message Passing: Message Passing is characterized by the need for explicit send and receive operations, also known as two-sided communication. The *Message Passing Interface (MPI)* has become the standard for inter-node communication in the CPU domain and is also well-known for distributed GPU programming.

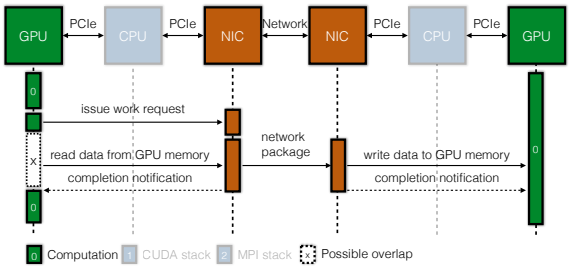
Usually, the communication is controlled by the host and the data has to be placed in host memory, which results in PCIe data movements when employing GPUs. The GPU copies the data to a host buffer and the CPU triggers the network device to start the data transfer. This approach is shown in Fig. 1a and used in this work. The measured bandwidth for a standard two-copy MPI send process is shown in Fig. 2 (MPI).

MVAPICH2 [6] is a well known example for CUDA-aware MPI libraries using Infiniband. It allows GPU memory buffers to be used directly within MPI routines. It uses GPUDirect RDMA [7] for small messages; however, for large messages the library uses staging buffers in host memory to avoid issues with PCIe, whereby the peer-to-peer read operation performs poorly.

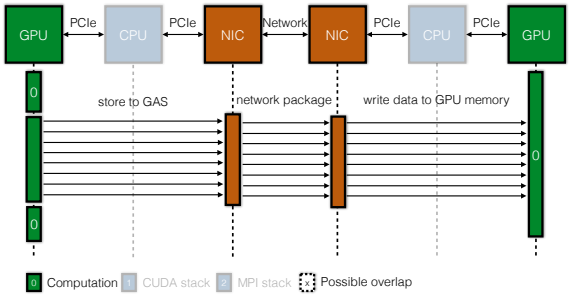
Another approach is DCGN [8], which enables message passing controlled by the GPU. CUDA kernels call messaging routines, while the data transfer is actually handled by the CPU. A CPU thread needs to poll on predefined memory



(a) Message passing



(b) Put/Get: put operation



(c) Load/Store: store operation to GAS

Fig. 1: Communication models

regions that signal whether a data transfer is supposed to be started. On the receiving side, the CPU signals the GPU that new data is available. Note that the authors of [8] explicitly call for a method for direct communication from the GPU.

c) Put/Get model: While messages require explicit send and receive calls, the put/get model is one-sided. A put command writes data directly to remote memory. The main advantage of this approach is that it allows the communication to be offloaded to the NIC instead of creating additional work for the communication source and sink due to tag matching, address translation and synchronization.

However, the system has to provide memory regions that can be used for put/get operations and they have to be registered. This is time consuming and reduces performance.

GPUDirect RDMA enables direct remote memory access to GPU memory. Normally, this communication is controlled from the CPU. For example, this approach is implemented in GPI for GPUs [9].

Another approach is to let the GPU trigger the *Remote Direct Memory Access (RDMA)* transfer with a put operation, as shown in Fig. 1b and as used in this work. The GPU generates the RDMA work request and directly forwards it to the NIC. This is enabled by some minor changes in the low-level device driver. Furthermore, the GPU can directly consume notifications to obtain status information. The CPU is released from communication control and can work on other tasks.

However, there is one drawback: the NIC has to read data from the GPU, which leads to a performance drop because of the PCIe issue.

This performance drop is also visible in Fig. 2 (RMA-Direct, at 1MB). We can avoid this behavior by replacing the PCIe peer-to-peer data transfer with GPU-triggered cudaMemcpys and CPU-to-CPU put operations, as shown in Fig. 2 (RMA-Host). The data transfer is then handled by the CPU, but still relying on put/get operations.

d) Load/Store model (GGAS): The remote load/store model allows direct data transfer between GPUs, entirely controlled by the GPU itself. We introduced GGAS [10], a communication approach perfectly matching the thread-collaborative execution model of GPUs. In GGAS, loads and stores are forwarded by the NIC to a remote GPU. In order to move data, each thread of a CUDA kernel copies at least one value to special addresses and the NIC forwards the plain store operation to the remote memory, as shown in Figure 1c.

Our experiments show that the overhead of this approach is very low. With our FPGA-based prototype network (Fig. 2) we achieve an end-to-end bandwidth that is close to theoretical peak (950MB/s). The network itself has a theoretical peak bandwidth of about 1.2GB/s, which is much less than the PCIe interface (8GB/s) and, in particular, than the GPU's on-device data paths (208GB/s). A large number of GPU threads can easily saturate on-chip data paths, therefore leading to buffer contention for PCIe and network. If store buffers are full, threads will stall, reducing the number of threads that are free to perform computational tasks. As a consequence, possible overlap is reduced.

C. EXTOLL

For this work we use EXTOLL as interconnection network, because unlike most other interconnects it provides functional units for one-sided and two-sided communication, address translation and load/store forwarding. One-sided communication is performed by the *Remote Memory Access (RMA)* unit with advanced notification support, such as hardware generated notifications and a simple format to keep the memory footprint very low. The data transfer is completely handled by the NIC, releasing the CPU to work on other tasks. The *Virtualized Engine for Low Overhead (VELO)* provides mailbox-based send and receive operations for two-sided communication. The NIC also supports distributed shared memory with its *Shared Memory Functional Unit (SMFU)*. Memory operations like loads and stores are forwarded by the network device enabling global address spaces within a cluster. More

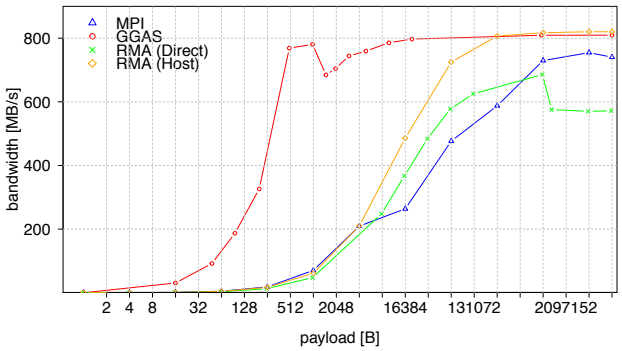


Fig. 2: Achieved bandwidth of various communication models. The custom network is implemented in an FPGA with 157 MHz core frequency and 64bit wide data paths.

details about EXTOLL can be found in [11] and [12].

III. Methodology

Measuring and in particular analyzing power and energy consumption for standard servers is still nascent. For this work, we strive to characterize the implications of different communication models for heterogeneous systems, thus our requirements include identifying which unit (different processor types and memory, network, etc.) consumes how much power with a sample rate that is sufficient to reveal short-lived effects, since these are common for fine-grained communication tasks. We prefer a real test-bed over architectural simulators, as accurate simulation of distributed systems is still prohibitive in terms of run time. This is particularly true as we attempt to cover a broad range of workloads and problem sizes. In addition, recent GPU models (such as Nvidia Kepler-class) are not yet available in simulators [13].

A. Measuring power and energy consumption

Two main methods are available for measuring the power consumption of standard computing systems (based on commodity components, including mainboard and chassis). The first is to use integrated power measurement facilities like Intel's RAPL [14] or Nvidia's NVML [15], which can be used to characterize single components like CPU, host memory, or a GPU add-in card. These facilities use power modeling to allow relatively fine grain instrumentation with a sampling period in the range of 200ms. The second possibility is to use external power meters, either for power outlets or specialized for individual components like add-in cards. Their main advantage is accuracy, however power meters for power outlets can only measure aggregated power consumption and prevent a breakdown into single components. Instrumenting each component with a power meter is at best very cumbersome since probing is invasive and difficult.

In order to characterize the implications of different communication models, it is mandatory to distinguish between the different components. Thus, our methodology relies on integrated power measurement, which does not cover components like the network. However, this is reasonable since network power consumption is independent of the actual traffic, and it is dominated by (de-)serializers, which employ serial link coding and embedded clocking. Network cards, including the port fraction of the switch, consume statically about 20-30 Watt, at least for medium-scale systems. Future work with hardware optimizations for collectives or link-level power saving might revisit this topic, though.

B. Comparing communication models

We already presented in previous work ([10] [16]) that GPUs can benefit substantially from a tailored communication model that allows for thread-collective data transfers. However, such optimizations require small changes in the network interface to allow for global address space mappings; therefore, they are not available in commodity networks like Infiniband and Ethernet. As architectural simulation is not an option, we rely on an FPGA-based prototype system, which is customizable and competitive in terms of performance. Each node of the system is composed of dual Intel Xeon Sandy Bridge CPUs, DDR3 main memory, one Nvidia Tesla K20 GPU and an FPGA-based custom network card. Associated pricing and availability limit our system to 12 nodes.

The custom network card is reconfigurable and allows for GPU-specific optimizations. All functional units (described in section II-C) are highly optimized and other work has shown the competitiveness of the implementations [11] [12]. Thus, the insights in terms of relative improvement can be applied to other interconnects like Infiniband or Ethernet.

C. Representative set of workloads

Many workloads and benchmarks are available for single GPUs, but rather few for GPU clusters. Additionally, available codes rely on standard communication models like MPI. Since the main objective of this work is to characterize the impact of alternative communication models, we unfortunately cannot rely on available codes. In order to ensure a fair comparison between the different communication models, we chose to implement our own set of workloads, each optimized for all three communication methods.

Applications exhibit a large variety of different behaviors, and with our set we want to cover as many as possible. The lack of available tests limits our possibilities, though. Thus, we choose workloads with a variety of characteristics in terms of the communication pattern, computational intensity, average network payload size and possibilities for overlap. We selected a set that includes n-body computations, stencil codes, global operations and randomized updates. The first three show communication characteristics that are regular with a static behavior, including patterns like rings, nearest-neighbor and all-gather. We add a GPU-based RandomAccess test (similar to the one in [17]) to cover irregular and dynamic communication characteristics, as they are often found in emerging codes like graph computations. Since most tests only cover a small

range of payloads for a given problem size, we also vary the problem sizes within the bounds imposed by memory capacity. As a result, our workload set covers many patterns, payload sizes, and characteristics and more to make our explorations as comprehensive as possible.

IV. Benchmarks

We use the following benchmarks to assess performance and energy consumption of different communication methods. This section gives a short overview of the benchmarks and explains their implementation for GGAS (load/store forwarding), RMA (put/get, one-sided) and MPI communication.

A. N-Body

The N-Body benchmark is a well known computationally intensive benchmark that scales strongly with the number of processing nodes. The algorithm can be applied to many physical problems in astrophysics and in molecular dynamics. The algorithm can be easily mapped to the execution model of GPU, where high degrees of parallelism can be exploited. This makes the N-Body problem applicable for distributed GPUs.

All bodies are distributed among the cluster nodes and communication follows an all-to-all pattern. If the number of bodies increases, the communication becomes less significant and the time that is spent for computation develops to be the dominating factor.

We implemented this benchmark using 3D gravitational forces. For the GGAS communication model, we did not place the bodies in global shared memory, because the GGAS read bandwidth is about a sixth of the write bandwidth [10]. Instead, mailboxes are used to provide send/receive semantics. To achieve the best overlap for MPI and RMA data transfers, the communication is done using a ring scheme. This means that bodies are sent to the successor node while interactions with the predecessor node are calculated. The compute CUDA kernels are the same for all communication methods, only the communication code differs. The computational code is based on the N-Body example from the Nvidia CUDA SDK [18].

B. Himeno (Stencil)

The Himeno benchmark solves a 3D Poisson equation on a structured curvilinear mesh and can be regarded as a 19-point stencil algorithm. Unlike the N-Body test, it is memory intensive [19], therefore suitable for GPUs as their memory bandwidth is about one order of magnitude higher than for CPUs. As GPUs are limited in their on-device memory capacity, distributed GPUs are helpful for large problem sizes.

The 3D grid is divided into planes in the z-direction. These planes are permanently assigned to the processing nodes. Each point of the grid depends on neighbor points that have to be exchanged with adjacent processing nodes. However, only single planes are communicated with the nearest neighbor node.

For GGAS and RMA, the control flow always remains on the GPU. Planes that have to be shared with other nodes are computed first and then sent to the neighbors to maximize overlap. This also applies to the MPI implementation, but the communication is handled by the CPU; therefore, data has to be copied to system memory before communication takes place. The computation is divided into three kernels (top, middle and bottom), as each has its own communication case.

C. Global Sum (Reduce)

The global sum benchmark is based on collective allreduce operations, which are very common in scientific simulations. This benchmark accumulates an array of input values, which is distributed over the GPUs. First, every GPU reduces its own field of input values, then an allreduce operation with one input value is used to determine the global sum.

In the GGAS version we used an all-gather approach for the allreduce operation, where all GPUs send their input data to all other GPUs, which then perform the reduction operation on their own local memory. Results in [16] have shown that this is the most efficient way for a small number of input values and a small number of GPUs. For a larger number of GPUs, the tree-based approach may be more efficient, however, our tests show that this was not the case for up to 12 GPUs.

In the MPI version, the local accumulation result is copied to host memory, and then an MPI-allreduce operation (with a single floating value) is used to determine the global sum.

This benchmark was not implemented for RMA communication, because only one value has to be communicated for each node. The RMA transfer requires a descriptor to be written to the NIC, therefore the overhead for one value is too high to perform the communication efficiently.

D. RandomAccess

The RandomAccess benchmark was introduced with the HPC Challenge Benchmark Suite [20] and implemented for MPI to assess network performance. Each node generates random indices to modify values of a distributed table. This results in random accesses and a lot of pressure on the network (device). Here, we port it to GPUs.

In general there are two kinds of accesses: local and remote. Local accesses directly modify the partition of the table that is kept by that node. Remote accesses are updates that target a different partition on another node. Communication is random but uniform due to the randomized updates. The message size is also random, because the rules allow the creation of up to 1024 indices (called look-ahead) at once, resulting in message sizes of up to 1024 elements.

For GGAS, in a naïve approach, the table could be accessed directly by all participating GPUs, meaning that the table is accessible by the global address space provided by GGAS. However, there are no atomic operations for remote accesses to ensure mutual exclusion. In addition, remote read operations are slow [10] and therefore remote-store programming should be used. To address these issues, we used the same mailbox approach as for N-Body. The MPI version stores the table in GPU memory, too, but creates all indices on the CPU. The table is updated by the target GPU. The RMA version keeps the control flow on the GPU and sends the indices directly

TABLE I: Overview of benchmark characteristics. The problem size is referred to as n and the number of nodes as N.

	N-Body	Himeno	global sum	Rnd.Access
Computation	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Memory	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Communication	$\mathcal{O}(n)$	$\mathcal{O}(\sqrt[3]{n^2})$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$
Communication Pattern	Ring	Neighbor	All-gather	Rnd. Unif.
Avg. Payload	$\frac{n}{N}$	$\sqrt[3]{n^2}$	1 val.	0-1024 vals.
Overlap	+	+ +	- -	-

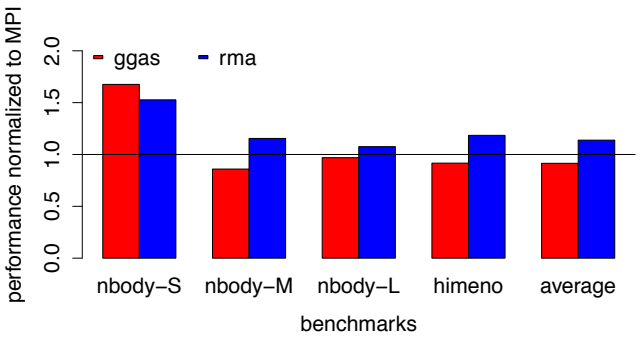


Fig. 3: Benchmark results to characterize overlap capabilities of different communication methods. *nbody-S* refers to 4K bodies, *M* to 15K and *L* to 128K. The benchmarks were executed on all 12 nodes. The average does not consider *nbody-S*, this benchmark serves as a reference at this point.

to the target GPU. For a look-ahead of 1024, every CUDA block runs 1024 threads and creates all indices in parallel. Decreasing the number of threads also decreases the look-ahead and therefore performance. For our experiments we always use a look-ahead of 1024.

E. Comparison and summary

We chose benchmarks with different characteristics to cover a broad range of applications. The comparison of these benchmarks is done in Table I. As shown, N-Body and Himeno offer lots of overlap and differ in communication complexity and patterns, while the global sum and RandomAccess benchmarks rely on small messages with almost no exploitable overlap. This diversity provides a good baseline to assess the different communication methods.

V. Performance analysis

This section presents and analyzes the performance of our benchmarks with regard to different characteristics. The test system consists of 12 dual-socket nodes with 10 nodes equipped with two Intel E5-2630 and two nodes with two Intel E5-2609 processors. Every node is supplied with one Nvidia K20 GPU. We use EXTOLL network cards based on an FPGA with 157 MHz core frequency and 64 bit wide data paths. The NIC and the GPU use PCIe 2.0 and share one root complex.

Fig. 4 shows the performance of all benchmarks and node counts to provide an overview of our experiments. N-Body and global sum are shown with two problem sizes each, one for significant long communication periods and one rather focused on computation with only less communication. The results lead to observations that will be analyzed in the following.

A. Overlapping of computation and communication

Overlap is an important aspect of efficient communication, since communication time can be hidden. MPI offers non-blocking communication routines to exploit overlap and RMA transfers offload communication to the NIC, releasing the processing unit to continue with other work. Furthermore, in our case MPI can offload communication entirely to the CPU, releasing the GPU for computation. In order to evaluate overlap capabilities of different communication methods, performance results of the N-Body and Himeno benchmark are shown in Fig. 3. Both applications offer enough overlap to hide data movement costs efficiently.

The figure shows three N-Body runs with different problem sizes together with the results of the Himeno benchmark. Regarding *nbody-S*, GGAS as well as RMA outperform MPI by being 1.7 respectively 1.5 times faster. Small problem sizes are more sensitive for communication since the computational effort is rather low, resulting in low overlap possibilities. Increasing the problem size leads to longer computing periods and therefore more overlap can be exploited.

Although GGAS achieves higher bandwidth than both RMA and MPI, the performance for *nbody-M* and *nbody-L* problem sizes is lower than MPI or RMA. It seems that overlap possibilities for GGAS are limited. GGAS needs usually one thread per word to be transferred, and therefore requires lots of resources for communication. This observation is backed up by the Himeno benchmark, where again RMA and MPI perform better than GGAS. RMA is superior than MPI because at some point communication takes too long and cannot be efficiently overlapped for MPI anymore. The CPU is still occupied with communication and the GPU has to idle until this is finished.

B. Irregular communication patterns and small messages

As we have shown in [10], GGAS performs superior for small messages since latency can be kept very low and message rates are high. Irregular communication has become more and more popular, nonetheless due to graph processing applications. We examined the performance of such irregular patterns by running the RandomAccess benchmark with different communication approaches. The results, together with the results of the global sum benchmark, are shown in Fig. 5.

The global sum benchmark calculates the sum of a local array and determines the global sum by using an allreduce operation, thus only one value per node has to be communicated across all nodes. For small array sizes, the performance using GGAS is 2 times better than using MPI. The difference becomes smaller the larger the array becomes, but GGAS still performs better. Increasing the array size requires more computation and communication becomes less important. The same applies for the RandomAccess benchmark, where GGAS is again almost 2 times faster than MPI and RMA. This is due

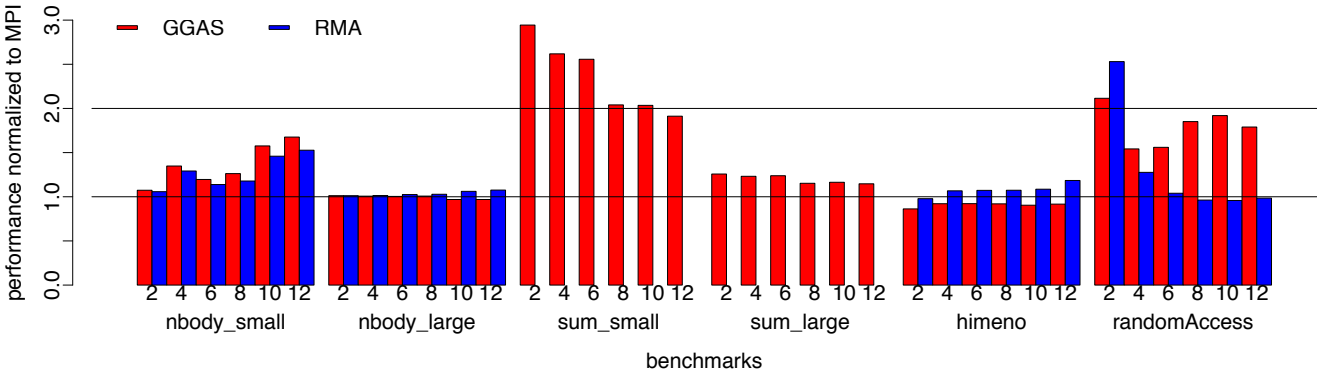


Fig. 4: Performance normalized to MPI for all benchmarks and node counts. *nbody_small* refers to 4K bodies, large to 128k bodies. The global sum benchmark is shown with 128 elements (small) and 256k elements (large).

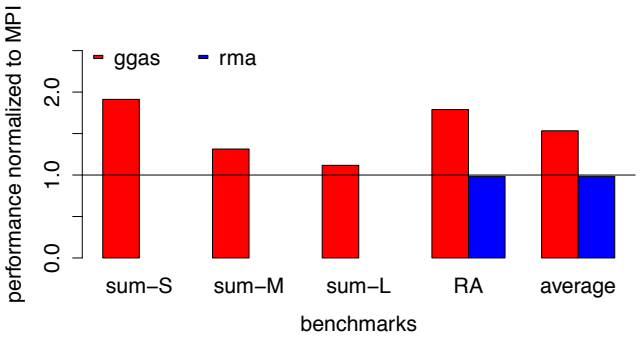


Fig. 5: Benchmark results to evaluate the performance of applications based on small messages respectively irregular patterns. *sum-S* refers to 128 elements, *M* to 32K and *L* to 512K. The benchmarks were executed on all 12 nodes.

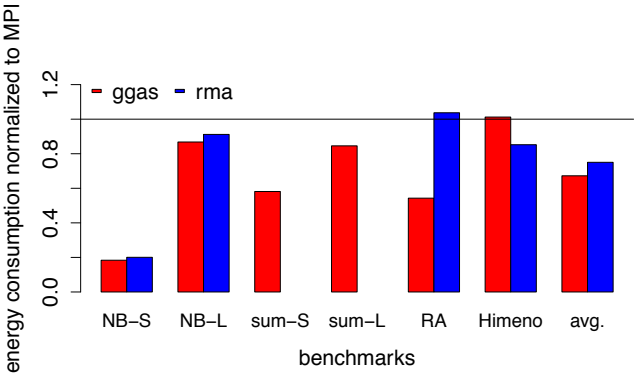


Fig. 6: Energy analysis results of different benchmarks. Lower means better than MPI. The benchmarks were executed on all 12 nodes.

to the very low latency that can be achieved with avoiding context switches and MPI overhead as well as PCIe copies.

C. Specialized communication vs. CPU-tailored communication

We showed that RMA can exploit overlap very well and GGAS is suited for small messages and irregular communication patterns. Regarding Fig. 4, another observation is that at least one of specialized communication models always performs better than traditional CPU-tailored communication like MPI. GPUs differ in their execution model inherently and additional PCIe copies increase latency significantly. GGAS perfectly matches the execution model of GPUs and like RMA, the control flow can be kept on the GPU during the entire communication process.

Taking scalability into account, the benchmarks behave differently. While N-Body scales very well for small problem sizes, the global sum benchmark shows that performance of GGAS decreases slightly compared to MPI. An increase of the number of nodes means that transfers between the GPU and the host become less important than the actual communication latency. However, the global sum calculation using GGAS is still 2 times faster with 12 nodes than MPI. The RandomAccess benchmark scales very well for GGAS, but less for RMA. More nodes means more communication and RMA communication requires system memory accesses, decreasing performance significantly [21]. For Himeno, the performance difference of GGAS, RMA and MPI is not affected by the number of nodes.

VI. Energy analysis

Complementing the performance analysis, the influence of different communication methods on the energy consumption is analyzed in the following.

A. Impact of performance on energy consumption

Energy is the product of power consumption and execution time, and therefore performance advantages while keeping power consumption constant leads automatically to less energy consumption. In general, we observe that it is easier to save time than power. However, direct communication like GGAS or RMA are promising to additionally save power by bypassing the CPU. The CPU remains idle or becomes free to

TABLE II: Power consumption of different benchmarks and communication methods regarding CPU, DRAM and GPU power consumption. All given values are in Watts.

		N-Body	Himeno	Gl. Sum	RA
GGAS	CPU	33.21	25.94	18.38	27.60
	DRAM	3.18	2.89	1.63	3.27
	GPU	81.01	105.13	63.57	68.67
	Total	**147.43**	**133.96**	**83.59**	**99.54**
RMA	CPU	32.16	23.96	NA	28.16
	DRAM	3.27	2.60	NA	3.22
	GPU	87.91	116.53	NA	79.48
	Total	**152.22**	**143.09**	**NA**	**110.85**
MPI	CPU	37.66	34.23	34.00	35.26
	DRAM	3.86	3.40	2.97	3.62
	GPU	82.93	106.99	57.48	51.79
	Total	**158.25**	**144.62**	**94.44**	**90.66**

perform other work, however the analysis of the latter is left for future work.

Regarding Fig. 6, the *nbody-S* benchmark saves up to 80% energy using GGAS or RMA. Most savings are based on the high performance benefit of these communication methods. The same applies to the global sum and RandomAccess benchmark, for which GGAS performs remarkably. Benchmarks that perform similar with different communication methods, such as N-Body, global sum with large problem sizes and also Himeno, can benefit from direct communication and save energy. On average, applications using GGAS or RMA show savings of about 35 % respectively 25 %.

Interestingly, MPI performs slightly better for large problem sizes and N-Body, but GGAS and RMA still need less energy due to power savings. Almost the same is true for Himeno, where the difference in energy is not as distinct as in performance.

B. Power consumption

Besides execution time, power consumption is the second factor of energy. Table II shows the average power consumption during the execution of the appropriate benchmark using different communication methods. Regarding CPU power, GGAS and RMA always need less power than MPI. On average, GGAS and RMA consume about 9 W less CPU power than MPI.

In most cases, GPU power is not increased when GGAS or RMA is used. Only RandomAccess behaves differently, however, this is due to the implementation, whereas MPI calculated the indices on the CPU to avoid further PCIe copies. This design decision was made to stay comparable to the other direct communication approaches.

Host memory accesses are necessary for MPI, but can be avoided by using GGAS. RMA enables the GPU to trigger communication, but still needs to access host memory to ensure synchronization by polling on notifications. Regarding Table II, host memory accesses caused by RMA communication do not increase CPU or DRAM power, but increases the power consumption of the GPU.

In summary, GGAS and RMA save a significant amount of power by bypassing the CPU. While GGAS always needs less power than MPI, RMA consumes roughly the same on average. RMA communication still requires host memory accesses to consume notifications and ensure synchronization, adding power to the GPU. DRAM power is almost not affected.

VII. Discussion

We showed performance and energy results of four benchmarks: N-Body, Himeno, global sum and RandomAccess. All benchmarks behave different and rely on distinct communication patterns. N-Body communicates in an all-to-all manner, while Himeno exchanges data only with neighbor nodes. The global sum benchmark uses only very small messages and RandomAccess puts a lot of pressure on the network by accessing random values across all nodes.

The performance analysis has identified observations 1-4, each supported by multiple experiments and summarized in Table III. This leads to following insights:

1) Specialized communication models like GGAS and RMA can offer substantial advantages for a variety of workloads. The case for which MPI is the better choice seems limited, and actually seems to be artificially emphasized by the PCIe peer-to-peer performance issue.
2) Overlap is the main drawback of GGAS as a tailored communication model for thread-collaborative processors. A GPU communication library should therefore combine it with a high-overlap model like RMA.

This strongly supports our statement that specialized processors like GPUs can benefit from specialized communication models tailored for these processors. The use of general-purpose communication models, which are designed and optimized for CPUs, should be avoided to maximize performance.

In terms of energy, performance is still an influential aspect. It seems that saving time is still much easier than reducing power consumption. We showed that energy behaves similar to performance, meaning that if a communication method leads to a superior performance it likely consumes less energy, too. In particular, see observations 5-9 in Table III.

If performance is very similar, GGAS and RMA benefit from the CPU being idle during almost the whole application. Our experiments show that GGAS as well as RMA need about 9W less CPU power and, except for RandomAccess, the GPU power is not increased significantly. However, the CPU still needs about 25W which is about 7W more than idling [16]. This is due to additional PCIe traffic when controlling the NIC from the GPU.

These observations support the previous insights. Compared to performance, advantages in terms of energy are even larger and bypassing components like CPUs allow for significantly reduced power consumption. In particular, the benefits of a GPU-tailored communication library are strongly supported by the energy experiments.

Looking ahead, specialized processors like GPUs will likely evolve from accelerators to autonomous processors. The OpenPOWER initiative is a prime example for this, in which IBM Power CPUs and Nvidia Tesla GPUs will be peer devices

TABLE III: Observations that are supported by our experiments: N-Body (N), Himeno (H), global sum (S), and RandomAccess (R).

	Observation	N	H	S	R
1	RMA and MPI offer a better exploitation of overlap possibilities	X	X		
2	GGAS performs outstanding for small payloads, as no indirections are required like context switches to the CPU or work request issues to the NIC	X		X	X
3	The PCIe peer-to-peer read problem results in MPI performing better than RMA or GGAS for large payload sizes	X	X		
4	GGAS in combination with RMA outperform MPI substantially (without the PCIe peer-to-peer read limitation)			X	X
5	In practice, the execution time has an essential influence on energy consumption	X		X	X
6	Accesses to host memory contribute significantly to DRAM and CPU socket power	X			
7	Bypassing a component like a CPU can save enough power to compensate a longer execution time, resulting in energy savings		X		
8	Staging copies contribute significantly to both CPU and GPU power, due to involved software stacks respectively active DMA controllers		X		
9	For irregular communication patterns or small payloads, GGAS saves both time and energy			X	(X)

connected by a high performance interface [22]. This renders interactions for supervision of accelerators unnecessary, resulting in less coupling between the heterogeneous processors for complex tasks. The freedom to schedule both computation and communication freely is highly desirable then, however, it is only feasible if we have suitable communication models for the different (specialized) processor types.

VIII. Related Work

Various related work exists in the area of energy-aware heterogeneous processing, however surprisingly little in the domain of distributed heterogeneous processing and the influence of communication models.

Instrumentation for power measurement of single processors is a key component of this work. Usually, performance counters are used to model power dissipation. In [23] a linear model is used, while in [24], the model is based on neural networks to improve accuracy. In [25], the authors take another approach by using an empirical power model to predict power consumption, and the model in [26] allows for analyzing single components. While we are currently using standard vendor solutions, we are considering adapting these models to characterize communication sub-operations in terms of energy.

To the best to our knowledge, analyzing energy consumption for heterogeneous clusters still has little related work. The approach in [27] optimizes energy management for heterogeneous processors and HPC workloads, but only investigates intra-node optimizations. Similarly, the authors of [28] explore the effects of *Dynamic Voltage and Frequency Scaling (DVFS)* for a single Nvidia Tesla GPU. The authors of [29] explore the impact of GPUs in a cluster on energy efficiency, but do not consider optimized communication models and focuses on quantifying performance-per-watt improvements.

On the other hand, power analysis of homogeneous clusters has received plenty of interest ([30] [31] [32] [33] [34]), in particular with regard to DVFS. In [35], the energy consumption for certain MPI communication primitives is analyzed. Compared to this work, these contributions neither address heterogeneity nor compare different communication models.

Optimizing communication for heterogeneous systems is currently seeing many contributions. Examples include optimizations for message passing [6] [7] and Put/Get models [36] [37] [9], based on OpenShmem, Global Arrays and the Global Address Space Programming Interface (GPI), respectively. GGAS [10] seems to be the only model that matches the thread-collaborative execution model of GPUs. For this work, we selected MPI, a Put/Get model and GGAS for a representative comparison.

IX. Conclusion

In this work, we compared different communication models for thread-collaborative processors in terms of energy and time. We implemented and analyzed three communication models for four benchmarks: N-Body, Himeno (Stencil), global sum (Reduce), and RandomAccess. We showed and analyzed the influence of the communication model on execution time and energy consumption. Our observations from performance and energy experiments lead to a set of insights, in particular about the benefits and drawbacks of each communication model depending on the workload characteristics.

In summary, we found that specialized communication methods have a significant impact on both energy and time. Similarly to heterogeneity in processing, heterogeneity in communication allows for more optimizations. General-purpose communication methods are designed and optimized for CPUs, which can be cumbersome when applying them to specialized processors like GPUs. Our experiments show that GGAS and Put/Get semantics offer substantial advantages.

In the future we plan to design communication abstractions that switch between models like GGAS and RMA depending on current demands, resulting in a specialized communication library for GPUs. Other plans include exploring savings for concurrent CPU/GPU tasks, and a larger variety of workloads. In particular, workloads related to data warehousing, graph processing and the emerging high performance analytics. While we focus here on performance and energy, improved productivity by avoiding hybrid programming models is left for future work, too.

Acknowledgment

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

REFERENCES

[1] [Online]. Available: www.top500.org

[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] R. A. et al., "Gpu peer-to-peer techniques applied to a cluster interconnect," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2013 IEEE International Symposium on*, 2013.

[6] 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.

[7] 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.

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

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

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

[11] H. Fröning, M. Nüssle, H. Litz, C. Leber, and U. Brüning, "On achieving high message rates," in *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2013.

[12] M. Nüssle, H. Fröning, S. Kapferer, and U. Brüning, "Accelerate communication, not computation!" in *High-Performance Computing using FPGAs*, W. Vanderbauwhede and K. Benkrid, Eds. Springer, 2013.

[13] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12. New York, NY, USA: ACM, 2012, pp. 335–344. [Online]. Available: http://doi.acm.org/10.1145/2370816.2370865

[14] 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.

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

[16] 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.

[17] [Online]. Available: http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/

[18] Nvidia, *CUDA GPU Computing SDK*.

[19] E. H. Phillips and M. Fatica, "Implementing the himeno benchmark with cuda on gpu clusters," in *Parallel and Distributed Processing (IPDPS)*, 2010.

[20] P. Luszcek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, "Introduction to the hpc challenge benchmark suite," 2005.

[21] B. Klenk, L. Oden, and H. Fröning, "Analyzing put/get apis for thread-collaborative processors," in *HUCAA Workshop in conjunction with ICPP, Minneapolis, MN, USA*, 2014.

[22] [Online]. Available: http://openpowerfoundation.org/

[23] 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.

[24] 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.

[25] S. Hong and H. Kim, "An integrated gpu power and performance model," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 280–289, Jun. 2010.

[26] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "Gpuwattch: Enabling energy optimizations in gpgpus," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 487–498, Jun. 2013.

[27] 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.

[28] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong, "Effects of dynamic voltage and frequency scaling on a k20 gpu," in *Parallel Processing (ICPP), 2013 42nd International Conference on*, 2013.

[29] 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.

[30] S. Pakin, C. Storlie, M. Lang, R. E. Fields, E. E. Romero, C. Idler, S. Michalak, H. Greenberg, J. Loncaric, R. Rheinheimer, G. Grider, and J. Wendelberger, "Power usage of production supercomputers and production workloads," *Concurrency and Computation: Practice and Experience*, 2013.

[31] B. Rountree, D. Lowenthal, S. Funk, V. W. Freeh, B. De Supinski, and M. Schulz, "Bounding energy consumption in large-scale mpi programs," in *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, 2007.

[32] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, pp. 835–848, Jun. 2007.

[33] D. Li, B. De Supinski, M. Schulz, K. Cameron, and D. Nikolopoulos, "Hybrid mpi/openmp power-aware computing," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010.

[34] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[35] A. Venkatesh, K. Kandalla, and D. Panda, "Evaluation of energy characteristics of mpi communication primitives with rapl," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, 2013.

[36] 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.

[37] V. Tipparaju and J. S. Vetter, "Ga-gpu: Extending a library-based global address spaceprogramming model for scalable heterogeneouscomputing systems," in *Proceedings of the 9th Conference on Computing Frontiers*, ser. CF '12. ACM, 2012.