HORIZON 2020
TOPIC FETHPC-02-2017
Transition to Exascale Computing

Exascale Programming Models for Heterogeneous Systems
801039

D2.1
Report on Current Landscape for Novel Network Hardware and Programming Models

WP2: Programming high-performance communication networks and fabrics in heterogeneous systems

Date of preparation (latest version): 31/1/2018

DOCUMENT INFORMATION

Deliverable Number	D2.1
Deliverable Name	Report on Current Landscape for Novel Network Hardware and Programming Models
Due Date	31/1/2018 (PM 5)
Deliverable lead	UEDIN
Authors	Dan Holmes (UEDIN) Timo Scheider (ETH) Carsten Lojewski (FhG) Valeria Bartsch (FhG) Luis Cebamanos (UEDIN)
Responsible Author	Dan Holmes (UEDIN) e-mail: `d.holmes@epcc.ed.ac.uk`
Keywords	Heterogeneous hardware programming models distributed-memory extreme scale
WP/Task	WP2/Task 2.1
Nature	R
Dissemination Level	PU
Planned Date	31/1/2018
Final Version Date	31/1/2018
Reviewed by	Gilbert Netzer (KTH), Harvey Richardson (Cray)
MGT Board Approval	YES

DOCUMENT HISTORY

Partner	**Date**	**Comment**	**Version**
UEDIN	12/10/2018	Created survey paper structure	0.1
ETH	22/11/2018	Added initial bibliography references	0.2
UEDIN	30/11/2018	Added structure and layout	0.3
ETH	03/12/2018	Added additional technologies in section 2	0.4
FhG	04/12/2018	Added information about GASPI/GPI-2	0.5
ETH	10/12/2018	Added more about network capabilities	0.6
UEDIN	13/12/2018	Added capabilities of ConnectX5 and BX	0.7
Cray	13/12/2018	Added capabilities of Cray networks	0.8
UEDIN	21/12/2018	Added FlexNIC and P4	0.9
UEDIN	04/01/2019	Added information about MPI	0.A
UEDIN	10/01/2019	Added challenges section	0.B
UEDIN	11/01/2019	Added deliverable header pages	0.C
FhG	30/01/2019	Modifications in response to internal reviews	0.D
UEDIN	31/01/2019	Modifications in response to internal reviews	0.E
KTH	31/01/2019	Final cleanup, final version	1.0

Executive Summary

This deliverable surveys existing and novel HPC networking hardware and software components and technologies, focusing on heterogeneous hardware capabilities and the challenges this new dimension poses to established Exascale-ready programming models for extreme-scale communication. The deliverable is in the form of a publishable survey paper and targeted at the state-of-the-practice track of the SC19 conference. This format was chosen to maximise the opportunity for dissemination to the broader HPC community. Submission of this paper to SC19 will also illicit, via peer review by other world-leading experts in the field, valuable critique of the assumptions and justifications for the work we intend to carry out in WP2 of the EPiGRAM-HS project. The paper analyses the state-of-the-art in high performance networking technologies (both hardware and low-level software capabilities). It then highlights gaps in the two standardised Exascale-capable programming models (MPI and GASPI) that might inhibit the full exploitation of novel networking technologies. Potential solutions to the gaps are also presented and discussed in the paper. These potential solutions form the basis of the ambition plan for WP2. The focus of the EPiGRAM-HS Consortium in WP2 will be on the integration of compute-in-network hardware and communicate-from-anywhere abilities into MPI and GASPI to support HPC applications and workflows.

Key technologies that exploit heterogeneity in network hardware include programmable network devices, such as Mellanox Scalable Hierarchical Aggregation Protocol (SHArP) and ConnectX-6 hardware. This presents a challenge to programming models of how to expose programmable routing and arbitrary compute-in-network capability to users. In the paper, we suggest using and extending the persistent collective operations interface in MPI as one possible solution for arbitrary compute-in-network and channels or streams with 'build-your-own-collective' functionality to take advantage of programmable routing. This will be the focus of UEDIN with contributions from other project partners.

Network capabilities are also exposed and used by heterogeneous accelerator devices; this may involve connecting heterogeneous compute devices via a 'normal' network, or via a specialised network such as NVLink and NVSwitch. This presents a challenge to programming models of defining the granularity of communication units. For MPI, this means deciding at what level MPI ranks should be assigned, e.g. 1 per node, 1 per device, 1 per NUMA region, 1 per core, or even 1 per individual compute thread. One possible solution is to attempt to run (some portion of) MPI natively on the accelerator device so that it can engage in communication completely independently of the host CPU. This will be the focus of ETH with contributions from other project partners.

Challenges to Extreme-Scale Distributed-Memory Programming Models from Heterogeneous Hardware for HPC

Daniel J. Holmes
EPCC, The University of Edinburgh
Edinburgh, Scotland, UK
d.holmes@epcc.ed.ac.uk

Timo Schneider
ETH Zurich
Zurich, Switzerland
timos@inf.ethz.ch

Carsten Lojewski
CC-HPC, Fraunhofer ITWM
Kaiserslautern, Germany
carsten.lojeswki@itwm.fhg.de

Valeria Bartsch
CC-HPC, Fraunhofer ITWM
Kaiserslautern, Germany
valeria.bartsch@itwm.fhg.de

Luis Cebamanos
EPCC, The University of Edinburgh
Edinburgh, Scotland, UK
l.cebamanos@epcc.ed.ac.uk

ABSTRACT

Modern HPC systems are heterogeneous: nodes include multicore CPUs alongside GPUs and specialized devices, such as FPGAs or other accelerators, such as Tensor Cores or TPUs. Furthermore, with the end of Dennard scaling in conjunction with Moore's law, networks are becoming more powerful, i.e., modern networks often also include capabilities to perform computations "in-network", a feature intended to allow offloading of group communication. In this paper, we survey the landscape of existing programming models for such systems, point out shortcomings with regards to their treatment of heterogeneous hardware, and provide suggestions for extensions and improvements.

ACM Reference Format:
Daniel J. Holmes, Timo Schneider, Carsten Lojewski, Valeria Bartsch, and Luis Cebamanos. 2019. Challenges to Extreme-Scale Distributed-Memory Programming Models from Heterogeneous Hardware for HPC. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC19).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Extreme-scale parallelism requires efficient communication mechanisms. Delivering performance to a wide range of applications on modern HPC systems requires a fast MPI library implementation. MPI exposes useful communication primitives, directly corresponding with well-known data-movement patterns. Many modern network hardware components intended to target HPC systems have features designed to accelerate particular MPI operations and semantics. Increasingly, innovations in hardware capabilities, in a variety of functional areas, are challenging the concepts and abstractions defined by middleware such as MPI. In addition, the contemporary resurgence of interest in reconfigurable computing hardware (FPGAs) is fundamentally challenging accepted ubiquitous traditional programming models, for example the message-passing model that underlies MPI. Section 2 reviews the relevant capabilities of modern network devices currently used in HPC systems and those in publicly available vendor road-maps. Section 3 describes the state-of-the-art for using these hardware capabilities to optimize candidate programming systems for extreme-scale communication, specifically MPI and GASPI. Section 4 sets out the current challenges to popular programming models, with suggested solutions and directions for future research.

2 CAPABILITIES OF DISTRIBUTED-MEMORY COMMUNICATION NETWORK HARDWARE FOR SUPERCOMPUTERS

This section first gives an overview of desirable capabilities of HPC networks (both hardware components and low-level software APIs) and then summarizes commercially available HPC interconnects, as well as research prototypes that fill gaps in the feature-space. This overview assumes as common-knowledge that HPC networks must deliver high bandwidth and low latency; it therefore focuses on additional important characteristics, including: overlap, zero-copy, OS bypass, application bypass, cache-injection, RDMA, path diversity, triggered operations, quality of service, and congestion control. A quantitative comparison (for example, of achieved message rate) is not the focus of this survey, because such a comparison would be biased towards more recent products which take advantage of general progress in clock-speeds and fabrication technology. Instead, we focus on a qualitative comparison, i.e., what novel features a specific network hardware or software API offers.

2.1 HPC Network Capabilities

One of the key techniques to achieve high scalability on HPC workloads is **overlap of computation and communication**: Since network latency cannot be improved beyond the limits of physics, it is important that latency can be hidden by overlapping it with computation the nodes can perform while waiting for data to arrive [25]. Overlapping communication and computation not only hides latency inherent to the transmission of information but also mitigates the problem of OS-Noise for large scale applications [26]. To allow overlap of computation and communication the API to send and receive data must be non-blocking, e.g., a recv() call immediately returns, and a separate call can be used to check if the receive buffer is filled completely.

A network that allows **zero-copy** data transfers provides the capability to transmit data out of a user-space buffer via the network into a different user-space buffer without creating another copy of the data in a different memory buffer i.e., in kernel-space. Zero-copy transfers allow for lower latency (since the time for additional copies is saved), do not pollute the CPU cache unnecessarily and do not spend CPU cycles on copying data, thereby increasing the possibility for computation overlap.

Since HPC applications run as user processes on most systems, but network cards are hardware shared among all such processes on a node, virtualized by the kernel, early network APIs, e.g., the POSIX Sockets API, required a context switch to kernel mode for every transmission or reception of a message, which increases latency. Therefore **OS bypass** [54] has been developed. A network that supports OS bypass can transmit or receive messages without requiring a switch to kernel mode.

Even with OS bypass, some situations still require a progress engine. For example, regular calls to MPI_Test() might be needed in order to progress large messages (where the message data is too large for the bypass mechanism or network hardware to handle in a single operation) or group communication (which typically involves multi-stage protocols that may not be directly supported in the network hardware). When data is exchanged using a rendezvous protocol, the receiver needs to signal readiness to receive a message and the sender must poll for the arrival of that notification. When a reduction is executed which is implemented as a tree, and the communication library has to poll several times to ensure the data from all children has arrived. This can either be done in a separate progress thread, or the computation which overlaps the communication has to be split into smaller blocks, and a call into the progress engine is made between blocks [24]. Both of these software approaches can be avoided by **offloading** these tasks to the NIC itself, when it supports such capabilities. This technique is referred to as **application bypass** [5, 9].

Another technique to reduce latency is **cache-injection** [35], which is based on the assumption that data which is received by the NIC will be accessed by the CPU shortly after. Thus, instead of depositing the received data into main memory it is placed into the CPU cache instead, which is faster than writing to main memory, but also allows the CPU to read the received data with lower latency and higher bandwidth.

The **RDMA** mechanism allows direct access to data in remote memories across an interconnection network. It is similar to machine-local DMA (Direct Memory Access), both mechanisms access memory without involving the CPU or OS. Similarly to DMA, the OS controls protection and setup but after that is not part of the data path. In order to transmit data using RDMA the initiator has to know both the local address of the data to transmit as well as the remote address where the data should be placed. This network feature matches well with the **One-sided** or **RMA** communication model, where the initiator of a communication request, i.e., a Put() or Get(), needs to specify both the source and destination address of the data and there is no implicit synchronization. Message passing can be implemented on top of RDMA networks. The required *matching* of send() to recv() calls can be performed in software on the CPU or it can be **offloaded** to the NIC.

While the techniques presented above mainly addresses the issue of either hiding or improving network latency, and mainly concerns the individual NICs and communication libraries, there are other important features of networks that greatly affect scalability. HPC networks can be categorized into two categories based on the topology used, **direct** and **indirect** networks, which refers to the way how individual nodes are connected to the rest of the network. In indirect networks each node is connected to a switch, whereas direct networks are switchless (or a low-radix switch is integrated on the NIC itself, e.g., for torus networks).

In many HPC network topologies there exists more than one path between some pairs of nodes. Exploiting this **path diversity** poses challenges for the network: If individual packets can be routed onto different paths, even though they belong to the same message, the receiver might need to reorder packets which requires more memory at the receiver.

2.2 Network hardware components

2.2.1 Cray Gemini. Gemini [3, 31] is the 6th generation of scalable HPC networks developed by Cray. Gemini was designed to deliver high performance on MPI applications and filesystem traffic and provides hardware support for PGAS languages such as Chapel, UPC, and Co-Array Fortran on massively parallel systems. Gemini uses 3D torus networks that can scale to in excess of 100,000 multi-core nodes. Each Gemini ASIC provides two network interface controllers (NICs), and a 48-port router. Each of the NICs has its own HyperTransport 3 host interface that enabled Gemini to connect two AMD Opteron nodes directly to the network. Each Gemini NIC can also transfer 64 bytes of data in each direction every 5 cycles. Thus the maximum bandwidth per direction is 8.3 GBytes/s. The interface is 16 bits wide and transfers data on both edges of the clock, giving a raw bandwidth of 9.6 GB/sec in each direction at 2400 MHz. Latency in a Gemini network is determined by the end point latencies and the total number of hops. On a quiet network, the end-point latency is 700 nanoseconds for a remote PUT and 1.5 microseconds or less for small MPI messages. Each of the Gemini torus connections is comprised of 12 lanes in each direction operating at 3.125 to 6.25 GHz. Link bandwidths are 4.68 to 9.375 GBytes/sec per direction.Bandwidth after protocol is 2.9 to 5.8 GB/sec per direction. Message packets are distributed over all of the available links, enabling a single transfer such as a point-to point MPI message, to achieve bandwidths over 5 GBytes/s. Processes on the same node or processes on node connected directly to the same Gemini are able to achieve even higher bandwidth rates.

2.2.2 Cray Aries. Aries[30] is a high performance computing optimized interconnect developed by Cray and it presents a enhanced version of the Gemini device. Aries is a SoC device which integrates 4 NICs, a 48-port router and a multiplexer (Netlink) which provides dynamic load balancing. The Aries router connects 8 NICs to 40 network ports, operating at rates of 4.7 to 5.25 GB/s per direction per port. Each of the four Aries NICs provides an independent 16X PCI-Express Gen3 host interface. The Aries NIC can perform a 64-byte read or write every five cycles (10.2 GB/s at 800MHz). This number represents the peak injection rate achievable by user processes. The Cray Aries interconnect supports the Dragonfly topology used in the Cray XC systems. Dragonfly is a direct network, therefore avoiding the need for external top switches and reducing the number of optical links required for a given global bandwidth.

2.2.3 Cray Slingshot. On October 30 2018, Cray announced that their next generation Shasta system would incorporate a new exascale-era network called Slingshot. Although the systems are not expected to be available until later in 2019 some details of the network are available at the time of writing. Slingshot continues the tradition of the previous Cray networks in providing a high-radix switch architecture and a low-dimension Dragonfly topology. Limited technical information is currently available, the most detail being from a blog [53] by Steve Scott on the Cray website. The new network will be compatible with Ethernet, use advanced adaptive routing and provide new congestion control and quality of service features. The switch architecture can provide 12.8 Tb/sec/direction from 64 200Gbps ports. The network is designed to support a quarter of a million endpoints with up to three network hops, the last utilising optical connections. The new congestion control features are designed to support mixed workloads to minimize the interference between workloads that all networks are susceptible to when dealing with particular communication patterns. Such interference often manifests itself in long tail latency that can have a significant effect on latency-sensitive or synchronization-heavy applications. Slingshot provides traffic classes that can be configured to prioritise specific aspects of network performance required by workloads. Ethernet (and IP) compatibility means that the high-performance network can interoperate directly with storage and beyond the HPC system. The network does provide important features that one would expect of an HPC interconnect such as smaller packet sizes, reliable delivery and synchronization primitives that can be used within the Shasta system.

2.2.4 Mellanox ConnectX series. ConnectX-5 [41] is a low latency and high message rate EDR 100 Gb/s Infiniband and 100 Gb/s Ethernet network interface developed by Mellanox Technologies. ConnectX-5 was designed to connect with almost any computing infrastructure including x86, POWER, FPGAs, GPUs and ARM and offers improvements to HPC infrastructures by providing MPI and SHMEM/PGAS offload. ConnectX-5 enables MPI Tag Matching and MPI AlltoAll operations, as well as advanced dynamic routing [40]. Furthermore, ConnectX-5 supports GPUDirect and Burst Buffer offload without interfering in the main CPU. Mellanox ConnectX architecture has advanced scheduling engines which can assign processing duties such as protocol processing to idle processing elements. The scheduling of packet processing is done directly in hardware, without firmware involvement [56].

Mellanox has recently announced a new product in the ConnectX series, the ConnectX-6 [39].The latest Mellanox addition includes a new generation of 200GbE and HDR Infiniband adapters.s ConnectX-6 is the first adapter to deliver 200Gb/s HDR InfiniBand, 100Gb/s HDR100 InfiniBand and 200Gb/s Ethernet speeds. This makes ConnextX-6 adapters as a main contender to lead HPC data centers toward Exascale levels of performance and scalability.

2.2.5 NVIDIA NVLink. NVLink [15] is a point-to-point interconnect which allows connection between GPUs or between GPUs to CPUs. Each bidirectional link (multiple links can be aggregated if desired) offers 19.2 GB/s of bandwidth. NVLink allows DMA transfers from pinned memory (transfers from pageable memory are much slower). NVLink supports load/store semantics (as well as atomics) to a unified address space of local GPU, remote GPU and system memory (if the host CPU supports NVLink). The NVlink GPUs are connected through the NVSwitch [2]. NVSwitch is an NVLink switch chip with 18 ports of NVLink per switch. Internally, the processor is an 18 x 18-port, fully connected crossbar. The crossbar is non-blocking, allowing all ports to communicate with all other ports at the full NVLink bandwidth. Any port can communicate with any other port at full NVLink speed, 50 GB/s, for a total of 900 GB/s of aggregate switch bandwidth. The NVLink technology is used in the DGX-2 [47] system as well as the Summit [45] supercomputer. Summit is a 200 PFLOPS system made of a hybrid architecture; each node contains multiple IBM POWER9 [32] CPUs and NVIDIA Volta V100 [46] GPUs all connected together with NVIDIAâĂŹs high-speed NVLink.

2.2.6 Tofu. Tofu [1] is a 6D torus network that was developed for the K computer [17], but will also be used in the Post-K Computer [18]. An InterConnect Controller (ICC) chip connects to the SPARC64 host processor using a point-to-point link, and to remote nodes using four separate network interfaces. Tofu supports RDMA to user-space addresses and provides hardware acceleration for the necessary address translation. Tofu utilizes 6D topology and virtual cut-through routing to minimize the impact of the high network diameter (illustrated in Figure 1) of torus networks. The Tofu network chip also includes the Tofu Barrier Interface (TBI), which offloads Barrier, Broadcast, Reduce and Allreduce operations. It supports AND, OR, XOR, MAX and SUM operators for integers and SUM as floating point reduction operators on a single number.

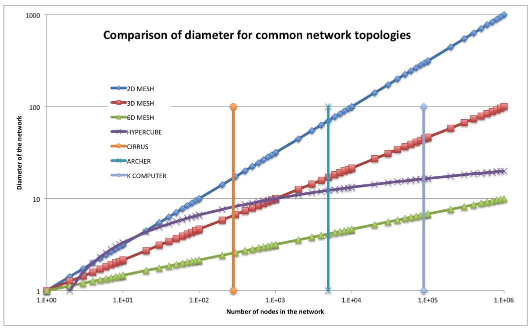

Figure 1: Network topology diameter

2.2.7 BXI. BXI, Bull eXascale Interconnect [12], is a highly scalable interconnection network developed by Atos. BXI is based on Portals 4 [5] and features high speed links (100Gb/s), high message rates (>100 M msg/s) and low latency. This enables the BXI network to scale up to 64,000 nodes. The BXI NIC interfaces with the system node through a 16x PCI Express gen3 interface and with the BXI fabric through a 100Gb/s BXI port (4 lanes at 25Gb/s). The BXI network provides hardware acceleration for the MPI Send/Recv (two-sided) semantics, RDMA (single-sided) operations and collective operations. Upon reception of a particular message, the data is copied to the target memory location without involving the host processors, thus allowing computational resources to be freed from communication tasks.

2.2.8 *Quadrics QSNet.* Quadrics QSNet [50] consists of the 8-port Elite switch and the Elan NIC. QsNet fabrics are built as quaternary fat-trees. The Elan NIC contains a programmable SPARC processor attached to 64 MB of memory and a hardware RDMA engine. QSNet has been used to offload parts of an earlier version of the Portals API [48]. Even though the QSNet NICs are no longer available, they are included in this survey because they incorporate a couple of resurfacing ideas in the NIC design space: their architecture and flexibility is close to sPIN.

2.2.9 *EXTOLL.* The EXTOLL network [36] extends the HyperTransport (HT) protocol such that more than the usual 32 endpoints per fabric can be addressed by adding a routing mechanism. This enables communication with remote HT devices with very low latency, since no protocol conversion is required. It is implemented using Open Source Verilog cores and an FPGA as a prototyping platform. Since its announcement in 2007 many improved versions have been presented, for example it is possible in newer versions [44] to use PCIe instead of HT and to send PCIe packets to remote devices without involvement of any host CPU. This allows direct accelerator-to-accelerator communication. More recently, the Tourmalet ASIC [21] offers higher-performance via a PCIe attached network interface card.

2.3 Low-level network software APIs

2.3.1 *SHArP.* SHArP [22] technology was designed to offload collective operation processing to the network. SHArP defines a protocol for reduction operations, which is instantiated in a MellanoxâĂŹs SwitchIB-2 device and provides support for small data reduce and allreduce,and for barrier. The protocol optimises the global operations by performing the reduction operations on the data as it traverses a reduction tree in the network, reducing the volume of data as it goes up the tree, instead of using a CPU-based algorithm where the data traverses the network multiple times. SHArP enables the the manipulation of data while it is being transferred in the network, instead of waiting for the data to reach the CPU to operate on this data.

2.3.2 *sPIN.* sPIN [23] is a proposed offload-interface and architecture for network-offload capable RDMA NICs. It is based around the observation that virtually all HPC networks packetize transmitted data, and that each packet can be handled individually by a *Header Processing Unit* (HPU) to allow for a high level of parallelism and thus allow each individual HPU to be a relatively power efficient general purpose compute core. Thus the HPUs can be programmed in C.

2.3.3 *P4.* P4 (Programming Protocol-independent Packet Processors) [8, 10] is a domain-specific language designed for programming how packets are processed by the data plane of a programmable forwarding element such as a hardware or software switch, a NIC or a router. Specific targets for P4 include FPGA components attached to network interface cards, such the Xilinx Virtex UltraScale+ FPGA in the NFB-200G2QL SmartNIC from Netcope Technologies [63] and programmable network switches, such as the Barefoot Tofino [7]. P4 is packet-processing DSL (rather than a general language intended to express arbitrary code) that allows the programmer to decide how the forwarding plane processes packets without worrying about implementation details. A compiler transforms an imperative program into a table dependency graph that can be mapped to many specific target devices, including optimized hardware implementations. P4 is designed to specify only the data plane functionality of a network device and P4 programs also partially define the interface by which the control plane and the data-plane communicate.

2.3.4 *Portals 4.* Portals 4 [5] is a network API developed by Sandia National Laboratories which newer versions have benefited from experience gained from ASCI Red, Red Storm and Cray XT systems and Atos BXI [13]. The basis for Portals 4 are non-blocking RMA operations (Put/Get), as well as a set of atomic compute operations on local memory. Portals 4 defines *Triggered Operations* which are a novel concept in the space of network offload: Counters can be attached to different types of events, such as bytes transmitted to a specific memory location, once a counter reaches a threshold value, an operation (such as a Put or Get) is executed. It has been demonstrated that many collective communication schemes can be offloaded to the NIC using this interface [61]. An important difference between standard RMA operations and Portals 4 communication primitives is that Portals 4 supports tag matching in conjunction with RMA: In addition to the data to be transferred, a Put operation can specify a vector of match-bits to be sent along with the data, and the data is placed at a memory location on the receiver only if the receiver posted a match list entry with a matching set of match-bits. While this feature allows for offload of MPI tag matching, and even allows for partial offload of the MPI rendezvous protocol, it can also be leveraged to implement more complex distributed data-structures, such as distributed hash tables [34]. Portals 4 allows for OS as well as application bypass (communication progress can be made independently of the host node) and supports complete offload of MPI collective operations. User-defined collective operators need to be translated to a sequence of Portals 4 triggered operations. Until now, no automated approach exists to perform this task.

2.3.5 *FlexNIC.* FlexNIC [4, 33] is a flexible DMA programming interface for network I/O. FlexNIC allows applications and OSes to install packet processing rules into the NIC to obtain fine-grained control over how data is exchanged. This idea assumes that the model can be implemented in an inexpensive way based on the fact that OpenFlow switches have demonstrated this capability with minimum sized packets [55]. FlexNIC allows for the flexible offload of performance packet processing functionality, while allowing the programmer the flexibility to keep the rest of the application code on the CPU.

3 STATE-OF-THE-ART OF DISTRIBUTED-MEMORY PROGRAMMING SYSTEMS FOR SUPERCOMPUTERS

This section carefully examines two standardized distributed-memory programming models that are likely to be used for Exascale systems, i.e. MPI and GASPI/GPI-2.

Traditionally, HPC systems are viewed by programming models as homogeneous, where all processors have equal (or equivalent) access to all local memory locations and to all local network connections. Increasingly, this simple model is inaccurate for complex systems. Some adjustments have already been made to accommodate heterogeneous hardware in modern HPC systems. Typical examples of these existing adjustments include hierarchical collective operations to optimize for non-uniform memory access (NUMA) regions within a local node and permitting memory references that refer to memory location in GPU memory when calling MPI functions on the host CPU, which can be implemented using GPUDirect (if available) or by copying data to/from the host memory. However, as this section describes, hardware becoming more heterogeneous exposes gaps in the conceptual framework underpinning these programming models.

3.1 Requirements of MPI

The MPI Standard has been the ubiquitous expression of the message-passing programming model for 25 years. There have so far been three major versions MPI-1 [60], MPI-2 [20], and MPI-3 [42]. Several MPI libraries implement this standard, including some open source projects and various commercial or propriety offerings, mostly from HPC system vendors. The most recent version of the MPI Standard includes a wide range of functionality to support HPC applications and many features aimed at a broader community of users. The primary use-case is for distributed-memory architecture parallel systems with many processors and some form of high-performance interconnect.

MPI functionality can be broadly categorized into communication (movement of data between memory locations), process management (start-up, shutdown, creation, connecting), and virtual organization (topologies, groups, and so on). Communication can be further broken down into point-to-point, single-sided, collective, and I/O operations.

3.1.1 Point-to-point communication operations in MPI. The most commonly used communication operations in MPI are the point-to-point send/receive operations. The distinguishing feature set for these operations is the rules governing how sends are matched with receives. All messages include ‘envelope’ information: communicator, source rank, destination rank, and tag. Matching requires that these fields be identical (or be replaced by a wild-card value, only supported for source and tag, and only available at the receiver). When multiple send operations issued by a single thread can match a single receive operation (or vice versa), the order of issuing the operations determines the order of matching. When multiple send operations issued by distinct threads can match a single receive operation (or vice versa), then the matching order is non-deterministic. This flexibility is challenging to implement efficiently; several simplifications have been proposed to exploit additional optimization opportunities [14]. This complexity is also a target for offloading to specialized hardware that combines limited computational and buffering capabilities into a network device.

3.1.2 Single-sided communication operations in MPI. Less commonly used communication operations in MPI include the single-sided put/get/accumulate operations. The distinguishing feature set for these operations is the separation of data movement and the synchronization required for correctness. For ‘active target’ single-sided operations, both the origin and the target processes are required to call synchronization functions. For ‘passive target’ operations, only the origin process is required to issue MPI commands in order to guarantee that the data has moved correctly. The lack of explicit notification of data movement at the target process can be a challenge for the producer-consumer communication pattern. This category of MPI functionality has direct competition from other RDMA-based programming systems, such as GPI-2/GASPI (see section 3.2).

3.1.3 Collective communication operations in MPI. The communication pattern for many application algorithms can be conveniently expressed as a single collective operation rather than many point-to-point or single-sided operations. The distinguishing feature set for these operations is the holistic description of a multi-stage communication pattern. The challenge for these operations is to improve on the performance of the equivalent set of point-to-point or single-sided operations, for example, by careful routing and scheduling of messages, or by combining messages. In addition, MPI includes collective operations, such as reduction and prefix scan, that involve combining or aggregating data values during the operation. The challenge for these aggregation operations is to perform the computation in the most efficient manner, which is a good candidate for compute-in-network capabilities.

3.1.4 I/O communication operations in MPI. The I/O communication operations in MPI are used to implement commonly used I/O libraries, such as HDF5 [59]. The data movement part of their functionality can be implemented efficiently using other MPI communication operations, such as collective communication. Alternatively, the single-sided communication interface can be used for I/O operations, for example, by providing memory that is mapped to/from a file [51]. Whichever approach is used, the main issues are the depth of typical I/O software stacks, where kernel-bypass is a recent innovation [62], and managing the non-volatile nature - commonly expressed with POSIX file-system semantics - and the expectation that non-volatile storage hardware devices operate at a slow speed relative to volatile memory devices. One key design decision for MPI is how to expose novel non-volatile memory devices to programmers - either via memory semantics (using single-sided communication operations) or via file semantics (using I/O communication operations). However, the trade-offs of performance and ease-of-use are not yet well-understood.

3.1.5 Blocking, nonblocking, and persistent semantics in MPI. MPI includes nonblocking versions of most performance-critical functions in order to encourage overlap of communication with other useful work, e.g. computation or other communication. MPI also includes persistent operations, which codify a planning step and encourage re-use of the plan for many identical subsequent instances of the communication operation. Persistence has recently been extended to apply to call collective communication operations [29, 43], in addition to just the point-to-point operations. This change to MPI, which will be included in the next official release of the MPI Standard, presents an opportunity to improve the implementation of collective communication operations in MPI libraries. Exposing an explicit method for the programmer to provide information to MPI that particular operations will be re-used in the application permits different design and optimization trade-offs within MPI implementations. For example, optimizations that would usually be avoided because of a high overhead become viable because the high cost will be amortized over many uses of the operation. Also, hardware resources could be reserved for exclusive use by a single persistent operation, which avoids delays due to contention for shared resources - as long as sufficient resources exist to continue support for other MPI functionality.

3.1.6 *Topologies, groups, and 'enclaves' in MPI.*

MPI has provided representations of virtual topology and facilities to express groups of communicating processes since its first version, MPI-1. These features are intended to support advanced mapping of hardware resources to aid optimization and to support resource isolation for parallel libraries built on top of MPI. However, MPI provides little information about the actual layout and connectivity of the hardware devices within the HPC system. External mechanisms, such as hwloc [57] and netloc [58], can help fill this gap but these tools assume a hierarchical hardware topology with distinct layers. More complex and heterogeneous HPC systems make use of hardware topologies that cannot accurately be represented as a hierarchical tree. In addition, the resource isolation offered by MPI still relies on a central component in each MPI process that initializes MPI and disseminates MPI objects to all parallel libraries. Optimizing MPI for heterogeneous hardware may require different levels of thread support or different implementations of the internal progress engine, even within a single MPI process because it may span several very different hardware devices. Key issues in this space are how to permit resource sharing whilst managing resource contention and congestion. That may be addressed by the proposal to introduce 'sessions' into MPI, which define enclaves of hardware resources accessible only by certain software components [27].

3.2 Requirements of the GASPI specification and implementation details of the GPI-2 implementation

We describe the relevant components of the GASPI specification in section [19] and then describe the requirements on the hardware which derive from the specification and the implementation details described in section [16]. The European GASPI standard is a Partitioned Global Address Space (PGAS) API. It initates a paradigm shift from bulk-synchronous two-sided communication patterns towards an asynchronous communication and execution model. To that end, GASPI leverages remote completion and one-sided RDMA driven communication. The asynchronous communication allows to overlap communication and computation.

3.2.1 *Abstract overview over the GASPI specification.*

The GASPI API is semantically very similar to the asynchronous MPI communication commands. The main design idea of GASPI is to have a lightweight API ensuring high performance, flexibility and failure tolerance.

Memory segments in GASPI. The global memory can be accessed by other nodes using the GASPI API and is divided into so-called segments. A segment is a contiguous block of virtual memory. The segments may be globally accessible from every thread of every GASPI process and represent the partitions of the global address space. The segments can be accessed as global, common memory, whether local - by means of regular memory operations - or remote - by means of the communication routines of GASPI. Memory addresses within the global partitioned address space are specified by the triple consisting of the rank, segment identifier and the offset.

Communication queues in GASPI. GASPI provides the concept of message queues. These queues facilitate higher scalability and can be used as channels for different types of requests where similar types of requests are queued and then get synchronized together but independently from the other ones (separation of concerns, e. g. one queue for operations on data and another queue for operations on meta data). The several message queues guarantee fair communication, i. e. no queue should see its communication requests delayed indefinitely

One-sided communication operations in GASPI. One-sided asynchronous communication is the basic communication mechanism provided by GASPI. The one-sided communication comes in two flavors. There are read and write operations from and into the partitioned global address space. One-sided operations are non-blocking and asynchronous, allowing the program to continue its execution along the data transfer. The entire communication is managed by the local process only. The remote process is not involved. The advantage is that there is no inherent synchronization between the local and the remote process in every communication request. At some point the remote process needs knowledge about data availability managed by weak synchronization primitives.

Notifications in GASPI. In order to allow for synchronization notifications are being used. In addition to the communication request, in which the sender posts a put request for transferring data from a local segment into a remote segment, the sender also posts a notification with a given value to a given queue. The order of the communication request and the notification is guaranteed to be maintained. The notification semantic is complemented with routines which wait for an update of a single or even an entire set of notifications. In order to manage these notifications in a thread-safe manner GASPI provides a thread safe atomic function to reset local notification with a given identifier. The atomic function returns the value of the notification before reset. The notification procedures are one-sided and only involve the local process.

Collectives in GASPI. Collective operations are operations which involve a whole set of GASPI processes. Collective operations can be either synchronous or asynchronous. Synchronous implies that progress is achieved only as long as the application is inside of the call. The call itself, however, may be interrupted by a timeout. The operation is then continued in the next call of the procedure. This implies that a collective operation may involve several procedure calls until completion. We note that collective operations can internally also be handled asynchronously, i.e. with progress being achieved outside of the call. Beside barriers and reductions with predefined operations, reductions with user defined operations are also supported via callback functions.

3.2.2 Implementation details of the GPI-2 implementation and hardware requirements. While MPI has several implementations there is only one implementation for GASPI called GPI-2. This allows us to dive deeper into the hardware requirements of the GPI-2 implementation compared to the previous MPI section 3.1. We give an abstract overview of the hardware requirements for GPI-2 that are needed to run parallel GASPI applications without any limitations and with all interface features enabled on modern RDMA networks. The abstraction is given in a form of different memory regions with different properties in terms of cache coherence and visibility. These memory regions are not visible to the end user, but part of the implementation details of GPI-2.

Memory Regions of GPI-2. The four main memory regions used by GPI-2 are: post-op, completion, payload memory, and notification area. The post-op region is often implemented as a ring buffer with write-through semantic. As soon as a special address location is written in each slot, a DMA-Controller-IP is taking over and does the operations (e.g. Get, Put, Collectives) in the background by overlapping with the computation on the CPU or Accelerator. From the perspective of a calling instance this operation is fairly simple and fast. In a multi-threaded environment several Post-op regions are needed to avoid global locking within the communication library. This global locking is for example needed on CRAY's Gemini or Aries networks. Beside the typical parameters like region identifiers, access rights, offsets, communication tokens and payload lengths extra side-channel information must be given in each slot to program the DMA-Engine on the remote side. In a truly one-sided communication paradigm (as used in GASPI) there is no receiver instance on the remote side that initializes and programs the memory controller to accept incoming data. The receiving DMA Engine must be able to do self-programming by using the side-channel information coming in together with the payloads. Infiniband (Mellanox) provides currently the best match concerning hardware supporting the GASPI communication concepts. When the remote-side has finished writing to memory, status information is propagated back to the sender side and written into the completion memory region together with the communication tokens. GASPI provides three mechanisms to poll on completions:

- test (single shot)
- block (poll until completion comes in)
- timeout (poll for a given amount of time)

Beside polling on each separate operation, GASPI allows polling on groups of RDMA-operations. This concept is realized by a software abstraction on top of the post-op region and is visible to the programmer as a GASPI communication queue/channel.

If the application is able to provide perfect load-balancing units for computations and communications, zero latency and a huge amount of bandwidth can be realized in theory. Setting up a GPI communication call costs only three to four local memory operations. Because it is non-blocking the call immediately comes back and does not need to wait for completion. If progress operations are needed to push the data through the network (like it is the case today with Intel's Omnipath [6]), no deterministic behaviour of the communication pattern will be available and the total runtime of a parallel job will most often increase by some ratio.

Implementation of collectives in GPI-2. GASPI supports collective operations. These are handled in the GPI-2 implementation similarly to reads and writes by writing meta data into the post-op region. Beside the parameters given above, addresses of the input and output vectors as well as the built-in or user-defined operation on the vectors must be defined. The current status of the running collective operation is available from the completion ring buffer at any time and allows for non-blocking collectives. If the network is able to do hardware assisted collectives, intermediate status information must be written into the completion ring buffer to fulfill the communication semantic of GASPI. With no hardware support in place, an internal communication tree with offload semantic is used and the collective operation itself happens each time a status is determined from the completion region.

Implementation of notifications in GPI-2. GPI-2 uses a memory driven approach internally and allows compute kernels to use the same mechanism via notification areas. This coalesced view on ongoing and offloaded communications and computations without calling any library functions allows for a performance level that cannot be reached with any other communication library like MPI or others.

Care must be taking when updating bits and bytes within the notification region due the fact that these updates will forecast available/updated data in one of the payload regions. The transport network has to ensure that on cache coherent systems all memory layers (caches, write buffers, reservations stations, completion buffers at the pipeline level etc.) are up-to-date and present the latest view onto the memory. If special local opcodes must be executed locally or remotely to refresh the memory views, active messaging can be used to trigger code sequences via operating systems or daemon processes. Most x86 are aware of reservation stations and reorder buffers transparently with respect to the memory system. However ARM architectures and research architectures (as found in the EuroExa, ExaNoDe, ExaNeSt EC-funded projects) have separate memory control and cache coherence control (cci+).

4 CHALLENGES TO DISTRIBUTED-MEMORY PROGRAMMING MODELS FOR SUPERCOMPUTERS

This section examines how key capabilities offered by heterogeneous network technology can be expressed in programming models so that users and applications can take advantage of them. Some new capabilities can be incorporated into the implementations of the programming model (i.e. into the source code for particular MPI libraries or into GPI-2). However, to take full advantage of some capabilities requires that the programming model expose the new functionality through modifications or extensions of the standardized API.

4.1 Programmable adaptive routing logic in programming models

Basic assumptions when considering how programmable adaptive routing logic can be expressed and used in programming models are that (re)configuring will take significant time and that the resources available will be limited, possibly to a few (or even just one) operation at a time. This suggests that, at least in the short-term, programming models should provide a mechanism for the programmer to specify explicitly which operations are most likely to benefit from expensive optimization using reconfigurable hardware. The recent inclusion of persistent collective communication operations into MPI is one step in this direction. Further steps for MPI are to consider persistent I/O communication operations and even persistent single-sided communication operations. The difference in semantics between the newly introduced persistent collective operations in MPI and the already existing persistent point-to-point operations suggests that a new class of point-to-point operations should be considered that permit planning during the initialization step for the operation. This class of operations express channel or stream semantics, depending on the details of the interface definition. Some work on using stream of messages via MPI show interesting potential performance improvements [28, 49].

In addition, moving beyond the communication operations defined in MPI to permit customized communication protocols extends the idea of planning for performance to encompass the relationships between all the regularly issued MPI operations. Informally referred to as 'build-your-own collective', this would take advantage of current research into run-time configurable protocol definitions, such as session types [37, 38].

4.2 Arbitrary compute-in-network in programming models

Expressing arbitrary compute-in-network in programming models is problematic because of the generality of the capability offered. Within the context of MPI, the scope for compute-in-network functionality is limited to the reduction and prefix collective aggregation operations with user-defined operators. Support for some specific reduction operations already exists in some hardware. However, enabling the execution of arbitrary user-defined operator code on compute-in-network hardware devices (e.g. FPGAs) suggests that cross-compilation will be necessary to target the intended heterogeneous architecture. There is currently no way to specify that a customized operator should be compiled differently to the code that issues the collective MPI aggregation operation or, more generally, the code for the compute-in-node hardware devices (e.g. CPU or GPU).

4.3 Advanced network switch vs. specialized compute node in programming models

Adding ever more capability to network switches increases their utility for a variety of compute-in-network applications. However, advanced network switches present a potential conceptual problem for HPC programming models like MPI; specifically, when should the switch be assigned a rank and treated as an independent MPI process?

For example, it is clear that, for the stated checkpoint/restart use-case, the Network Attached Memory hardware from the DEEP-ER project [11, 52] is intended to be programmed and operated as an advanced Extoll network switch and not an independent MPI process or specialized compute node. However, if the NAM programmable logic is exposed to reconfiguration by the user, the same device could execute arbitrary computation using incoming and locally stored data, as well as create and inject new network communication messages. In this mode of operation, there would seem to be a need to treat the switch as an MPI process and assign it a rank to identify the source of messages it generates and to provide new incoming data to the correct destination switch.

This dual nature can already be seen in MPI, even in homogeneous systems - messages can be transmitted from one MPI process to another via other uninvolved MPI processes, either using store-and-forward or cut-through routing. The intermediate MPI processes do not call any specific MPI function to enable the transit of such messages; from the programming model point-of-view, they operate in the same manner as a network switch.

The key question raised is: what the granularity of MPI processes should be for optimal expression of the hardware capabilities and application requirements. The traditional "flat MPI" model assigns ranks based on computational hardware - one per core. The "hybrid MPI + X" model assigns ranks based on shared-memory domains - one per node or one per NUMA region. Possible future models might assign each GPU and FPGA a different (set of) MPI ranks to their host CPU and might also assign MPI ranks to advanced network switches as well as traditional compute nodes.

4.4 Clean and efficient communication semantics in interconnect layer

GASPI uses one-sided RDMA and in-memory notifications for the communication allowing efficient overlap of communication and computation. Each thread can communicate whilst optimizing the time spent in communication. Such communication principles are not yet optimally supported by the underlying interconnect libraries. Currently the situation concerning these features is quite diverse:

- support of multithreaded communication can be found in some interconnects, but not all. E.g. CRAY employs a process-based endpoint model.
- RDMA is supported by many interconnects (like the CRAY BTE but not the CRAY FMA which is optimized for small messages). RDMA support is important as it means that the CPU is not involved in the data transfer.
- adding notifications to the communication protocols: in order to communicate efficiently, the notifications need to be added to the communication protocol instead of adding to the payload. For small messages the communication time can increase by up to a factor of 2 if the notifications are added to the payload. Some interconnect libraries (like the CRAY FMA for small messages) have added such a support, however some (like the CRAY BTE (RDMA)) have not.

A clean implementation of efficient communication semantics through all layers (starting from interconnect library, to the communication library and the applications) would decrease the communication latencies notably when comparing with state-of-the-art technologies.

5 CONCLUSIONS

Heterogeneous hardware for HPC systems, both in network devices and for other capabilities, presents strong challenges to well-established extreme-scale programming models. The assumption that all basic units within the programming model (e.g. MPI processes in MPI) are homogeneous limits the conceptual representation power of the programming model. For many novel and emerging heterogeneous systems, programmers cannot accurately express the true capability and connectivity, which restricts the opportunity to make effective and efficient use of these systems. As current trends suggest that Exascale systems will rely on hardware heterogeneity to meet power constraints, there is a risk that traditional standardized programming models for distributed-memory HPC applications will not permit applications to scale to the necessary degree without extensions and enhancements.

This paper carefully examines two standardized distributed-memory programming models that are likely to be used on future Exascale HPC machines. Several important issues with the fundamental assumptions in those programming models, and the resultant gaps in the conceptual representation of heterogeneous hardware, are identified and potential solutions involving extensions or improvements to the standardized APIs are suggested.

6 ACKNOWLEDGEMENTS

This work was funded by the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement 801039 (the EPiGRAM-HS project).

All trademarks are the property of the respective owners.

REFERENCES

[1] Yuichiro Ajima, Shinji Sumimoto, and Toshiyuki Shimizu. 2009. Tofu: A 6d Mesh/Torus interconnect for exascale computers. *IEEE Computer* 42, 11 (2009), 36–40.

[2] Alex Ishii and Denis Foley. 2018. NVSwitch and DGX-2 NVLink-Switching Chip and Scale-Up Compute Server. Retrieved 2019-01-31 from http://www.hotchips.org/hc30/2conf/2.01_Nvidia_NVswitch_HotChips2018_DGX2NVS_Final.pdf

[3] R. Alverson, D. Roweth, and L. Kaplan. 2010. The Gemini System Interconnect. In *2010 18th IEEE Symposium on High Performance Interconnects*. 83–87. https://doi.org/10.1109/HOTI.2010.23

[4] Thomas E. Anderson Antoine Kaufmann, Simon Peter and Arvind Krishnamurthy. 2015. FlexNIC: Rethinking network DMA.. In *2015 15th Workshop on Hot Topics in Operating Systems*.

[5] Brian W. Barrett, Ron Brightwell, Ryan E. Grant, Kevin Pedretti, Kyle Wheeler, and et al. 2018. The Portals 4.2 Network Programming Interface. (November 2018).

[6] Mark S. Birrittella, Mark Debbage, Ram Huggahalli, James Kunz, Tom Lovett, Todd Rimmer, Keith D. Underwood, and Robert C. Zak. 2015. Intel®Omni-path Architecture: Enabling Scalable, High Performance Fabrics. In *Proceedings of the 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects (HOTI '15)*. IEEE Computer Society, Washington, DC, USA, 1–9. https://doi.org/10.1109/HOTI.2015.22

[7] Patrick Bosshart. 2018. Programmable Forwarding Planes at Terabit/s Speeds. Retrieved 2019-01-31 from http://www.hotchips.org/hc30/2conf/2.02_Barefoot_Barefoot_Talk_at_HotChips_2018.pdf

[8] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[9] Darius Buntinas, Dhabaleswar K Panda, and Ron Brightwell. 2003. Application-bypass broadcast in MPICH over GM. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*. IEEE, 2–9.

[10] The P4 Language Consortium. 2018. *P4 Language Specification*. Technical Report. Retrieved 2018-12-17 from https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.pdf

[11] DEEP project partners. 2017. DEEP Projects - The European funded Exascale research project DEEP-ER. Retrieved 2019-01-31 from https://www.deep-projects.eu/

[12] S. Derradji, T. Palfer-Sollier, J. Panziera, A. Poudes, and F. W. Atos. 2015. The BXI Interconnect Architecture. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. 18–25. https://doi.org/10.1109/HOTI.2015.15

[13] Saïd Derradji, Thibaut Palfer-Sollier, Jean-Pierre Panziera, Axel Poudes, and François Wellenreiter Atos. 2015. The BXI interconnect architecture. In *High-Performance Interconnects (HOTI), 2015 IEEE 23rd Annual Symposium on*. IEEE, 18–25.

[14] Mario Flajslik, James Dinan, and Keith D. Underwood. 2016. Mitigating MPI Message Matching Misery. In *ISC High Performance 2016: High Performance Computing*. Springer, Cham, 281–299.

[15] Denis Foley and John Danskin. 2017. Ultra-performance Pascal GPU and NVLink interconnect. *IEEE Micro* 37, 2 (2017), 7–17.

[16] Fraunhofer ITWM. 2019. GPI-2 main website. Retrieved 2019-01-31 from http://www.gaspi.de/

[17] Fujitsu Ltd. 2011. K Computer, What is K? Retrieved 2019-01-31 from https://www.r-ccs.riken.jp/en/k-computer/about/

[18] Fujitsu Ltd. 2018. Fujitsu High Performance CPU for the Post-K Computer. Retrieved 2019-01-31 from http://www.fujitsu.com/jp/Images/20180821hotchips30.pdf

[19] GASPI Forum. 2019. Forum of the PGAS-API GASPI. Retrieved 2019-01-31 from http://www.gaspi.de/

[20] Al Geist, William Gropp, Steve Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, William Saphir, Tony Skjellum, and Marc Snir. 1996. MPI-2: Extending the message-passing interface. In *Euro-Par 1996: Euro-Par'96 Parallel Processing*. Springer, Berlin, Heidelberg, 128–135.

[21] EXTOLL GmbH. 2019. TOURMALET (ASIC). Retrieved 2019-01-31 from http://extoll.de/products/tourmalet

[22] Richard L. Graham, Devendar Bureddy, Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, Lion Levi, Alex Margolin, Tamir Ronen, Alexander Shpiner, Oded Wertheim, and Eitan Zahavi. 2016. Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction. In *Proceedings of the First Workshop on Optimization of Communication in HPC (COM-HPC '16)*. IEEE Press, Piscataway, NJ, USA, 1–10. https://doi.org/10.1109/COM-HPC.2016.6

[23] Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E Grant, and Ron Brightwell. 2017. sPIN: High-performance streaming Processing in the Network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 59.

[24] Torsten Hoefler and Andrew Lumsdaine. 2008. Message progression in parallel computing - to thread or not to thread?. In *Cluster Computing, 2008 IEEE International Conference on*. IEEE, 213–222.

[25] Torsten Hoefler, Andrew Lumsdaine, and Wolfgang Rehm. 2007. Implementation and performance analysis of non-blocking collective operations for MPI. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. ACM, 52.

[26] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. 2010. Characterizing the influence of system noise on large-scale applications by simulation. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 1–11.

[27] Daniel Holmes, Kathryn Mohror, Ryan E. Grant, Anthony Skjellum, Martin Schulz, Wesley Bland, and Jeffrey M. Squyres. 2016. MPI Sessions: Leveraging Runtime Infrastructure to Increase Scalability of Applications at Exascale. In *Proceedings of the 23rd European MPI Users' Group Meeting (EuroMPI 2016)*. ACM, New York, NY, USA, 121–129. https://doi.org/10.1145/2966884.2966915

[28] Daniel J. Holmes and Caoimhín Laoide-Kemp. 2016. Streams as an alternative to halo exchange. *Advances in Parallel Computing* 27 (2016), 305–316. https://doi.org/10.3233/978-1-61499-621-7-305

[29] Daniel J. Holmes, Bradley Morgan, Anthony Skjellum, Purushotham V. Bangalore, and Srinivas Sridharan. 2019. Planning for performance: Enhancing achievable performance for MPI through persistent collective operations. *Parallel Comput.* 81 (2019), 32 – 57. https://doi.org/10.1016/j.parco.2018.08.001

[30] Cray Inc. [n. d.]. *CrayÂő XCâĎć Series Network*. Technical Report. Retrieved 2018-12-18 from https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf

[31] Cray Inc. 2010. *The Gemini Network Rev 1.1*. Technical Report. Retrieved 2018-12-14 from https://wiki.alcf.anl.gov/parts/images/2/2c/Gemini-whitepaper.pdf

[32] Jeff Stuecheli and William Starke. 2018. The IBM POWER9 Scale Up Processor. Retrieved 2019-01-31 from http://www.hotchips.org/hc30/2conf/2.12_IBM_

POWER9_HC30.POWER9Cv7.pdf

[33] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. *SIGOPS Oper. Syst. Rev.* 50, 2 (March 2016), 67–81. https://doi.org/10.1145/2954680.2872367

[34] D Brian Larkins, John Snyder, and James Dinan. 2018. Efficient Runtime Support for a Partitioned Global Logical Address Space. In *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 70.

[35] Edgar A León, Kurt B Ferreira, and Arthur B Maccabe. 2007. Reducing the impact of the memorywall for i/o using cache injection. In *High-Performance Interconnects, 2007. HOTI 2007. 15th Annual IEEE Symposium on*. IEEE, 143–150.

[36] Heiner Litz, H Fröening, Mondrian Nuessle, and U Brüening. 2007. A hypertransport network interface controller for ultra-low latency message transfers. *HyperTransport Consortium White Paper* (2007).

[37] Hugo A. López, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, César Santos, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. 2015. Protocol-based Verification of Message-passing Parallel Programs. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2015)*. ACM, New York, NY, USA, 280–298. https://doi.org/10.1145/2814270.2814302

[38] Hugo A. López, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, César Santos, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. 2015. Protocol-based Verification of Message-passing Parallel Programs. *SIGPLAN Not.* 50, 10 (Oct. 2015), 280–298. https://doi.org/10.1145/2858965.2814302

[39] Mellanox. [n. d.]. ConnectX-6. http://www.mellanox.com/related-docs/prod_silicon/PB_ConnectX-6_EN_IC.pdf

[40] Mellanox Technologies. 2018. *ConnectX-5 EN IC*. Technical Report. Mellanox Technologies. http://www.mellanox.com/related-docs/prod_silicon/PB_ConnectX-5_EN_IC.pdf

[41] Mellanox Technologies. 2018. ConnectX-5 Single/Dual-Port Adapter supporting 100Gb/s with VPI. Retrieved 2018-12-13 from http://www.mellanox.com/page/products_dyn?product_family=258&mtag=connectx_5_vpi_card

[42] Message Passing Interface Forum. 2015. MPI: A Message-Passing Interface Standard Version 3.1. Retrieved 2019-01-31 from https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

[43] Bradley Morgan, Daniel J. Holmes, Anthony Skjellum, Purushotham Bangalore, and Srinivas Sridharan. 2017. Planning for Performance: Persistent Collective Operations for MPI. In *Proceedings of the 24th European MPI Users' Group Meeting (EuroMPI '17)*. ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/3127024.3127028

[44] Sarah Neuwirth, Dirk Frey, Mondrian Nuessle, and Ulrich Bruening. 2015. Scalable communication architecture for network-attached accelerators. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 627–638.

[45] NVIDIA. [n. d.]. Introducing the world's most powerful supercomputer. Retrieved 2019-01-31 from https://images.nvidia.com/content/pdf/worlds-fastest-supercomputer-summit-infographic.pdf

[46] NVIDIA. [n. d.]. NVIDIA TESLA V100. Retrieved 2019-01-31 from https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf

[47] NVIDIA. [n. d.]. The IBM POWER9 Scale Up Processor. Retrieved 2019-01-31 from https://www.nvidia.com/content/dam/en-zz/es_em/Solutions/Data-Center/dgx-2/dgx-2h-datasheet-us-nvidia-841283-r6-web.pdf

[48] Ron Brightwell Kevin T Pedretti and Ron Brightwell. 2004. A NIC-Offload Implementation of Portals for Quadrics QsNet. In *Fifth LCI International Conference on Linux Clusters*.

[49] Ivy Bo Peng, Stefano Markidis, Erwin Laure, Daniel Holmes, and Mark Bull. 2015. A data streaming model in MPI. In *Proceedings of the 3rd Workshop on Exascale MPI*. ACM, ACM Press, New York, New York, USA, 2. https://doi.org/10.1145/2831129.2831131

[50] Fabrizio Petrini, Wu-chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. 2002. The Quadrics network: High-performance clustering technology. *Ieee Micro* 22, 1 (2002), 46–57.

[51] Sergio Rivas-Gomez, Roberto Gioiosa, Ivy Bo Peng, Gokcen Kestor, Sai Narasimhamurthy, Erwin Laure, and Stefano Markidis. 2017. MPI Windows on Storage for HPC Applications. In *Proceedings of the 24th European MPI Users' Group Meeting (EuroMPI '17)*. ACM, New York, NY, USA, Article 15, 11 pages. https://doi.org/10.1145/3127024.3127034

[52] Juri Schmidt. 2016. Xilinx Adaptable Intelligent Boards, Kits, and Modules - NFB-200G2QL. Retrieved 2019-01-31 from http://sc16.supercomputing.org/sc-archive/doctoral_showcase/doc_files/drs106s2-file7.pdf

[53] Steve Scott. 2018. Meet Slingshot: An Innovative Interconnect for the Next Generation of Supercomputers | Cray Blog. Retrieved 2018-12-13 from https://www.cray.com/blog/meet-slingshot-an-innovative-interconnect-for-the-next-generation-of-supercomputers/

[54] Piyush Shivam, Pete Wyckoff, and Dhabaleswar Panda. 2001. EMP: zero-copy OS-bypass NIC-driven gigabit ethernet message passing. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*. ACM, 57–57.

[55] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs HÃűlzle, Stephen Stuart, and Amin Vahdat. 2015. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in GoogleâĂŹs Datacenter Network. In *Sigcomm '15*.

[56] Sayantan Sur, Matthew J. Koop, Lei, and Dhabaleswar K. Panda. 2007. Performance Analysis and Evaluation of Mellanox ConnectX InfiniBand Architecture with Multi-Core Platforms. In *15th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2007)*. IEEE, 125–134. https://doi.org/10.1109/HOTI.2007.16

[57] Open MPI Team. [n. d.]. Hardware Locality (hwloc) 2.0.3. Retrieved 2019-01-31 from https://www.open-mpi.org/projects/hwloc/doc/hwloc-v2.0.3-a4.pdf

[58] Open MPI Team. [n. d.]. Portable Network Locality (netloc). Retrieved 2019-01-31 from https://www.open-mpi.org//projects/netloc

[59] The HDF Group. 1997-NNNN. Hierarchical Data Format, version 5. http://www.hdfgroup.org/HDF5/.

[60] CORPORATE The MPI Forum. 1993. MPI: A Message Passing Interface. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing (Supercomputing '93)*. ACM, New York, NY, USA, 878–883. https://doi.org/10.1145/169627.169855

[61] Keith D Underwood, Jerrie Coffman, Roy Larsen, K Scott Hemmert, Brian W Barrett, Ron Brightwell, and Michael Levenhagen. 2011. Enabling flexible collective communication offload with triggered operations. In *High Performance Interconnects (HOTI), 2011 IEEE 19th Annual Symposium on*. IEEE, 35–42.

[62] Daniel Waddington and Jim Harris. 2018. Software Challenges for the Changing Storage Landscape. *Commun. ACM* 61, 11 (Oct. 2018), 136–145. https://doi.org/10.1145/3186331

[63] Xilinx Inc. 2019. Xilinx Adaptable Intelligent Boards, Kits, and Modules - NFB-200G2QL. Retrieved 2019-01-31 from https://www.xilinx.com/products/boards-and-kits/1-pcz51c.html