Scalable Communication Architectures for GPU-centric Systems

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Benjamin Klenk, Holger Fröning

Institute of Computer Engineering
Ruprecht-Karls University of Heidelberg
Mannheim, Germany

We appreciate Nvidia's sponsoring to support this work

NVIDIA

Objectives

- Analysis of communication models for GPU-centric systems
- GPU-sided NIC access to enable communication offloading
- Managed communication schemes on top of NVLink-enabled Distributed Shared Memory (DSM)

Introduction

Performance scaling in the Post-Dennard era is mainly based on a steadily increasing amount of parallelism. In such parallel systems, GPUs have been proven to be very efficient for compute-intensive tasks, both in terms of performance and energy efficiency. The introduction of NVLink and hardware-assisted unified virtual memory (UVM) demonstrate that **GPUs are climbing up the food-chain to become peer processors**.
Although computation runs efficiently on GPUs, they have lacked the ability to source and sink networking traffic accordingly. **Communication is traditionally controlled by the CPU**. Consequently, many MPI implementations have become GPU-aware, which allows pointers to GPU memory to be passed to MPI routines. However, control flow still has to be returned to the CPU for every communication phase. This not only adds latency, but also reduces programmability and usability.
Due to **GPUDirect, a Network Interface Controller** (NIC) can access GPU memory via PCIe's many CPU chipsets significantly limit the bandwidth of remote read accesses through their PCIe root complex. In this case, a **store-only programming model** becomes inevitable.
NVIDIA recently released the **NVLink-capable Pascal architecture**, which enables **complex communication semantics on the GPU** itself. Nonetheless, the question what are the right communication semantics for such highly parallel processors remains still open.

Communication Models and GPU-NIC Interactions

A. CPU-CONTROLLED

In a traditional system, GPUs are merely compute accelerators, leaving communication up to the **CPU**. Additionally, **MPI** is the de-facto standard for data exchange in distributed systems. This combination significantly adds latency as data needs to be copied from the GPU to the host before it can be sent across the network.
GPUDirect enables NICs to read GPU memory directly, however, traversing the PCIe root complex in x86 CPUs limits read bandwidth to only hundreds of MBs/s. If the NIC and the GPU do not share a PCIe switch, **staging copies** in the host are inevitable for utilizing full PCIe bandwidth.

B. GPU-CONTROLLED

GPUs are optimized for throughput, making them less sensitive to latency. This makes a strong case for a **global shared address space** where memory operations are forwarded across the network fabric (GGAS - GPU Global Address Space [1]. A communication library that builds on the EXTOLL interconnect).

Put/Get communication on a GPU is divided into two parts: **writing descriptors and triggering communication**. Infiniband requires descriptors to be written to work queues and then to ring a doorbell register, informing the NIC about newly arrived work. EXTOLL, as an example for another network architecture, fuses these operations into a single write operation to the device's PCIe BAR region [2].

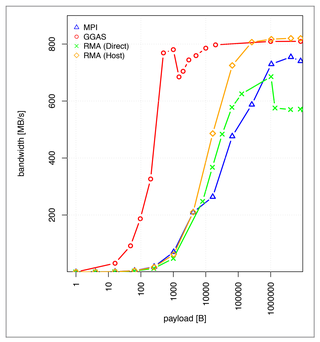

Fig. 1: Bandwidth of different communication models. RMA offloads the data transfer to the NIC. GGAS allows threads to directly write to remote GPU memory.

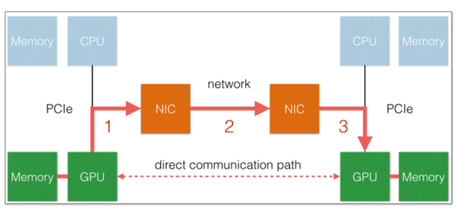

Fig. 2: Device-sided communication model where the NIC is controlled by the GPU and the CPU is bypassed.

COMPARING COMMUNICATION METHODS [3]

- CPU-controlled communication
  - expensive in terms of performance and energy
  - context switches reduce programmability
- GPU-controlled load/store communication (GGAS)
  - tremendous performance advantage for small messages
  - remote-store programming hinders overlap
  - CPU bypass saves energy by allowing the CPU to enter sleep states
- GPU-controlled put/get communication (RDMA)
  - cannot fully utilize bandwidth, but offers overlap
  - control structures have to be resident in GPU memory [2]

Software-defined GPU-centric Communication Models

NVIDIA's introduction of **NVLink** allows to bypass the CPU and to communicate with GPU peers directly at high bandwidth. Furthermore, GPUs offer a tremendous amount of massively parallel streaming multiprocessors (SM), ranging up to more than 50 in the current Pascal architecture. Instead of offloading communication to a NIC, **one SM is dedicated to perform communication, implementing a software-defined NIC.**
Although NVLink enables a virtual shared global address space (GAS) across connected GPUs, it seems promising to explore managed communication, like message passing or one-sided communication on top. In particular at scale, it seems questionable whether it's feasible to fully expose buffer management without any management library in between. Additionally, send/recv semantics have proven to be easy to understand with its exposed access costs. Put/get-based communication, on the other hand, keeps the overhead to a minimum. This creates familiarity with CPU-based message passing, but also offers a simple data exchange model many applications can be mapped to.

MESSAGE PASSING CHALLENGES

- Efficient queue structures for inter-CTA and inter-GPU data exchange
- Message and receive request matching is sequential, but GPUs require a substantial amount of parallelism to operate efficiently
- CTA scheduling is transparent and cannot be controlled by the user
- GPUs do not allow for dynamic memory allocation at kernel level

Here, the first two items are addressed. Recent Pascal-class GPUs provide initial support for preemption, and we assume that future GPU generations are going to introduce kernel-level memory allocations (see also [5]). Please note that this is ongoing work.

A. EFFICIENT QUEUEING MECHANISMS

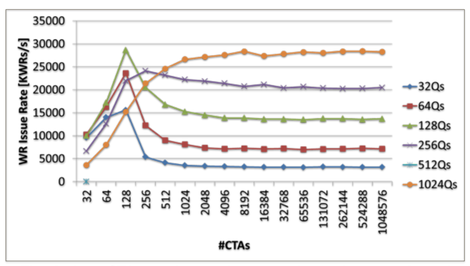

Fig. 3: Inter-CTA queuing performance for various number of warps. The Queue length is 4K Work Requests (4B/WR). 32 WRs are enqueued per CTA. One thread dequeues from one queue.

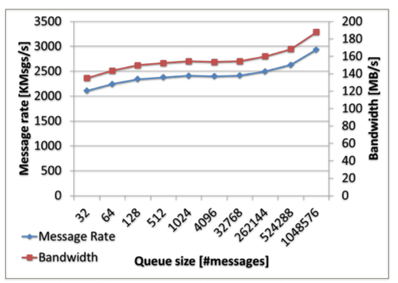

Fig. 4: Inter-GPU queueing. Note that this is the performance of a single warp. Each message amounts to 4B. GPUs are connected via PCIe.

Queues are essential data structures for communication as they preserve ordering and allow for dynamic resource management. Fig. 3 and 4 depict performance results of some benchmarks.

Hardware mechanisms are needed to yield high message rates on GPUs.

B. TAG MATCHING

The amount of dependencies found in typical matching tasks dramatically limits parallelism. Guarantees have to be relaxed to reduce the amount of dependencies. This particularly includes prohibiting wildcards as well as unexpected messages, and relaxing ordering guarantees. For an improved understanding of the current use of message passing in applications, we have analyzed a set of Exascale proxy applications (CESAR, EXMATEX, Design Forward, EXACT) [4] with the following key findings:

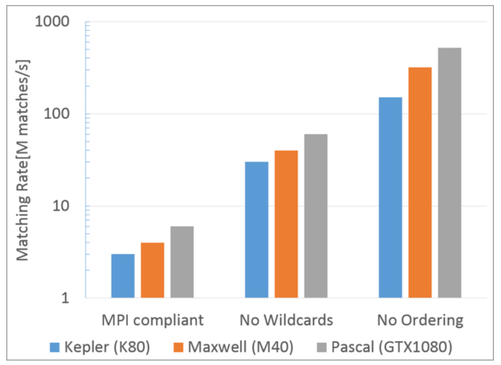

Fig. 5: Message/receive request matching performance for various restrictions, such us no wildcards and no ordering. Performance is shown for three generations of GPUs.

KEY FINDINGS OF OUR EXASCALE APPLICATION ANALYSIS

- Send/Recv is still the majority of MPI calls in most applications
- None of the applications uses tag wildcards, only two applications apply source wildcards
- Both the Unexpected Message Queue (UMQ) and the Posted Receive Queue (PRQ) are mostly smaller than 1024 entries
- The number of peers for an MPI rank ranges between 20 and 50 for most applications

Based on these insights, a parallel message/receive matching is promising without introducing too many restrictions. For example, **wildcards can be prohibited** and message queues be partitioned by source rank. This allows matching rates up to **60M matches/s. Removing ordering constraints even yields 500M matches/s (GTX1080).**

Future Work

Autonomous GPU communication becomes inevitable for future HPC systems as GPUs are getting clustered together with NVLink. Applying traditional message passing, such as MPI, seems sub-optimal due to its sequential behavior and guarantees. We identified a set of relaxations that allow for improved parallelism, rendering message passing better suited for such massively thread-parallel processors. We believe our insights to be highly valuable for the design of communication models and libraries for GPUs, as they provide a detailed understanding of performance implications.

Our next step is to compile a **library** that provides essential **building blocks**, including **queues**, **tag matching** (queue and hash table based) and **memory management.** These blocks can be assembled to create a dynamic and fully adaptable communication engine for GPUs.

References

[1] L. Oden and H. Fröning, "GGAS: Global GPU address spaces for efficient communication in heterogeneous clusters," in IEEE Cluster, 2013.
[2] 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.
[3] B. Klenk, L. Oden, H. Fröning, "Analyzing Communication Models for Distributed Thread-Collaborative Processors in Terms of Energy and Time", in International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, 2015
[4] NERSC, "Characterization of the DOE Mini-apps." Retrieved July 14, 2016 from https://portal.nersc.gov/project/CAL/doe-miniapps.htm.
[5] M. Steinberger, M. Kenzel, B. Kainz, and D. Schmalstieg, "Scatteralloc: Massively parallel dynamic memory allocation for the gpu," 2012 (cit. on p. 35).