OMB-FPGA: A Microbenchmark Suite for FPGA-aware MPIs using OpenCL and SYCL

Nicholas Contini*
contini.26@osu.edu
Ohio State University
Columbus, OH, United States

Mustafa Abduljabbar
abduljabbar.1@osu.edu
Ohio State University
Columbus, OH, United States

Hari Subramoni
subramoni.1@osu.edu
Ohio State University
Columbus, OH, United States

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

ABSTRACT

The advancement of traditional CPU architectures is seeing diminished returns due to the slowing of Moore's Law and end of Dennard Scaling. FPGA-based accelerators provide a new path forward for High-Performance Computing. Instead of being confined to Von Neumann architecture and instruction set architectures, reconfigurable hardware allows computational scientists to develop application-specific architectures that can be expressed with common software programming languages thanks to advances in High-Level Synthesis. Work has been done to provide more support to these devices in the HPC community to ease adoption. This includes work to provide MPI implementations that enable data transfers between FPGAs within a compute cluster. However, a standardized benchmarking suite that measures the performance of this functionality does not exist. In this paper, we propose an extension to the OSU-Micro-Benchmarks suite, OMB-FPGA, that enables measuring the performance of MPI point-to-point and collective communication between FPGAs and other communication endpoints.

CCS CONCEPTS

• **Networks** → **Network measurement**; **Network performance analysis**; • **Hardware** → *Networking hardware*; • **Computing methodologies** → *Parallel computing methodologies*; *Distributed computing methodologies*; • **Computer systems organization** → *Reconfigurable computing*; *Heterogeneous (hybrid) systems.*

KEYWORDS

FPGA, MPI, Benchmark, Reconfigurable Computing, High Performance Computing, HPC, Message Passing, SYCL, OpenCL, Heterogeneous Computing

*This research is supported in part by NSF grants #1818253, #1854828, #2007991, #2018627, #2311830, #2312927, and XRAC grant #NCR-130002.

PEARC '24, July 21–25, 2024, Providence, RI, USA

ACM ISBN 979-8-4007-0419-2/24/07
https://doi.org/10.1145/3626203.3670518

ACM Reference Format:
Nicholas Contini, Mustafa Abduljabbar, Hari Subramoni, and Dhabaleswar K. Panda. 2024. OMB-FPGA: A Microbenchmark Suite for FPGA-aware MPIs using OpenCL and SYCL. In *Practice and Experience in Advanced Research Computing (PEARC '24), July 21–25, 2024, Providence, RI, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3626203.3670518

1 INTRODUCTION

With the slowing of Moore's Law and the end of Dennard Scaling, the High-Performance Computing community has reached an era of uncertainty [10]. Increases in computing performance are pursued through non-traditional architectures rather than simply improving the implementation of existing general-purpose processors. The community has achieved much success with the adoption of GPUs and heterogeneous computing frameworks such as CUDA and ROCm [8, 17]. These devices are almost required for Deep Learning workloads for training large models as otherwise computation may take several orders of magnitude longer using a traditional CPU architecture.

However, GPUs only are one foray into new architectural approaches. Multiple types of accelerators have been explored, such as TPUs and DPUs. Perhaps the most unique of these architectures is the Field Programmable Gate Array (FPGA). FPGAs have a long history of being a prototyping platform for digital logic. However, their uses have grown beyond prototyping into accelerating Digital Signal Processing code, implementing hardware "glue code," and playing roles in Embedded Systems. The HPC community has shown interest in leveraging FPGAs since the early 2000s, but their greatest growth only just happened over the past couple of years. This is owed to the advancement of High Level Synthesis (HLS) and an increase in soft logic resources on FPGA chips.

Traditionally, FPGAs are configured using Hardware Description Languages (HDL). These languages, although powerful, are very different from languages that are ubiquitous in the software development community. Skill gained from programming in C, Java, or Python do not necessarily transfer to these HDL languages where lines of code are not necessarily sequential, due to HDL being more of a method to describe circuitry rather than issue instructions. In addition, abstractions enabling dynamic memory management do not exist, and a syntactically and semantically correct code may fail to fully synthesize. This has prompted vendors to invest in HLS tooling to enable software engineers to use familiar software

languages to program FPGAs. Such high level interfaces have been used in various applications to accelerate computation in a highly productive manner [13, 16, 18]. The most notable vendor suites utilizes these higher level development flows are Vitis, a collection of tools supporting programming Xilinx FPGAs using C/C++ and OpenCL, and DPC++, a SYCL implementation by Intel that targets Intel FPGAs as well as GPUs and CPUs.

Furthermore, FPGAs previously had much fewer logic resources, both in terms of variety and quantity. However newer FPGAs targetting HPC and data center workloads have much more LUTs, Flip-Flops, and on-chip memories as well as newer resources such as DSP slices and High Bandwidth Memory (HBM). This has both enabled larger and more performant designs to be configured on the FPGA. Finally, vendors have made an effort to make FPGA accelerators suited for HPC, and prompting computing centers to deploy them in large systems [4].

As these devices become more suitable for HPC workloads, adoption among HPC applications, middleware, and benchmarking suites will be vital for the widespread adoption of these accelerators. There are already multiple projects leveraging these devices, such as HPCC FPGA [13] and neural network applications[11]. However, there is a lack of projects focused on easing the process of scaling out FPGA-based applications in a manner familiar to current HPC developers. One of the few projects targeting this problem is the MVAPICH2-FPGA project [3]. This project enables OpenCL host applications to pass buffer mapped to FPGA memory into MPI routines, relieving application developers of the task of first handling FPGA to host data transfers before initiating MPI communication. Furthermore, the implementation provides optimization at the MPI level that would otherwise be nontrivial or impossible to do at the application level. However, readily available methods to measure the performance of this implementation currently do not exist. Furthermore, as the popularity of SYCL interfaces grows, supporting these interfaces at the MPI-level will be necessary thus requiring microbenchmarking suites to support this as well. Intel has even gone as far as deprecating the Intel OpenCL for FPGAs SDK in favor of SYCL being the only supported API for programming their line of FPGAs.

As one of the most widely used benchmarking suites used to measure MPI performance, the OSU Micro-benchmark (OMB) is an ideal medium to implement micro-benchmarking support for FPGA-aware MPI implementations. By integrating support for FPGAs within OMB, MPI performance can be measured in various configurations supported by the MPI implementation, such as point-to-point and collective communication between FPGAs attached to the same host or FPGAs on separate hosts connected over high-performance networks such as InfiniBand or Slingshot. Users will also be able to test the performance of implementations using different buffer reuse patterns - device-to-host, host-to-device, and device-to-device transfers. Furthermore, since there may be various types of interconnects between different pairs of FPGAs, we propose the ability for users to test various processes to FPGA bindings. Since FPGA accelerators have different types of memory available, We enable OMB users to specify whether buffers should be located on DDR, HBM, or on-chip memory (PLRAM) on the device. Lastly, we support MPI implementations supporting both OpenCL, which is currently used for much of the research involving using FPGAs in HPC, and SYCL, the only high-level interface for programming Intel FPGAs.

In this paper, we detail work done to enable these benefits within the OMB suite. We propose the following contributions:

(1) We extend an MPI implementation to support FPGA-to-FPGA communication through a SYCL interface.
(2) We implement OMB-FPGA with support for both OpenCL and SYCL interfaces, enabling the micro-benchmarking of FPGA-aware MPI-based point-to-point and collective operations supported by emerging implementations.
(3) We provide users with the ability to choose the type of memory used on the FPGA accelerator.
(4) We create benchmarks measuring the performance of point-to-point, pairwise, and collective communications within an FPGA-enabled cluster and present the results collected.

2 BACKGROUND

2.1 FPGAs in HPC

FPGAs are traditionally used in hardware prototyping, signal processing, and circuit "glue code." However, interest in the field of HPC has picked up in recent years thanks to the diversification of on chip resources (such as DSPs and RAMs) and High Level Synthesis programming flows. FPGAs are not constrained to the Von Neumann architecture, where a processor executes a list of instructions and moves intermediate results of computation to and from memories. This makes them a promising choice to implement application-specific acceleration. Instead of loading, executing, and then storing the result of an instruction, a specialized circuit can be developed to enable efficient data locality, feeding outputs of one logical block directly into another. Furthermore, these logical blocks can be duplicated to enable data parallelism within the design. Pipeline parallelism can be implemented by having logical blocks immediately process new inputs after outputting results to another logical block. Many operations can be done using these techniques in a few clock cycles, ultimately increasing computational throughput despite FPGAs having lower clock frequencies than most CPUs. However, very little work has been done to scale out FPGA-based computation compared to other accelerators such as GPUs. Enabling this is still an ongoing research problem.

2.2 OpenCL

OpenCL, i.e. "the Open Computing Language" is an open standard developed by the Kronos group [14]. It's primary purpose it to provide an interface for running computational tasks across a variety of devices, whether they are CPUs, GPUs, ASICs, or FPGAs. Included in OpenCL is a language for writing device kernels as well as host-side APIs for orchestrating data transfers, compiling kernels at runtime, launching tasks, etc. Much previous work on FPGA-based computation utilizes OpenCL in the host application in combination with C/C++-based HLS or OpenCL kernels.

2.3 SYCL

SYCL is another open standard maintained by the Kronos group [6]. Originally developed as an extension to OpenCL, SYCL aims to provide a higher-level abstraction conforming to C++ programming

practices. Its interfaces allow allows data transfers to be inferred by the runtime by establishing dependencies between tasks using buffer objects. Application developers can thus opt to avoid explicitly writing data transfers in their code. However data transfers can still be optimized by using lower level APIs when necessary, such as when communicating between accelerators on separate nodes. Multiple SYCL implementations are noteworthy in relation to FPGAs. For example, Intel's DPC++ implementation supports the use of FPGA accelerators with Intel FPGAs, while TriSYCL aims to provide SYCL interfaces when utilizing Xilinx FPGAs [7, 15].

2.4 MPI

Message Passing Interface (MPI) is the defacto solution for interprocess communication in HPC [12]. It abstracts underlying data transfer mechanisms from the application enabling high productivity as well as high performance, making it popular among the largest computing systems in the world. Without an interface like MPI, applications would be forced to employ many different methods of data transfer, causing a replication of work across applications as well as increasing the potential of writing suboptimal or bug-ridden code. With the correct MPI implementation, applications can avoid interacting directly with high-performance network APIs, shared memory transfers, and host-to-device or device-to-host transfers. For example, an MPI implementation can support transfers over InfiniBand, Slingshot, and RoCE interconnects or GPU to GPU transfers with virtually no changes to an applications communication code [19, 20]. Contini et. al. [3] discusses providing high-level inter-FPGA communication by removing the need for applications to execute OpenCL calls to transfer data from FPGA to host memory while also providing FPGA-specific optimizations within an MPI implementation. In order to support more work in this area, a proper benchmark must be developed to measure the current and future performance of MPI implementations supporting inter-FPGA communication.

2.5 OSU-Micro-Benchmarks

OMB [9] is a communication microbenchmarking suite. It has several benchmarks measuring the performance of point-to-point, multi-pair, and collective communication, and also supports measuring multiple communications libraries, including MPI, NCCL, UPC, and UPC++. Python and Java interfaces to MPI can also be tested. Lastly, it also has added CUDA [1] and ROCm support, enabling users to measure GPU to host, host to GPU, and GPU to GPU latency between two processes. This suite is widely used in the HPC community and enables the comparison of different communication runtimes. By extending support to FPGAs, communication between these devices can be measured and compared with other devices.

3 DESIGN

3.1 Supporting the OpenCL Interface

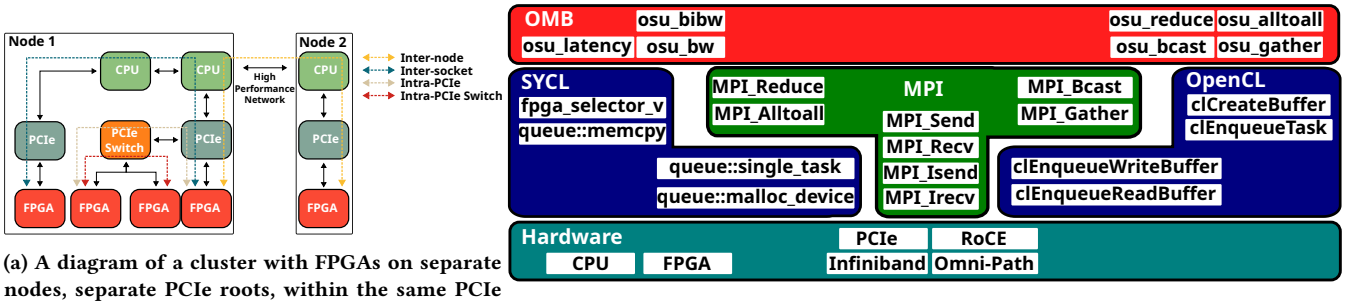
The first step to enabling FPGA support in OMB is creating a way for each benchmark to transparently allocate device buffers. Using OpenCL, one might expect that this is a simple call to *clCreateBuffer*, but in fact, there are many steps required before attempting to allocate a device buffer.

3.1.1 Device Selection. First, the available platforms must be queried using *clGetPlatformIDs*, giving the caller the ability to view the available devices for different vendors using *clGetDeviceIDs*. From here, OMB must have logic to appropriately assign each rank in a benchmark to a proper device. Since most HPC applications use a 1:1 mapping of processes to accelerators, we take the same approach in this work. As a result, the default mapping scheme maps the first device to rank 0, the next device to rank 1, and so on.

However, users may not want to use the default mapping scheme. As shown in Figure 1a, systems may have multiple FPGAs within a single node, so the choice of which FPGA is mapped to a given rank may affect the results of the benchmark. For example, it can generally be expected that messages that must travel between different PCIe root complexes will incur a higher latency than those traveling over the same root complex. It is also possible that some MPI implementations take advantage of P2P transfers provided by Xilinx's OpenCL implementation [3], thus transfers that only travel over a PCIe switch may experience lower latency than transfers that must go through a shared PCIe root complex. Figures 1a demonstrates these different configurations. With this in mind, we provide a *FPGA_MAPPING* environment variable that allows users to specify which FPGA maps to which rank. The expected value of the variable is a colon-separated list of FPGA indices. For example, if the user passes a string such as *2:0:3*, OMB will map the device at index 2 to rank 0, device index 0 to rank 1, and device index 3 to rank 2. By providing this flexibility, users may test an MPI implementation's performance when sending messages over different data paths.

3.1.2 Placeholder Kernels and Buffer Allocation. Even if a device to be used for the benchmark has been selected for a given process, a device buffer still cannot be created. This is because some OpenCL runtimes (such as Xilinx's implementation) require that a device binary be flashed to the FPGA before creating buffers. Since FPGAs have different types of memory and arguments to device kernels might be mapped to specific memories, the runtime cannot determine where device buffers should be physically located until a program is loaded. Our OMB implementation resolves this by creating a device binary containing an placeholder kernel at build time. This binary is programmed to the FPGA card using *clCreateProgramWithBinary* during an initialization step.

With these functionalities done at init time, we can finally wrap calls to 'clCreateBuffer' in a high-level buffer allocation function. However, there still remains a problem. In our testing, we noticed that although with these steps we can successfully complete a basic point-to-point benchmark and pass validation, the measured latency and bandwidth in these tests are impossible to achieve. In theory, a Xilinx U200 card can only transfer data over a PCIe Gen3x16 interface at a bandwidth of 16 GB/s. Furthermore, Xilinx's own *xbutil* only achieves a 12 GB/s read bandwidth and 8 GB/s write bandwidth on our MRI cluster. However, our device to host transfer tests were demonstrating bandwidths of up to 32 GB/s, which is not possible to achieve with the given interconnect. This is due to the OpenCL runtime attempting to skip data transfers between devices and hosts if the runtime thinks that the data in a device buffer has not been updated.



(a) A diagram of a cluster with FPGAs on separate nodes, separate PCIe roots, within the same PCIe root, and within the same PCIe switch.

(b) System architecture of OMB with FPGA support.

Figure 1: Diagrams showing the system of a cluster with FPGAs and the system architecture of OMB with FPGA support

To force OpenCL to execute data transfers and measure the true bandwidth of transferring between device buffers, we added function calls that execute the placeholder kernels mentioned earlier in this section using *clCreateKernel* and *clEnqueueTask*. This kernel contains no computation or I/O. The kernel only takes a device buffer as an argument and has an empty body. This is sufficient enough to cause the OpenCL runtime to believe the device buffers have potentially been altered, and thus device-to-host data transfers will occur. Kernel execution is called before any measured MPI call involving FPGA buffers is made. However, this kernel execution is not included in any latency or bandwidth calculations. This is because there is a kernel launch overhead that could significantly skew the measured results, especially at lower message sizes, and no communication occurs during kernel execution.

3.2 Supporting the SYCL Interface

3.2.1 Device Selection. SYCL provides an abstract class referred to as a selector. The role of the selector is to determine which device to use for a given SYCL command queue. This is generally used to provide a simple way to create command queues specific to GPUs, CPUs, etc. However, users can implement their own selectors to enforce stricter requirements. For example, on a system with multiple GPUs, a user may implement a selector that only uses a specific model, exists in a specific numa domain, etc. In our benchmarks, we specifically use these selectors to implement appropriate mapping.

The selectors provided by SYCL implementations are generally not very robust; if you use an FPGA selector, the selector will simply choose the first FPGA detected. However in the context of OMB, we want each rank to be assigned to a different FPGA. Also, we would like to provide users to provide their own mappings through *FPGA_MAPPING*, just as in the previous section. We implement a selector that will selects the *nth* device, where *n* is either the rank of the current process or a user provided index.

3.2.2 Buffer Allocation and Data Transfers. With this selector implemented, we can instantiate an instance of this selector and pass it to the SYCL queue constructor, thus assigning the *nth* device to be used for the current rank in the benchmark. Upon queue creation we can finally allocate device buffers using *sycl::queue::malloc_device*. These buffers are used throughout the course of the benchmark, making calls to *sycl::queue::memcpy* when necessary to initialize and validate the contents of these buffers.

3.2.3 Placeholder Kernels. For the specific implementation of SYCL that we are using, Intel DPC++, a placeholder kernel is not necessary. It doesn't appear that this implementation tries to avoid redundant staging. This is especially interesting since this version of SYCL is implemented on top of OpenCL. However a placeholder kernel may be necessary for other implementations of SYCL or other devices with different drivers, thus we implement the placeholder kernel anyway. We do not utilize it for evaluation, so we provide these details merely to detail how it would be implemented in the case that other SYCL implementations require it. We simply implement the placeholder kernel using a C++ lambda function with an empty body, explicitly listing buffers that we intend to use for communication in the lamba's capture list. This lambda function is then added to the SYCL queue for execution before measuring the MPI communication.

3.3 Enabling User Specified Memory Type for Benchmark Buffers

In previous extensions to OMB, it was required to implement the ability for users to specify whether the buffers used in the MPI operation were located within host or accelerator memory. The same requirement is needed for FPGA support, although requires a bit more robustness. One of the unique aspects of FPGA-based accelerators when compared to other accelerators like GPUs is the variety of available memories. For example, many chips have SRAM available on-chip. Soft logic resources can also be used to synthesize memories on-chip, although at much smaller capacities. Furthermore, the Xilinx Alveo U200, U250, and U280 cards have DDR available off-chip, while the Alveo U280, U50, and U55c cards have High-Bandwidth Memory available off-chip. The Vitis development flow allows user to allocate buffers on any of these memory resources, thus it is important to provide these different memories as optional pools to allocate memory from.

We implement this functionality by extending our buffer allocation wrapper to utilize the appropriate interfaces. Xilinx's OpenCL extension enables applications to specify which of these memories a buffer can be allocated from. When calling *clCreateBuffer*, a *cl_mem_ext_ptr_t* struct must be populated with the kernel to be called and the index of the kernel argument this buffer will be passed as. This indicates another challenge when implementing FPGA support for OMB. The kernel discussed in section 3.1 must be built multiple times, each time mapping the kernel arguments

to a different type of memory. Furthermore, not all FPGA accelerators have the same available memory resources as discussed above. In order to resolve this issue, the build system has been modified to ask users for the target platform of the kernel. Using this it can be determined what memory resources are available thus allowing the build system to build the kernel with the appropriate configurations.

While DPC++ does have a few extensions for FPGAs that allow the user to control how memory is implemented in kernel code, it does not provide a way for the host to allocate kernel arguments. As a result, we do not implement this feature for the SYCL-based benchmarks.

3.4 Adding Validation for FPGA-based MPI Calls

The default behavior of each benchmark is to measure the latency or bandwidth of an MPI operation. However, users of benchmark suites, such as OMB, do not always simply seek to make measurements but also to validate that the MPI operations are functionally correct. This is useful when trying to recreate a bug experienced in an application at the micro-benchmarking level or for MPI implementers to verify new versions of their libraries. OMB features the ability to validate operations by adding a *-c* argument to the command line. To implement validation for FPGAs, we add calls to *clEnqueueReadBuffer* for OpenCL runs and *sycl::memcpy* for SYCL runs after the operation has completed to stage data from the FPGA accelerator to host memory, after which that host memory can be checked to see if it contains valid contents. To prevent false positives, the staging buffer is allocated each time validation is run. This buffer is also only ever used for validation purposes, meaning at no stage during the communication is this buffer altered.

3.5 Implementing User Choice of Buffer Reuse Policy

The default behavior of OMB only allocates one buffer each for the send and receive data in each benchmark. This essentially means the benchmark reuses the same buffers for each measurement of the MPI operation. Applications often reuse the same buffers for MPI communication, as many MPI optimizations are only useful in this context. For example, in the FPGA-aware designs in MVAPICH2, there is a design that uses P2P buffer transfers to optimize intranode point-to-point communication. However, in the design, there is an import and export process that incurs a single time overhead for each buffer used in this design. If only one buffer is ever used for the operation, then the cost is only paid once. However, applications do not always reuse buffers, sometimes out of necessity. In this case, this cost would be paid multiple times. The aggregated cost of these multiple import and export procedures may be significant, thus in OMB, there is an option to specify that multiple buffers should be used during the course of the benchmark to simulate this behavior. For benchmark runs involving FPGAs, we achieve this through multiple calls to *clCreateBuffer* for OpenCL runs or *sycl::malloc_device* for SYCL runs for both send and receive buffers, using the same processes we discussed in section 3.1.

3.6 FPGA-based MPI Benchmarks Usage

Using the work discussed in previous sections, we can integrate FPGA support for several benchmarks provided by OMB. For point-to-point and multi-pair OpenCL-based benchmarks, users can indicate that the benchmark allocate the send and receive buffer from a specific memory type by passing 'H' for host memory, 'D' for DDR memory, 'B' for HBM, or 'P' for PLRAM on the command line. For example, if a user wants to measure latency from a host buffer to DDR memory on an FPGA accelerator, the command should be typed as *osu_latency H D*. For SYCL-based benchmarks, only 'H' and 'D' are provided. The default memory type is host memory, thus it is important to pass these arguments to test FPGA support. For collective benchmarks, the *-d ⟨platform⟩* flag specifies that OMB should utilize device memory allocated through a given runtime, thus when using the OpenCL interface, the command line would have *-d opencl* in it, while *-d sycl* would be used for the SYCL interface.

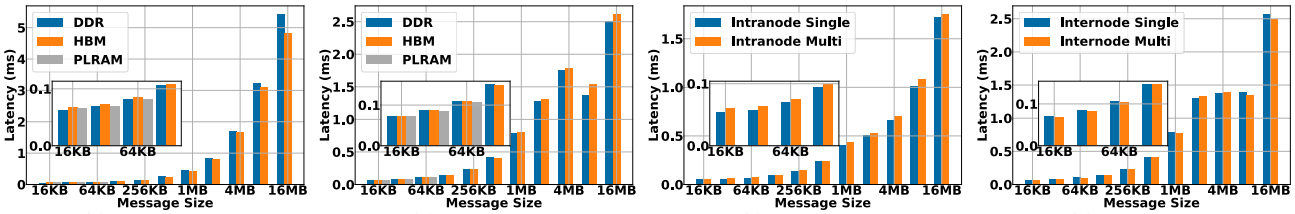
4 EXPERIMENTS AND EVALUATION

We run OMB utilizing the added support discussed in Section 3 for demonstrative purposes. All results are collected using MVAPICH2-FPGA. We have run the benchmarks with validation and all benchmarks passed. While the FPGA mapping functionality works, they have no significant effect on the results due to the limited number of configurations available on our test system. As a result all our measurements involve either inter-PCIe root tranfers or internode transfers over a high performance network.

We collect all of our numbers on a multi-node system with multiple FPGAs available per node. The FPGAs used were Xilinx U280 cards, each card existing under its own PCIe root complex. Each card contains 32GB of off-chip DDR, and 8GB of HBM. We use version 2.14 of the Xilinx Runtime. Each card uses the 202211.1 version of the XDMA shell and has PCIe Gen 3 x16 connectivity. Furthermore, we utilize nodes with Bittware 520N featuring Intel Stratix 10 GX 2800 FPGAs. The cards are connected using PCIe Gen 3 x16, and we use the 32 GB of off-chip DDR for our testing. The boards are flashed with the vendor-provided board support package. Each node also features a dual socket configuration with a AMD Milan 7713 processor, clocked at 2 GHz, in each socket. Each node is connected to a high performance network utilizing Infiniband 100/200 HDR.

4.1 Point-to-Point

We run each OMB point-to-point benchmark with three configurations. For all configurations we run 5 back-to-back executions of the benchmark and present the average results. Within each execution, multiple iterations of the MPI operation are carried out (up to 1000 for small messages, as low as 100 for large messages). Using the OpenCL interface, we execute evaluations with Xilinx FPGAs located in the same node, and another with each FPGA on separate nodes. For each of these configurations, we have runs where the send and the receive buffers are allocated from one of the available memory types: DDR, HBM, and PLRAM. Since PLRAM has a very limited capacity, we were only able to successfully run the benchmarks from a 16 KB up to a 64 KB message size. Both DDR and HBM based results are gathered from 16 KB up to 16 MB.



(a) Intranode (b) Internode (c) Intranode (d) Internode

Figure 2: Results from measuring latency of OpenCL-based point-to-point inter-FPGA communication with OMB-FPGA.

The last configuration we present is using the SYCL interface on the Bittware cards, but only using the DDR memory since the SYCL interface does not allow us to specify different memory banks from the host, as explained in Section 3. Furthermore the drivers for the Bittware cards have an issue where multiple MPI ranks on the same node cannot create SYCL queues using separate devices. Since the 520n cards are no longer supported, we cannot provide results for intranode configurations. That said, these configurations will be usable with other FPGA cards whose drivers do not have this issue, thus the designs we presented are still usable. The average variance was very low for all SYCL and OpenCL testing.

4.1.1 Latency. Figures 2a and 2b show the measured latency for both intranode and internode configurations for OpenCL-based MPI. For most data points, the memory type doesn't affect the end-to-end latency of the point-to-point communication. However, in the intranode measurements (Figure 2a) at 16MB we see that DDR memory incurs approximately 10% more latency. In Figure 2b the latency dips significantly at 8MB. This indicates that likely there is a change in the underlying protocol within MPI at this message size between 4MB and 8MB. Within MPI implementations there are many different parameters that can be tweaked to improve MPI performance without changing the higher level protocol used. Internal buffers sizes, cache sizes, queue lengths, etc. can all be set on a per system basis to increase performance. It is possible that with proper tuning of these parameters latency experienced some messages of sizes 4 MB and below can be improved.

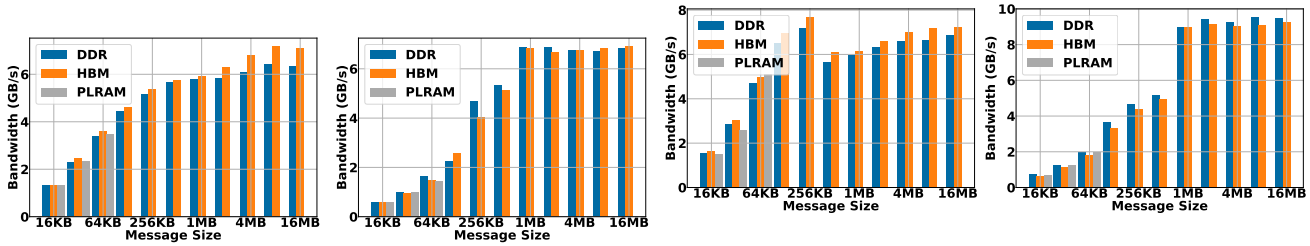
Lastly, the difference in latency between the intranode and internode configurations is significant; the internode configuration is almost half the latency of the intranode at multiple data points. Both configurations will require sending the data over a PCIe interface, but the internode configuration must also send the data over the InfiniBand network. This means that theoretically the internode latency should be either the same or greater than the intranode latency, but this is not what we see in our results. Based on our OMB-FPGA results, there is a need to retune the intranode point-to-point protocols to close this performance gap. Based on the protocols presented in [3], the implementation is trying to utilize P2P transfers for intranode communications. Given this, the P2P transfers used by the MPI implementation will be suboptimal since the FPGA cards on this system are not connected over a PCIe switch. This explains why the intranode performance is lower than internode. It is likely that the protocol selection code in MVAPICH2-FPGA needs to be changed to be aware of the interconnects between communicating FPGAs and only use P2P transfers when it would be beneficial.

We also compare the latency of point to point communication with different reuse policies in Figures 2c and 2d respectively, where single refers to maximum buffer reuse by using a single buffer and multiple means decreased buffer reuse by using multiple buffers. Across the board, the effect of using multiple buffers is insignificant. Using multiple buffers increases the latency by less than 10% in most cases. Using multiple buffers in the test generally affects caching behavior. However, since a majority of the latency is caused by the transfer between the host and the FPGA, the extra latency due to decreased cache performance is a small portion of the overall latency. We also surmise that the P2P buffer registration costs are negligible compared to other portions of the communication.

Figure 4a shows the latency of the SYCL-based MPI. The latency experienced with this MPI implementation is significantly higher than the OpenCL-based MPI implementation. This is because the MPI implementation itself uses naive designs for implementing point-to-point communication. These results should not be used to make conclusions on the performance of using OpenCL vs SYCL to implement MPI as a result. The latency can be significantly increased by implementing the optimizations presented in [3] in our SYCL-based MPI, such as implementing pipelining. However this is out of the scope of this paper. Nevertheless, our OMB-FPGA design can aid the development of these optimizations.

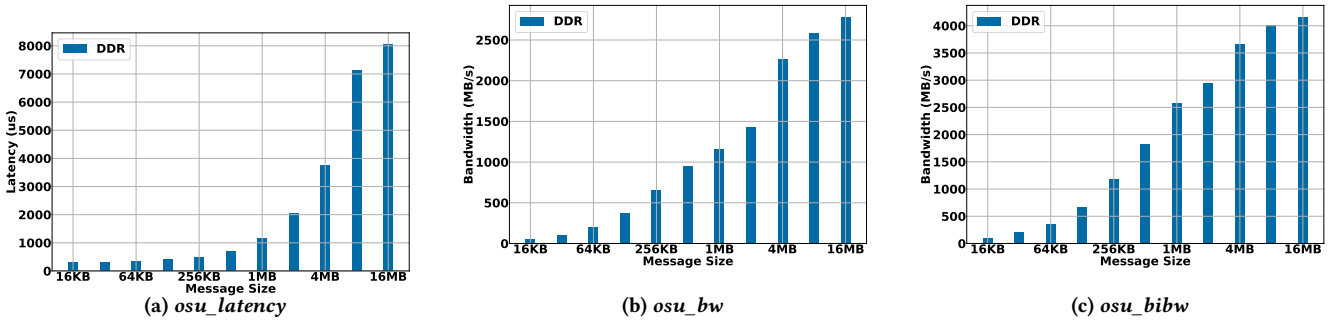
4.1.2 Bandwidth. The results collected using *osu_bw* are shown in Figures 3a and 3a for both intranode and internode configurations respectively with OpenCL-based MPI. Again we see the HBM-based tests achieve a superior bandwidth at higher message sizes (2MB and above) when compared to DDR in our intranode measurements (Figure 3a. However, when comparing the bandwidth achieved by intranode and internode tests at 1MB message size and above, we can see that the internode test reaches a peak bandwidth of 7 GB/s sooner than the intranode test. This indicates that the intranode protocols need to be re-tuned, as the intranode performance should be at least as good as the internode performance. Again, this is likely related to the P2P protocols, as discussed in 4.1.1.

Xilinx provides tests that measure the bandwidth achieved when transferring data between the FPGA and the host. The test transfers 16 MB of data and achieves an average of 9.7 GB/s when writing from host to FPGA in our test environment. This represents the peak theoretical performance of host to FPGA transfers when using OpenCL. We also execute osu_bw with using host buffers for send and receive buffers, achieving 26 GB/s and 12 GB/s for intranode and



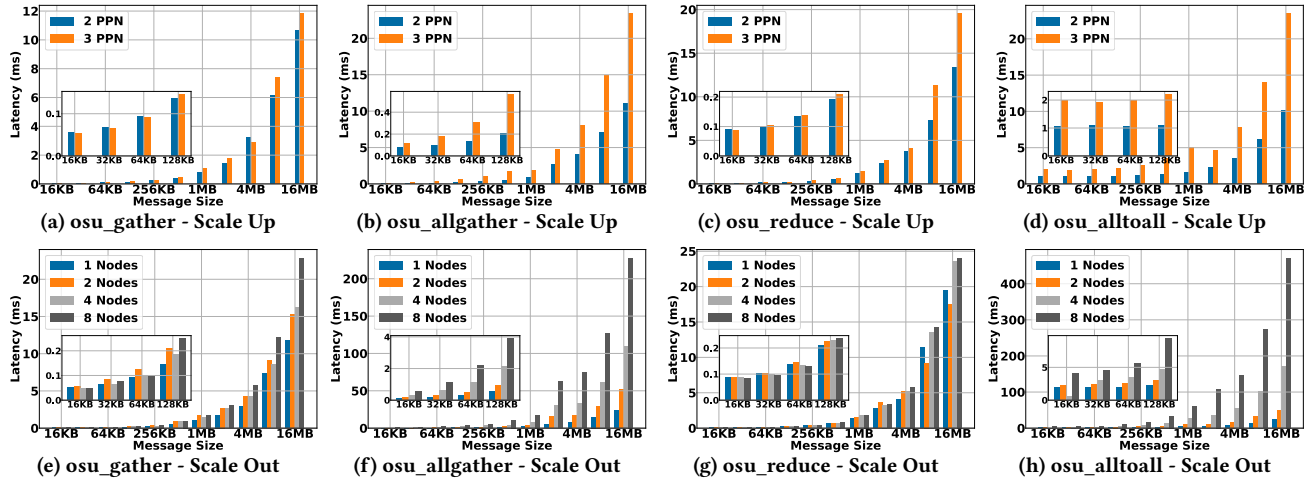
(a) Intranode (b) Internode (c) Bidirectional Intranode (d) Bidirectional Internode

Figure 3: Results from measuring bandwidth of OpenCL-based point-to-point inter-FPGA communication with OMB-FPGA.



(a) *osu_latency* (b) *osu_bw* (c) *osu_bibw*

Figure 4: Measurements for internode latency, bandwidth, and bi-directional bandwidth for SYCL-based inter-FPGA point-to-point communication



(a) osu_gather - Scale Up (b) osu_allgather - Scale Up (c) osu_reduce - Scale Up (d) osu_alltoall - Scale Up

(e) osu_gather - Scale Out (f) osu_allgather - Scale Out (g) osu_reduce - Scale Out (h) osu_alltoall - Scale Out

Figure 5: Measured latencies of various MPI collectives

internode tests respectively. This represents the peak theoretical performances of our MPI implementation when no host to device or device to host transfers are involved in the communication. Furthermore, the theoretical peak of a PCIe Gen 3 x16 interface is ~15GB/s. Since the achieved bandwidth of the xbutil test is lower than the theoretical peak PCIe bandwidth, we conjecture that the Xilinx's drivers and/or OpenCL implementation may be limiting peak bandwidth. Furthermore, since MVAPICH2-FPGA achieves a bandwidth lower than the xbutil test and host-based communication achieves higher bandwidth than MVAPICH2-FPGA, we conjecture that optimizations can be made to fully saturated the PCIe bandwidth, which is the bottleneck of inter-FPGA communications.

The SYCL-based MPI internode performance can be seen in Figure 4b. The bandwidth is far from the maximum available bandwidth. Again this is related to the naive design of our MPI implementation. For each *MPI_Isend* call, the data is first staged

to host memory before starting the asynchronous send. Since the staging call is the highest latency portion of the transfer process, this effectively serializes the communication, preventing greater bandwidth from being achieved. Bandwidth can be improved by more tightly integrating the staging code with the MPI runtime.

4.1.3 Bidirectional Bandwidth. Figure 2 demonstrates the results collected using *osu_bibw* for intranode and internode configurations with the OpenCL-based MPI. We see similar discrepancies to the bandwidth results between the peak bandwidth of the intranode and internode runs. However, the bidirectional bandwidth in the internode results has a significant jump between 512KB and 1MB that is not apparent in the unidirectional bandwidth benchmarks. This may indicate that the protocol used at message sizes 512KB and below may be limiting both directions of communication thus suggesting that tuning should be changed to select a different protocol or reworking of the used protocol may be necessary. According to [3], the pipelining protocol is used around this message range, perhaps suggesting that pipelining should be used at lower message sizes to maximize bidirectional bandwidth. The SYCL-based MPI internode performance can be seen in Figure 4b. The bidirectional bandwidth is twice of the unidirectional bandwidth, which is expected. However, this is still poor performance, and as stated above, tighter integration of the code responsible for moving data between device and host can better utilize the available interconnect bandwidth.

4.2 Collectives

We also measure the performance of FPGA-aware collective communication using OMB benchmarks. The presented numbers are an average of 5 runs, where we observed the variance to be very low. In these benchmarks we only utilize DDR to allocate the send and receive buffers for each run. We show two different types of scaling: scale up by increasing the number of communicating FPGAs within a single node and scale out by increasing the number of nodes participating in communication with 3 processes per node. The results are shown in Figure 5. In general, we see that the collectives with more dense communication patterns (i.e. alltoall) scale less effectively than others. We also see that scaling out communication is much more taxing than scaling up, as expected. A few of the measurements warrant an attempt to retune the collective operations for better performance such as the 1MB to 4MB message range in the scale out results for osu_allgather in Figure 5f and the 1MB to 2MB message range for the osu_alltoall scale up results in Figure 5d. It is also abnormal for the 2-node runs to outperform the 1-node runs in osu_reduce shown in 5c, thus we shall retune this collective to have the other scales utilize similar algorithms to ones used in the 2-node runs.

5 CONCLUSION AND FUTURE WORK

In this paper, we presented an extension to the OSU-Micro-Benchmarks suite. This extension enables the benchmarking of FPGA-aware MPI implementation by utilizing OpenCL calls to appropriately allocate buffers and execute device kernels. Furthermore, this contribution enables the ease of testing allocation of buffers from multiple memory types on the FPGA accelerator, as well as flexible process-to-device mapping to allow users to easily test the data paths between different FPGAs. Finally, we demonstrate our benchmark designs by running point-to-point and collective OMB benchmarks. Our results reveal that there is still improvement to be done in the MPI implementation we tested with. This future work will be accelerated due to the work presented in this paper, as otherwise there would be no standardized way of measuring the performance of the implementation. We plan to make our extension to OMB publicly available for use.

In the future, we would like to further develop our implementation of a SYCL-aware MPI for FPGAs. Devlopment in this area has already been done in [2], however it is not clear if this work applies to FPGAs, as well as multiple implementations of SYCL. OMB also has dedicated benchmarks for measuring NCCL, RCCL, and other non-MPI collective communication libraries' performance. ACCL [5] is a collective communication library that enables communication through FPGAs that are connected through a dedicated FPGA network. We would also like to explore adding the ability to benchmark this runtime.

ACKNOWLEDGMENTS

We would like to thank the Paderborn Computing Center for use of the Noctua2 system and AMD for admittance to the HACC program.

REFERENCES

[1] Devendar Bureddy, Hao Wang, Akshay Venkatesh, Sreeram Potluri, and Dhabaleswar K Panda. 2012. OMB-GPU: A Micro-Benchmark Suite for Evaluating MPI Libraries on GPU Clusters. In *Recent Advances in the Message Passing Interface: 19th European MPI Users' Group Meeting, EuroMPI 2012, Vienna, Austria, September 23-26, 2012. Proceedings 19*. Springer, 110–120.

[2] Chen-Chun Chen, Kawthar Shafie Khorassani, Goutham Kalikrishna Reddy Kuncham, Rahul Vaidya, Mustafa Abduljabbar, Aamir Shafi, Hari Subramoni, and Dhabaleswar K Panda. 2023. Implementing and Optimizing a GPU-aware MPI Library for Intel GPUs: Early Experiences. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 131–140.

[3] Nicholas Contini, Bharath Ramesh, Kaushik Kandadi Suresh, Tu Tran, Ben Michalowicz, Mustafa Abduljabbar, Hari Subramoni, and Dhabaleswar Panda. 2023. Enabling Reconfigurable HPC through MPI-based Inter-FPGA Communication. In *Proceedings of the 37th International Conference on Supercomputing*. 477–487.

[4] HACC. [n. d.]. Heterogeneous Accelerated Compute Cluster (HACC) Program. https://www.amd.com/en/corporate/university-program/aup-hacc.html. Accessed: 2024-02-19.

[5] Zhenhao He, Daniele Parravicini, Lucian Petrica, Kenneth O'Brien, Gustavo Alonso, and Michaela Blott. 2021. Accl: Fpga-accelerated collectives over 100 gbps tcp-ip. In *2021 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*. IEEE, 33–43.

[6] Ronan Keryell, Ruyman Reyes, and Lee Howes. 2015. Khronos SYCL for OpenCL: a tutorial. In *Proceedings of the 3rd International Workshop on OpenCL*. 1–1.

[7] Ronan Keryell and Lin-Ya Yu. 2018. Early experiments using SYCL single-source modern C++ on Xilinx FPGA: Extended abstract of technical presentation. In *Proceedings of the International Workshop on OpenCL*. 1–8.

[8] Evgeny Kuznetsov and Vladimir Stegailov. 2019. Porting CUDA-based molecular dynamics algorithms to AMD ROCm platform using hip framework: performance analysis. In *Supercomputing: 5th Russian Supercomputing Days, RuSCDays 2019, Moscow, Russia, September 23–24, 2019, Revised Selected Papers 5*. Springer, 121–130.

[9] Network-Based Computing Laboratory. [n. d.]. OSU Micro-Benchmarks. http://mvapich.cse.ohio-state.edu/benchmarks/. [Online; accessed July 12, 2024].

[10] Charles E Leiserson, Neil C Thompson, Joel S Emer, Bradley C Kuszmaul, Butler W Lampson, Daniel Sanchez, and Tao B Schardl. 2020. There's plenty of room at the Top: What will drive computer performance after Moore's law? *Science* 368, 6495 (2020), eaam9744.

[11] Yi-Chien Lin, Bingyi Zhang, and Viktor Prasanna. 2022. Accelerating GNN Training on CPU+Multi-FPGA Heterogeneous Platform. In *High Performance Computing*, Philippe Navaux, Carlos J. Barrios H., Carla Osthoff, and Ginés Guerrero (Eds.). Springer International Publishing, Cham, 16–30.

[12] Message Passing Interface Forum. 2021. *MPI: A Message-Passing Interface Standard Version 4.0*. https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

[13] Marius Meyer, Tobias Kenter, and Christian Plessl. 2020. Evaluating FPGA accelerator performance with a parameterized OpenCL adaptation of selected benchmarks of the HPCChallenge benchmark suite. In *2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*. IEEE, 10–18.

[14] OpenCL 2022. OpenCL Specification. https://registry.khronos.org/OpenCL/specs/3.0-unified/pdf/OpenCL_API.pdf.

[15] James Reinders, Ben Ashbaugh, James Brodman, Michael Kinsner, John Pennycook, and Xinmin Tian. 2021. *Data parallel C++: mastering DPC++ for programming of heterogeneous systems using C++ and SYCL*. Springer Nature.

[16] Ahmed Sanaullah and Martin C Herbordt. 2018. Fpga hpc using opencl: Case study in 3d fft. In *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*. 1–6.

[17] Kawthar Shafie Khorassani, Jahanzeb Hashmi, Ching-Hsiang Chu, Chen-Chun Chen, Hari Subramoni, and Dhabaleswar K Panda. 2021. Designing a ROCm-aware MPI library for AMD GPUs: early experiences. In *International Conference on High Performance Computing*. Springer, 118–136.

[18] Rahul Steiger. 2022. *HPCG for FPGAs: A Data-Centric Approach*. B.S. thesis. ETH Zurich.

[19] Jerome Vienne, Jitong Chen, Md. Wasi-Ur-Rahman, Nusrat S. Islam, Hari Subramoni, and Dhabaleswar K. Panda. 2012. Performance Analysis and Evaluation of InfiniBand FDR and 40GigE RoCE on HPC and Cloud Computing Systems. In *2012 IEEE 20th Annual Symposium on High-Performance Interconnects*. 48–55. https://doi.org/10.1109/HOTI.2012.19

[20] Hao Wang, Sreeram Potluri, Miao Luo, Ashish Kumar Singh, Sayantan Sur, and Dhabaleswar K Panda. 2011. MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters. *Computer Science-Research and Development* 26, 3 (2011), 257–266.