CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **00**:1–24
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

MPI and UPC Broadcast, Scatter and Gather Algorithms in Xeon Phi

Damián A. Mallón[1]*, Guillermo L. Taboada[2] and Lars Koesterke[3]

[1]*Jülich Supercomputing Centre, Institute for Advanced Simulation, Forschungszentrum Jülich, 52425 Jülich, Germany*
[2]*Computer Architecture Group, Department of Electronics and Systems, Univ. of A Coruña, 15701 A Coruña, Spain*
[3]*High Performance Computing Group, Texas Advanced Computing Center, 10100 Burnet Road (R8700) Austin, Texas*

SUMMARY

Accelerators have revolutionised the HPC community. Despite their advantages, their very specific programming models and limited communication capabilities have kept them in a supporting role of the main processors. With the introduction of Xeon Phi this is no longer true, as it can be programmed as the main processor, and has direct access to the InfiniBand network adapter. Collective operations play a key role in many HPC applications. Therefore, studying its behaviour in the context of manycore coprocessors has great importance. This work analyses the performance of different algorithms for broadcast, scatter and gather, in a large scale Xeon Phi supercomputer. The algorithms evaluated are those available in the reference MPI implementation for Xeon Phi (Intel MPI), the default algorithm in an optimised MPI implementation (MVAPICH2-MIC), and a new set of algorithms, developed by the authors of this work, designed with modern processors and new communication features in mind. The later are implemented in UPC, a PGAS language, leveraging one-sided communications, hierarchical trees and message pipelining. This study scales the experiments to 15360 cores in the Stampede supercomputer, and compares the results to Xeon and hybrid Xeon + Xeon Phi experiments, with up to 19456 cores. Copyright © 2010 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: Collective Operations; Xeon Phi; Manycore; UPC; MPI; InfiniBand

1. INTRODUCTION

Processor manufacturing hit the power wall nearly 10 years ago. As a result the microprocessor industry was forced to embrace a new era where the performance for new generation microprocessors came mainly from exposing more parallelism, rather than increased frequency. This posed a major shift in the industry, whose consequences are still difficult to deal with today.

*Correspondence to: Jülich Supercomputing Centre, Institute for Advanced Simulation, Forschungszentrum Jülich, 52425 Jülich, Germany. E-mail: d.alvarez.mallon@fz-juelich.de

Contract/grant sponsor: Stampede has been funded by the National Science Foundation; contract/grant number: OCI-1134872

The supercomputing community is now hitting a new power wall, where the power budget for supercomputers cannot grow anymore. As a result new creative ways are needed to circumvent this limitation. New approaches propose using small low-power processors like the ones used in cell phones (ARM) [1], with promising results. However, nowadays solutions rely in power-hungry manycore processors, such as Graphic Processing Units (GPUs) or Many Integrated Cores (MICs), but with a good MFLOPS/Watt ratio.

The emergence of these manycore processors has caused major changes in the HPC community. It has introduced a new level of complexity that the application developers have to deal with, adapting their applications to the new hardware, the new levels of memory hierarchy present in the new processors, and the new APIs. Moreover, until recently direct communications involving manycore processors were not possible. This issue has been solved now with modern GPUs and MPI runtimes [2], and newer techniques alleviate some of the problems still present [3]. The Xeon Phi coprocessor, released in early 2013, addresses these issues in a different way. This solution consists of new manycore processors able to run applications using the same programming model, the same tools and the same environment as the main processor. Furthermore, Xeon Phi has direct access to the network, without buffering data in the main memory of the system. However, the hardware complexity still requires significant efforts from the software developers. The software has to be adapted in order to achieve good performance. In this context, parallel runtimes play an important role in taking advantage of the underlying hardware. In particular, collective operations need to be adapted for the upcoming systems. The number of cores is increasing exponentially, and achieving a good scalability is more complicated than ever, with the efficient access to the network playing a critical role [4].

In the last few years the Partitioned Global Address Space (PGAS) programming model is becoming more popular. Despite not having the relevance of MPI, different important research groups are focusing on PGAS languages due to their potential [5]. One of their main features, one-sided asynchronous memory copies, has been already adopted in MPI. This feature can have an important role in the design of collective operations in scenarios with thousands of cores, where synchronisations becomes especially costly. This paper proposes a new set of algorithms for collective operations that take advantage of one-sided memory copies, among other features. Despite not being developed specifically for manycore systems, its design can make an effective use of the hardware available.

This work presents a performance evaluation on a Xeon Phi supercomputer of all the algorithms currently implemented in Intel MPI. The performance of these algorithms is compared with the performance of the algorithms used in MVAPICH2-MIC, a highly optimised MPI implementation for Xeon Phi. Besides MPI, this paper also presents a performance evaluation, on the same supercomputer, of a set of algorithms developed by the authors of this work. These algorithms are implemented in UPC, using (1) one-sided memory copies; (2) message pipelining; and (3) hierarchical and fixed trees. The focus of the evaluation is in representative collective operations, broadcast, gather and scatter operations, as they are common to both MPI and UPC, thus allowing a cross-comparison.

The rest of this paper is organised as follows. Section 2 discusses the related work. Section 3 explains the algorithms implemented in MPI (Intel MPI and MVAPICH2-MIC), and the design of the proposed algorithms implemented in UPC. Section 4 analyses the performance of the collective

DOI: 10.1002/cpe

operations of Intel MPI, MVAPICH2-MIC and the proposed algorithms using Berkeley UPC, on one of the largest deployments of Xeon Phi, the Stampede supercomputer. Finally, Section 5 concludes the paper.

2. RELATED WORK

The optimisation of middleware for HPC is a complex task. It might involve all the layers of the runtime, and it should evolve with new versions of languages and library APIs [6, 7]. Within the runtime optimisation, one of the most important points is the optimisation of the collective operations. Most applications rely on them, both for programmability, as they implement popular operations, relieving the programmer from its error prone implementation, and also for performance, as they generally implement optimised and refined algorithms. Collective operations are usually key to achieve a good scalability. There are basically two approaches for the optimisation of collective operations: the algorithmic and the system approach.

The algorithmic approach focuses on how the data is transferred and how the processes are organised. Not all algorithms can be suitable or implemented for all systems. Previous works on this field can be split between two main groups: distributed memory algorithms and shared memory algorithms, which target the main current architectures.

Kandalla et al. [8] proposed a design for broadcast, reduce and all-reduce operations on symmetric scenarios on Xeon Phi. Their design minimises the use of the PCIe bus and always uses a process on the main processor to communicate with remote nodes. This design is just valid for hybrid scenarios, involving both Xeon Phi and main processors, and cannot be applied to native scenarios (Xeon Phi only), where there are not main processors involved. To solve that Potluri et al. [9] proposed the use of a proxy service running on the main processor, even if the main processor is not used in the job. Intel followed a similar approach with their Coprocessor Communication Link proxy (CCL-proxy) for Intel MPI. Kandalla et al. [10] also had previously developed and tested a topology-aware algorithm that builds the interconnect tree on InfiniBand clusters taking into account the process placement in relation to the switches, avoiding unnecessary switch hops. Similarly, Gong et al. developed an algorithm for MPI collectives in the cloud [11]. Bibo Tu et al. [12] described a new broadcast algorithm for multicore clusters. In this algorithm two sets of communicators were used. The first one for intra-node communications, where binding is used to improve locality within a node. The second one is for inter-node communications. This way a broadcast is performed in two steps, inter and intra-node transfer steps, avoiding the network interface congestion. Kumar et al. [13] designed and evaluated an all-to-all algorithm for multicore clusters. This algorithm is similar to the Bibo Tu's broadcast algorithm, performed also in two steps. Chan et al. [14] proposed an algorithm that takes advantage of architectures with multiple links, where messages can be sent simultaneously over different links in systems with N-dimensional meshes/tori.

Other works have focused on optimisations for shared memory. Ramos et al. [15] modelled collective communications for cache coherent systems, and proposed enhancements for Xeon Phi taking into account the specific details of its cache implementation. Nishtala et al. [16] conducted a series of experiments in three shared memory systems, based on multicore processors, using k-nomial trees for representing the virtual topology of the processes. These experiments demonstrated

DOI: 10.1002/cpe

that for each architecture and message size the optimal radix of the k-nomial trees is different. Graham et al. [17] designed and tested a series of algorithms for shared memory, each one appropriate for a set of functions and message sizes. The described algorithms are basically fan-in or fan-out trees of variable radix; reduce-scatter (each process reduces its data) followed by a gather or all-gather; and a recursive doubling algorithm. As expected, every algorithm is the best performer for some setups, whereas not optimal for others.

Additionally, there are some works that aimed to optimise both shared and distributed memory architectures, such as the work of Mamidala et al. [18], which implements and evaluates similar algorithms to the previous works. A work more closer to ours is that of Kandalla et al. [19], which proposed a multi-leader algorithm. This proposal is similar to Bibo Tu's broadcast algorithm, except for using more than one leader per node, initially considering only the all-gather operation. Nishtala et al. [20] leveraged shared memory and trees to optimise collectives and explore their autotuning possibilities. Qian [21] followed a similar path to Kandalla et al. and proposed a series of algorithms mainly focused on all-to-all and all-gather, targeting multicore systems with multiple connections per node, as well as optimising the algorithms for cases where different processes arrive at the collective at a different time.

The algorithms considered in the related work are usually independent of their actual implementation in a particular language. However, they have been generally developed using MPI or UPC, two of the most popular choices nowadays. They are representative of message-passing and PGAS solutions, respectively. However, usually there is no algorithm that always outperforms the others. In fact, the performance of an algorithm depends on three factors: (1) message size, (2) number of processes involved, and (3) the hardware, including the network topology . Providing the best algorithm for each setup and message size is the optimum approach, as demonstrated in [22]. However, selecting among the algorithms entails a significant effort, as they are highly dependent on the system. The solution typically relies on autotuning [23], generally based on an automatic performance analysis of each algorithm for a wide range of setups.

Furthermore, it is possible to adapt the runtimes to the underlying hardware. This typically rely on adding software features in order to achieve a better usage of the underlying hardware, or adapting hardware specific layers to a given architecture, e.g., the network layer. Gupta et al. [24] were the first to leverage both hardware capable of performing Remote Direct Memory Access and message pipelining, for broadcast and all-reduce operations. Sur et al. [25] followed a similar approach for all gather operations. For Xeon Phi Potluri et al. [26] optimised MVAPICH2 by leveraging the Symmetric Communications InterFace (SCIF), a low level API that allows to control the DMA engines. Miao et al. [27] proposed a single copy method to take advantage of shared memory architectures, avoiding the system buffer. The proposal of Trahay et al. relies on a multithreaded communication engine to offload small messages [28]. Brightwell et al. [29] propose the sharing of page tables between processes, speeding up applications performance. Höfler et al. [30] proposed the use of multicast in networks, resulting in highly scalable operations, but just valid for very small messages. Velamati et al. [31] designed a set of algorithms for MPI collective operations for the heterogeneous Cell processor.

Regarding experiences with Xeon Phi and UPC, Luo et al. [32] performed the first general performance evaluation of UPC and MPI on Xeon Phi, slightly covering point to point and collective communication performance, and computational kernels.

DOI: 10.1002/cpe

The work done up to now successfully adapted collectives to different architectures. Nevertheless, it does not combine the three main features of the algorithms here presented: one-sided communications so common in PGAS languages, pipelining/overlapping and hierarchical trees. The work of Gupta et al. [24] is the closest one, implementing two collectives using one-sided communications and pipelining. However, unlike this work, their research does not tackle multi- and manycore systems, do not use hierarchical trees, and do not cover gather and scatter collectives. Moreover, their performance evaluation used a small system with outdated hardware (a cluster with 16 Pentium III nodes and GigaNet network), and limited their analysis to small and medium messages (from 4 bytes to 4KB).

In order to assess the performance of different collectives this work evaluates all the algorithms implemented in Intel MPI, the leading MPI implementation for Xeon Phi. It also evaluates the default behaviour of a specifically optimised version of MVAPICH2 for Xeon Phi (MVAPICH2-MIC). These two MPI implementations are the only options available on Stampede, being Intel MPI the preferred implementation due its stability on Xeon Phi coprocessors. As new manycore coprocessors can benefit from one-sided communications, pipelining and hierarchical trees, we have developed a set of algorithms in UPC combining the aforementioned features. The performance of these algorithms has been also evaluated and compared with the state-of-the-art MPI implementations.

3. SCALABLE ALGORITHMS FOR COLLECTIVE OPERATIONS IN MANYCORE ARCHITECTURES

Scalability is a pervasive and complex problem in HPC. Sometimes an algorithm presents easy development and good performance but it might not scale to hundreds or thousands of cores, hindering the use of today's HPC systems at their full power. The use of highly scalable methods is key, even though they might be more complex and be outperformed by other methods when using an small number of cores. This kind of methods should be the preferred choice to face up large scale problems, as demonstrated in [33], whereas less complex algorithms are acceptable for small or medium scale setups. These principles are valid both for applications and collective operations libraries.

This section describes the algorithms used in this work. The set of algorithms present in Intel MPI and MVAPICH2 is described in Subsection 3.1. Our set of algorithms developed on top of UPC and tested using Berkeley UPC are described in Subsection 3.2. It should be noted that the only documented algorithms on MVAPICH2-MIC at this moment are the ones described in the previous section. However, there is no reason to believe that the algorithms present in MVAPICH2 are not present in MVAPICH2-MIC. Intel MPI provides the user with the possibility of easily selecting the algorithms. The algorithms mentioned in its documentation will be used as a conducting thread in our enumeration, as most of them are present also in MVAPICH2.

3.1. Algorithms Implemented in Intel MPI and MVAPICH2

Intel MPI has a set of algorithms for its collective operations. Some of them are common to different operations. There is no documentation that details the implementation of the algorithms in Intel MPI.

DOI: 10.1002/cpe

However, keeping in mind that Intel MPI is based on MPICH2, it makes sense to speculate that some of its algorithms are based on MPICH2 or its derived implementations (in particular MVAPICH2).

The first algorithm mentioned in the documentation of Intel MPI is the binomial algorithm. This algorithm is present in MPICH2 and MVAPICH2, and consists basically in a binomial tree of processes, where the data is propagated top-bottom for broadcast and scatter, and bottom-up for gather.

There is a variation of the binomial algorithm in Intel MPI, called topology aware binomial. There is no description of this algorithm in the documentation. However, MVAPICH2 has an algorithm implemented as a k-nomial tree that builds the tree taking into account the topology of the nodes and network participating in the job. Therefore, seems reasonable to assume that the algorithm implemented in MVAPICH2 is also the algorithm implemented in Intel MPI, with the difference of the tree radix, that is 4 by default in MVAPICH2.

The next algorithm mentioned in the documentation of Intel MPI is the ring algorithm. This algorithm is not present in scatter or gather operations. Whereas it is possible to implement a broadcast operation using purely a ring, i.e. passing the data to the next process in a ring fashion, this is highly inefficient. However, again, MPICH2 and MVAPICH2 have an algorithm for broadcast where the data is scattered across the processes, followed by an all-gather function. This all-gather function can be implemented in a ring fashion, taking $p - 1$ steps.

There is also a topology aware version of this algorithm in Intel MPI, implemented also in MVAPICH2 following the same principles as the topology aware binomial.

The next algorithm in Intel MPI is the recursive doubling algorithm. The description for this algorithm is basically the same as for the ring algorithm, as it is also present in MPICH2 and MVAPICH2, with the difference that the all-gather phase is implemented using recursive doubling, which takes $log_2(p)$ steps.

As for the ring algorithm, the recursive doubling algorithm also has a topology aware version in Intel MPI and MVAPICH2.

The last algorithm implemented in Intel MPI is the so called Shumilin's algorithm. However, no publicly available documentation exist about the details of this algorithm, and there are no other MPI implementations with algorithms that suggest any specific detail about it.

All the MPI implementations have a default algorithm. This algorithm is one of the above mentioned. Which one depends on the specific implementation, tuning and experiment, as the default algorithm can be different for different message sizes and number of processes involved. The default algorithm on Intel MPI will be evaluated to assess the suitability of the current thresholds for a Xeon Phi environment. The default algorithm on MVAPICH2-MIC will be evaluated to assess the performance of an implementation specifically optimised and tuned for Xeon Phi.

It should be noted that, even though an explanation of the algorithms in Intel MPI is not public and admittedly the authors have speculated about their behaviour, there is not reason to believe that the algorithms do not work as described, due to: (1) self descriptive name in most of the algorithms, (2) similarity to names of algorithms described in the literature, and (3) being largely based in open source implementations that include such algorithms.

DOI: 10.1002/cpe

3.2. Proposed Scalable Collective Algorithms Developed with UPC

One of the design principles for scalability is to avoid the use of dynamic structures whose size and build time overhead increases with the number of processes. Thus, in the proposed algorithms the first call to a collective function creates a persistent and fixed (invariable) process tree structure, which can be reused in a future collective call. If the root of the collective operation and the root of the tree are not the same, then a copy of the message into the tree root is required.

The trees are constructed in two levels: The cluster level and the node level. In the cluster level a tree connecting the nodes is built. For each node one process is selected as a node leader, in charge of communicating its node with the other nodes. For efficiency and scalability purposes, a binomial tree of node leaders will be built. In the node level a new tree will be built, with the node leader as root. Here each process is bound to a specific core, without oversubscribing it (a single process per core), thus avoiding process migration and resource contention. Flat trees do not scale, as they saturate the sender or receiver (depending on the operation) easily. However, for a small number of elements in the tree, a flat tree avoids intermediate steps, reducing the synchronisation overhead. The library presented in this work evaluates both binomial and flat trees in the node level.

Another feature present in high speed network fabrics is the presence of separate links for upload and download data. Bearing that in mind, it is possible to pipeline communications, overlapping send and receive operations to reduce latency. The proposed collective algorithms implement two fragmentation schemes for pipelining: fixed and dynamic. In the fixed mode the message is fragmented into chunks of a given size. The dynamic mode is similar to the fixed mode, except that it splits the message in two halves, instead of $\lceil message\_size/chunk\_size \rceil$ messages. Thus the size of the chunks changes depending on the size of the message. The selected chunk size for the fixed mode is 32768 bytes. The dynamic mode will start fragmenting the messages when they are larger than 8192 bytes. It should be noted at this point that MVAPICH2 implements pipelining, but in a different way. In hierarchical algorithms, with differentiated steps for intra and inter node communications, MVAPICH2 can slice up the messages and perform the inter and intra node steps for every slice, rather than the whole message. However, this does not allow communication overlapping at every branch, and is a simpler and less effective mechanism.

This library also takes advantage of one-sided memory copies, implementing most functions in two approaches: *push* and *pull*. In the push approach the source process puts the data in the destination process, whereas in the pull mode it is the destination process the one which gets the data. This way it is possible to achieve a higher degree of communication overlapping, since data is streamed to/from different sources at the same time.

Synchronisation of pairs of UPC threads is necessary to develop efficient tree-based collectives. UPC provides basic and global synchronisation mechanisms like barriers and fences. Using them would add a very important overhead to any collective. Therefore, to synchronise the UPC threads, point to point synchronisation is required. Bonachea proposed an interface of semaphores [34] to allow this kind of synchronisation. Such interface has been used during the development of the algorithms to minimise overhead.

3.2.1. Scalable Scatter and Gather Operations The use of trees pose more difficulties for data distribution in scatter and gather due to the fact that every process needs or have a different and

DOI: 10.1002/cpe

unique piece of data. However, a collective using trees avoids the overhead of each process copying data separately, since less copies from/to source/destination will be done. Such approach seem interesting in scenarios where the data held by each process is not excessive. Besides this, having a scatter or gather function that uses trees has an additional benefit. Since just a few processes will communicate with the root of the operation, the memory footprint will be smaller in some systems, leading to a higher scalability. In jobs with thousands of processes this becomes a big problem as it has been pointed out before in other works [35, 36]. Mitigating this effect usually involves deep changes in the communication layer of the runtime or the transport layer. The UPC collectives library evaluated in this paper and a runtime/driver with support for on-demand connections, where buffers are allocated as needed instead of at initialisation, help to solve this problem for scatter and gather at a higher level than runtime or transport layer modifications. The tradeoff that has to be made to use trees on scatter and gather operations is to partly renounce to have hierarchical trees, as there is no way to ensure that the data required by the children processes in an hierarchical tree is contiguous. Therefore, a binomial tree ignoring topology information is built and used.

Gather has an extra particularity. It does not have a dynamic fragmentation version. The reason for this is that, since the data flows upwards, copying the first half do not make sense in most situations. The parent process could not take advantage of it, since its own first half will be larger than any of its children's first half. Therefore, data cannot flow in halves because parent processes would have to wait for the second half anyway before sending their first half.

3.2.2. Summary of the Implemented Algorithms The noticeable number of variations of the developed algorithms is the result of combining different orthogonal optimisations suitable for the operations. Table I presents an overview of the developed algorithms. The correctness of all the variations has been assessed with GUTS (GWU Unified Testing Suite) [37]. These algorithms are heavily based in our previous work [38], where they were explained and discussed in more detail.

Table I. Algorithms implemented.

			Operations		
			Broadcast	Scatter	Gather
Push	Hierarchical binomial	Standard	✓	✓	✓
		Dynamic fragmentation	✓		
		Static fragmentation	✓		✓
	Hierarchical binomial + flat	Standard	✓		
		Dynamic fragmentation	✓		
		Static fragmentation	✓		
Pull	Hierarchical binomial	Standard	✓	✓	✓
		Dynamic fragmentation	✓	✓	
		Static fragmentation	✓	✓	
	Hierarchical binomial + flat	Standard	✓		
		Dynamic fragmentation	✓		
		Static fragmentation	✓		

DOI: 10.1002/cpe

4. PERFORMANCE EVALUATION

This section presents a performance evaluation of MPI collectives and the proposed algorithms implemented in UPC. This performance evaluation has been carried out on the Stampede supercomputer [39] at TACC (Texas Advanced Computing Center), using up to 15,360 Xeon Phi cores (256 nodes). Stampede is the 7th more powerful supercomputer in the world as of November 2014. Each node has 2 Xeon E5-2680 processors with 8 cores each, clocked at 2.7 GHz, and 32GB of memory. It also has one Xeon Phi SE10P per node, with 61 cores clocked at 1.1 GHz and 8GB of memory. The operating system is CentOS 6.4, with kernel 2.6.32. The Manycore Platform Software Stack (MPSS) version is 2.1.6720-21. Generally it is recommended to rely on hybrid parallelisation with few communicating threads or processes to take full advantage of the Xeon Phi computing power in an application. In this particular case 60 cores have been used per Xeon Phi in order to maximise its computational power, taking the Xeon Phi to its limits. The interconnection network is InfiniBand 4X FDR (54.54 Gbps of theoretical effective bandwidth). The node architecture is sketched in Figure 1. In Xeon Phi the operating system running on the accelerator is bound to core 61. The processes have been distributed in a block fashion, with 60 processes per node, avoiding the core where the operating system runs. The backend compiler used is Intel icc 13.1.1. The MPI compilers and runtimes are Intel MPI 4.1.1 and MVAPICH2-MIC (based on MVAPICH2 1.9). The Intel MPI experiments used the CCL-Proxy optimisation, that routes all the Xeon Phi communication through a proxy service running in the host processor. The experiments performed using MVAPICH2-MIC used the proxy optimisation for up to 1920 processes. It was not possible to successfully run the experiments with that optimisation and more than 1920 processes. Moreover, MVAPICH2-MIC did not run reliably on hybrid mode, failing on most setups, specially using messages larger than 128 bytes. Therefore, its results are not reported. It should be noted that MVAPICH2-MIC is not yet officially supported on Stampede. MVAPICH2 results on Xeon have not been obtained, since its algorithms and runtime optimisations are similar to those available on Intel MPI. The UPC compiler and runtime is Berkeley UPC 2.16.2. As the Xeon Phi is not officially supported on this release of Berkeley UPC, a few modifications had to be done. `lfence`, `sfence` and `mfence` memory fences were removed, as the Xeon Phi memory model does not rely on those memory fences for ordering. Also, support for 128 bit atomics was disabled on GASNet, as the `cmpxchg16b` instruction is not available on Xeon Phi. With those tweaks, Berkeley UPC runs experimentally on Xeon Phi. Berkeley UPC offers the user the possibility of running UPC threads as real operating system processes, or as POSIX threads. In Stampede, using the Xeon Phi coprocessors, the use of POSIX threads is the only way to run tests up to 15360 cores, as the use of the 2 or more operating system processes crashed when running on 256 nodes. Therefore, the execution of the tests used a single process per Xeon Phi, with 60 POSIX threads (pthreads in Berkeley UPC runtime). This limitation, using a single process per node, impacts performance negatively. Thus, using 1920 cores, with 2 processes per node and 30 pthreads, shows almost twice the performance than using a single process and 60 pthreads, using the broadcast pull based algorithm with dynamic fragmentation and flat trees at the node level, with message size of 1MB. This is due to the fact that the underlying GASNet layer has to be thread safe, and therefore the overhead for locks increases with the number of pthreads used. It is important to note that all threads participate actively on the collectives, as the distinction between process and thread is

DOI: 10.1002/cpe

abstracted by the Berkeley UPC runtime, offering to the upper layers just the notion of UPC thread. A new version of Berkeley UPC (2.18.0), with official support for Xeon Phi, has been released after conducting the experiments of this research. In order to ensure the validity of the results of these experiments some of them were repeated with the new runtime, choosing the most sensible setup, i.e.: 15360 cores. No significant differences have been observed, and it is still not possible to run more than one process per node when using 256 nodes. The latest version of Berkeley UPC (2.20.0) do not contain significant changes over 2.18.0 for this research. Despite this fact, it has been also tested, without observable differences in performance.

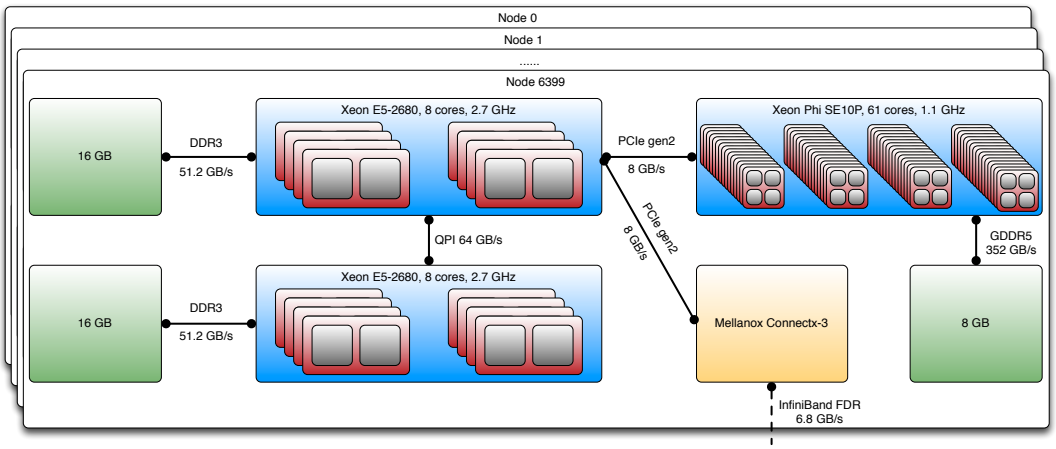

Figure 1. Stampede Node Diagram

The software used for the performance evaluation is the Intel MPI Benchmarks (IMB) suite [40] version 3.2.4, and the UPC Operations Microbenchmarking Suite (UOMS) [41] version 1.1, developed by us due to the lack of microbenchmarking tools in UPC. These two tools have small differences in how they measure performance. IMB reports minimum, maximum and average latency. However, this data is the average per message size per process. The formulas of the reported date are described in Equation 1, where p is the number of processes and n is the number of iterations for a given message size. UOMS also reports minimum, maximum and average latencies. However, these latencies are considering iterations, not processes, as UOMS considers one operation finished just when all the processes involved are done, using `UPC_IN_ALLSYNC|UPC_OUT_ALLSYNC` as synchronisation mode. The formulas for the reported data in UOMS are described in Equation 2. In order to allow comparisons as fair as possible the reported values for IMB are the maximum, i.e. the highest average time among processes, to guarantee a state where all the processes have finished the operation. The reported values for UOMS are the average, i.e. the average time per iteration needed to guarantee that all the processes have finished the operation. The reported bandwidth on this paper is calculated using these latencies. It should be noted that these two ways of measuring performance are not algebraically equivalent. The equation applied by UOMS penalises the UPC results when compared with the equation applied by IMB. However, the variability of the results, particularly in the cases with large message sizes, is small enough to render these differences negligible.

DOI: 10.1002/cpe

$$\min_{i=1}^{p}\left(\frac{\sum_{j=1}^{n} l_j}{n}\right)_i \qquad \max_{i=1}^{p}\left(\frac{\sum_{j=1}^{n} l_j}{n}\right)_i \qquad \frac{\sum_{i=1}^{p}\left(\frac{\sum_{j=1}^{n} l_j}{n}\right)_i}{p} \tag{1}$$

$$\min_{i=1}^{n}\left(\max_{j=1}^{p} l_j\right)_i \qquad \max_{i=1}^{n}\left(\max_{j=1}^{p} l_j\right)_i \qquad \frac{\sum_{i=1}^{n}\left(\max_{j=1}^{p} l_j\right)_i}{n} \tag{2}$$

Subsections 4.1 and 4.2 present the performance results of three representative collectives, broadcast, scatter and gather. Broadcast figures present the performance of a representative medium size message (16KB) on the left and the performance of a representative large size message on the right (1MB). Scatter and gather reported results have been obtained with shorter messages (8 bytes and 1KB). With large number of cores the message size tends to be shorter, and it is limited by the memory requirements in the root process, which is the result of multiplying the message size by the number of processes. The y axis represents latency in microseconds in the graphs on the left (medium size message case for broadcast and small size message case for scatter and gather), whereas the y axis represents bandwidth in GB/s or MB/s in the graphs on the right (GB/s for large size message case for broadcast and MB/s for medium size message case for scatter and gather). Variations in the same basic algorithm can lead to some dramatic performance differences. The graphs display only the most relevant algorithms (i.e.: the ones that scale and perform better) for each combination of function and message size, giving more importance to the setups with high number of cores.

Additionally, Subsection 4.3 compares the performance of the MPI and PGAS collectives on Xeon, Xeon Phi, and hybrid setups (using Xeon and Xeon Phi on the same experiment), comparing the results using fully populated nodes.

4.1. MPI Collective Performance Scalability on Xeon Phi

Figure 2 shows the performance of the broadcast operation for different MPI algorithms. In this case, the best algorithms are the binomial algorithm and the topology aware versions of the binomial algorithm and recursive doubling. The Shumilin algorithm has very good minimum run time, but the maximum is higher than the remaining algorithms. Stands out the poor performance of the default Intel MPI algorithm, shown here just for awareness of how important is to choose the right algorithm. In the medium size case the default MVAPICH2-MIC algorithm outperforms all the others, showing that the optimisations and tuning of the runtime are very effective on this scenario, even though the proxy support is not enabled for more than 1920 processes. Regarding Intel MPI the topology binomial algorithm is the one that exhibits the best performance, with a very flat increase in latency when using more than 8 nodes (480 cores). The binomial algorithm performs better than the topology aware recursive doubling, showing the suitability of tree-based algorithms for setups with small to medium sized messages and high core counts, where the recursive doubling algorithm cannot outperform the binomial algorithm, due to the small size of some of the transmitted fragments. In the large message size case, the observed results are different. MVAPICH2-MIC is not the best performer anymore, showing the importance of using the correct algorithm, regardless the specific runtime optimisations. Intel MPI outperforms MVAPICH2-MIC when using the correct

Prepared using cpeauth.cls
DOI: 10.1002/cpe

algorithm. In this scenario the fragments transmitted by the recursive doubling algorithm are not that small anymore, and its performance is closer to the binomial algorithms. It also stands out the change in the algorithm used by default by Intel MPI when using more than 4 nodes (240 cores). This leads to a significant performance boost, but far away from the best algorithms.

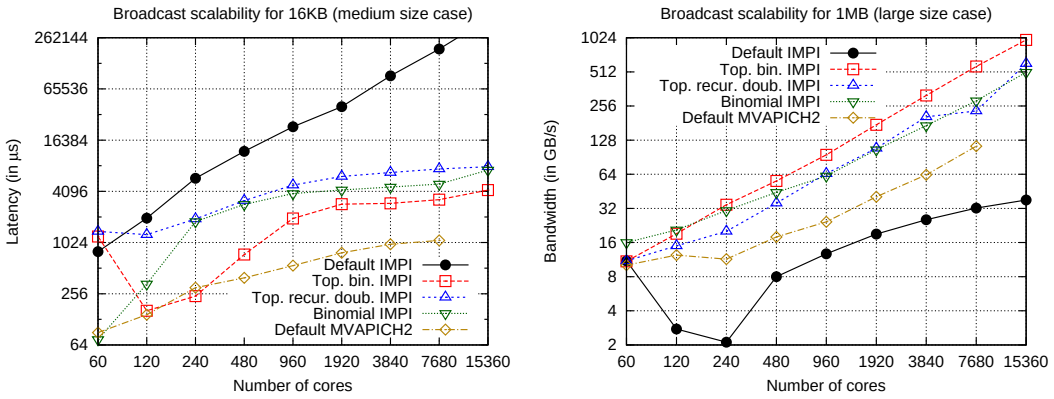

Figure 2. MPI broadcast performance and scalability on Xeon Phi

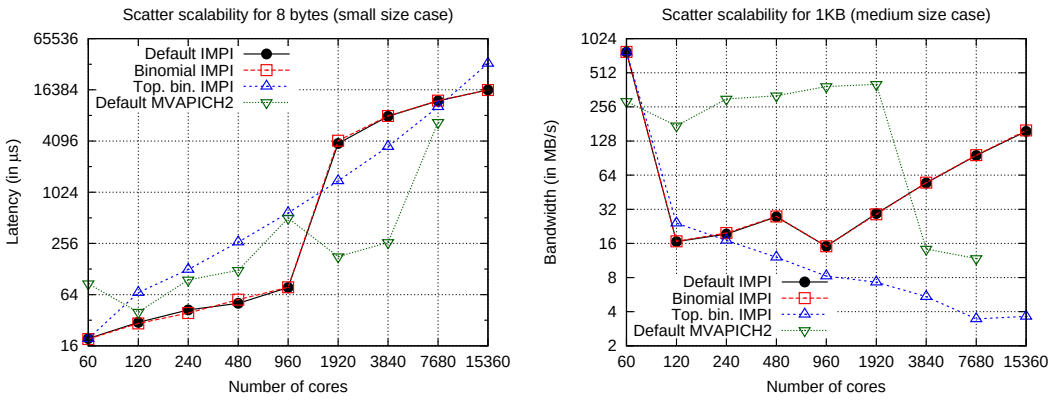

Figure 3. MPI scatter performance and scalability on Xeon Phi

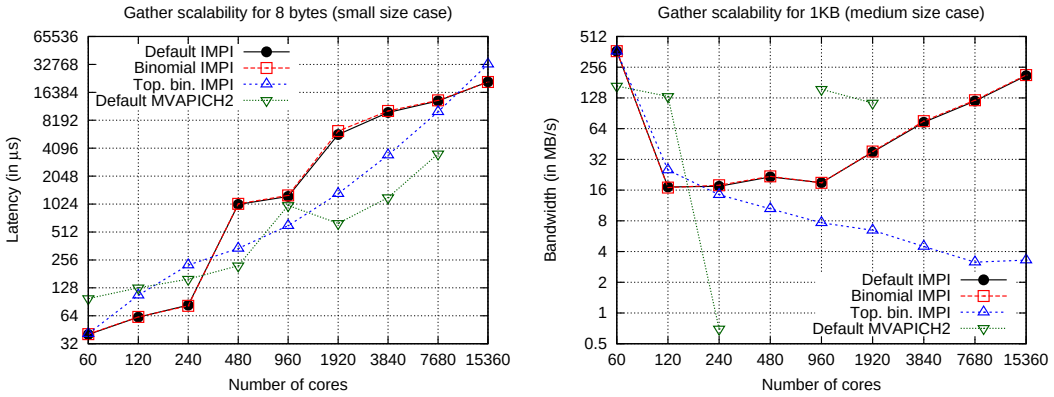

Figure 4. MPI gather performance and scalability on Xeon Phi

DOI: 10.1002/cpe

Figure 3 corresponds to the scatter operation. The Shumilin algorithm hangs for setups with more than 4 nodes, so its results are not reported. Therefore, for Intel MPI, besides the default algorithm, the relevant algorithms are the binomial and the topology aware binomial algorithms. The first result to notice is that the default MVAPICH2-MIC algorithm does not outperform the default Intel MPI algorithm for less than 1920 processes, on the small size case. The Intel MPI default algorithm behaves like the binomial for the whole range, which indicates that the default algorithm does not change depending on the number of cores or message size. In the small size case the binomial algorithm performs better than the topology aware algorithm for almost the whole range. The overhead of performing the algorithm in two phases outweighs the faster transfers, that are very fast due to the small size of the message. The binomial algorithm has a drop in performance when going from 960 cores to 1920, but keeps outperforming the topology binomial algorithm at the 15360 cores mark. For the medium size case all the algorithms drop in performance when going from 1 node to 2 (from 60 cores to 120 cores), as the network access becomes the bottleneck. Nevertheless, the default MVAPICH2-MIC algorithm keeps performing remarkably good for up to 1920 processes. With more processes the proxy optimisation has not been used, and performance degrades sharply, proving the effectiveness of this optimisation in this case. The binomial algorithm on Intel MPI keeps improving its performance when increasing the number of cores, from 120, whereas the topology binomial algorithm keeps degrading its performance. In this scenario the topology binomial could be able to slightly outperform the binomial algorithm. Nevertheless, this is not the case. The measure of the experimental results has shown a very erratic behaviour for this algorithm, with extremely differences between the minimum and the maximum for setups with more than 240 cores (being similar both values with 240 cores or less), and significant differences in performance between 64 and 128 bytes messages.

Figure 4 displays the gather operation. Gather is conceptually the inverse of scatter, and the analysis and observations done for the later are also valid here, where the algorithms are the same, and the observed behaviour and performance are very similar. The exception to this statement is the performance of MVAPICH2-MIC in the medium size case, with a very bad performance for 240 processes and no results for 480 processes. However, for 960 and 1920 processes its performance is better than any of the algorithms implemented on Intel MPI.

4.2. UPC Collective Performance Scalability on Xeon Phi

Figure 5 shows the performance of different broadcast algorithms implemented in UPC. In this case, the best algorithms are variations of the pull approach. In particular the pull algorithm with flat trees at the node level and without pipelining, the pull algorithm with dynamic fragmentation and binomial trees in the node level, and the pull algorithm with dynamic fragmentation and flat trees in the node level. In the medium size case stands out the behaviour of the three selected algorithms, that reach a plateau at around 960 cores. This shows a very good scalability, as adding more cores to the operation does not impact on the latency. This is due to the tree built to connect different nodes. Going from 1 node to 2 (60 cores to 120) has a big impact, as communications are going through the InfiniBand network, rather than exclusively through shared memory. However, the latency increases slowly, as adding an extra level to the tree is not meaningful. Stands out also the better performance of the pull approach with flat trees, which seems to be more adequate for medium messages than the other algorithms. In the large message case the performance of all the implemented algorithms

DOI: 10.1002/cpe

is similar with 15380 cores. Nevertheless, the pull algorithm with dynamic fragmentation is slightly better, as in large messages pipelining becomes more important. The algorithms perform better than MVAPICH2-MIC, similarly to the binomial and topology recursive doubling algorithms of Intel MPI, and worse than the topology binomial algorithm.

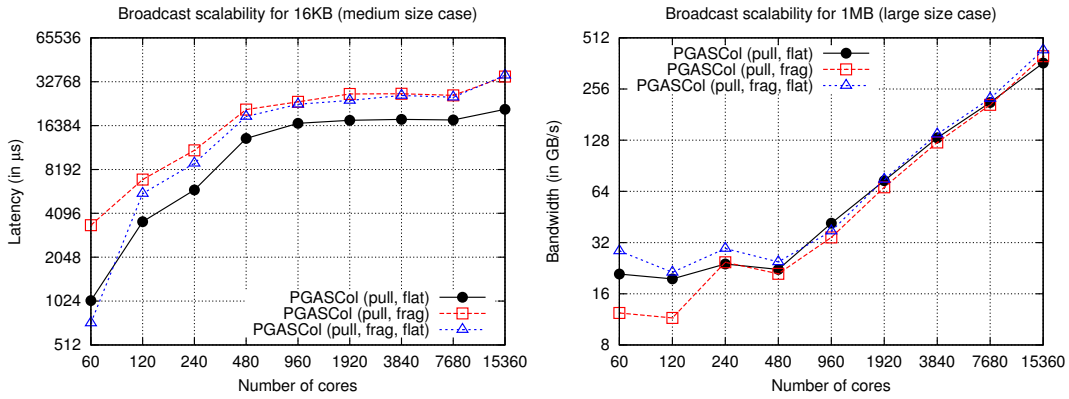

Figure 5. UPC broadcast performance and scalability on Xeon Phi

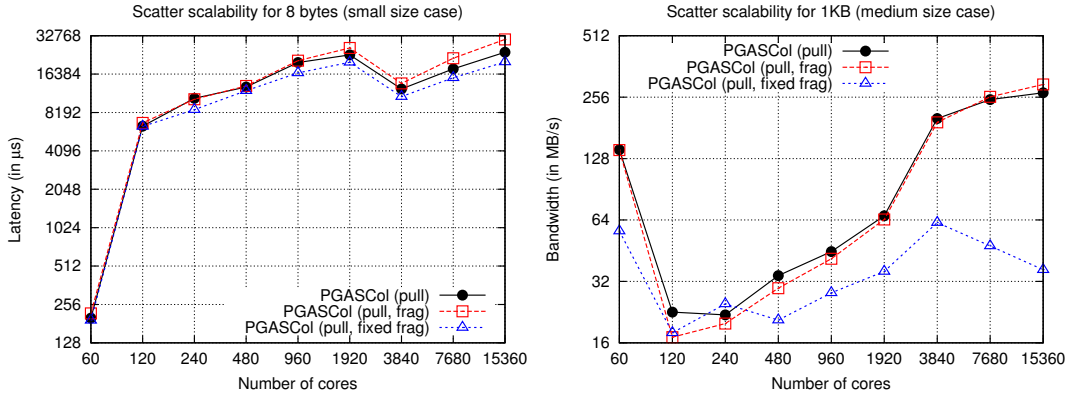

Figure 6. UPC scatter performance and scalability on Xeon Phi

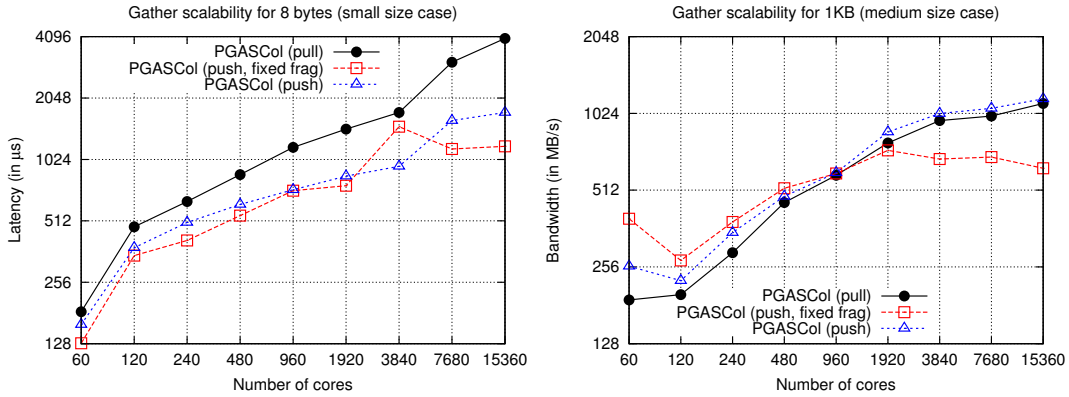

Figure 7. UPC gather performance and scalability on Xeon Phi

Prepared using cpeauth.cls
DOI: 10.1002/cpe

Figure 6 presents the performance of the scatter operation. The best performing algorithms are all pull based, with binomial trees at both levels. The 3 selected algorithms show similar scaling in the small message case. In fact, in this scenario these algorithms are essentially the same for a number of cores smaller than 3840 (for the dynamic fragmentation algorithm, 15380 for the fixed fragmentation algorithm), since there is no message fragmentation for aggregated messages smaller or equal to 8KB (32KB in the fixed fragmentation case). However, they perform differently. This suggests that the small differences in the implementation that calculates offsets, number of chunks to complete a message and their size have an impact larger than expected, an effect that probably has been augmented by the slow cores present in Xeon Phi. Here the best performer is the algorithm with fixed fragmentation, taking advantage of a more straightforward implementation with less computational load. However, in the large message case this algorithm is heavily penalised when scaling to thousands of cores because of the extra steps to synchronise the pipelining. All the algorithms have a drop in their aggregated bandwidth when going from 60 cores (1 node) to 120 cores (2 nodes), due to the impact of the use of the network, which is a major source of overhead in collective operations. In fact, the performance keeps dropping until 240 cores (4 nodes). From then on, for the pull algorithm and the pull algorithm with dynamic fragmentation the performance increases, as the messages sent through InfiniBand become larger and the usage of the network increases its efficiency. Stands out the good performance of the pull with dynamic fragmentation and pull implementations when compared to Intel MPI, as their bandwidth is up to 75% better with 15360 cores. No comparison can be done with MVAPICH2-MIC using the large 15360 cores, since MVAPICH2-MIC did not successfully run with that number of cores.

Finally, Figure 7 shows the performance of the gather operation. Typically top-down collectives like broadcast or scatter benefit from a pull approach, whereas bottom-up operations like gather benefit from a push approach. However, one of the best algorithms in these experiments follows a pull approach. This confirms the trend seen in broadcast and scatter: the performance benefit of message pipelining seems to do not compensate the extra overhead involved. The slow cores and high start-up latency of small messages discourage the use of message pipelining. In the small message case the push approach scales worse than the other two algorithms, as expected. In the medium message case the push and pull approaches scale beyond 1GB/s, whereas the algorithm with fixed fragmentation stop scaling at 1920 cores (32 nodes), where the performance degradation due to fragmentation overhead reduces the aggregated bandwidth. As with scatter, stands out the poor performance of Intel MPI when compared with these algorithms, as Intel MPI has 17$\times$ the latency and below 20% of the bandwidth in the most extreme cases. More importantly, MVAPICH2-MIC, with its specific optimisations, could not outperform these algorithms, showing their importance as opposed to focusing exclusively on runtime optimisations.

4.2.1. Influence of Runtime Configuration (pthreads vs. processes) on Xeon Phi As mentioned before, the Berkeley UPC runtime can implement UPC threads as POSIX threads or as processes. One common problem for communication runtimes is dealing with multiple threads, were internal structures have to be protected by locks. This increases the overhead, that becomes more apparent with a large number of threads. Similar algorithms to the ones here presented were studied in the past, in a traditional cluster, with performance advantages over MPI [38]. In Xeon Phi, it looks like our collectives do not achieve a good performance when compared with MPI in certain cases, like

DOI: 10.1002/cpe

on broadcast. In the experiments done in Stampede, just one process per node was used, and as many threads as cores, i.e. 60 threads per Xeon Phi. However, Figure 8 shows clearly that reducing the number of threads per process –and increasing the number of processes per node by the same factor– has a positive performance impact on performance. The performance of our algorithms when using 15 POSIX threads per process –and 4 processes per node– rivals the performance of the MPI algorithms. Avoiding the use of POSIX threads altogether in the Berkeley UPC runtime is likely to produce even better performance. Nevertheless, this is not possible on Xeon Phi when using a large number of nodes, as the Berkeley UPC runtime crashes with this configuration. Therefore, using 60 POSIX threads per process is the only way to scale to 15360 cores.

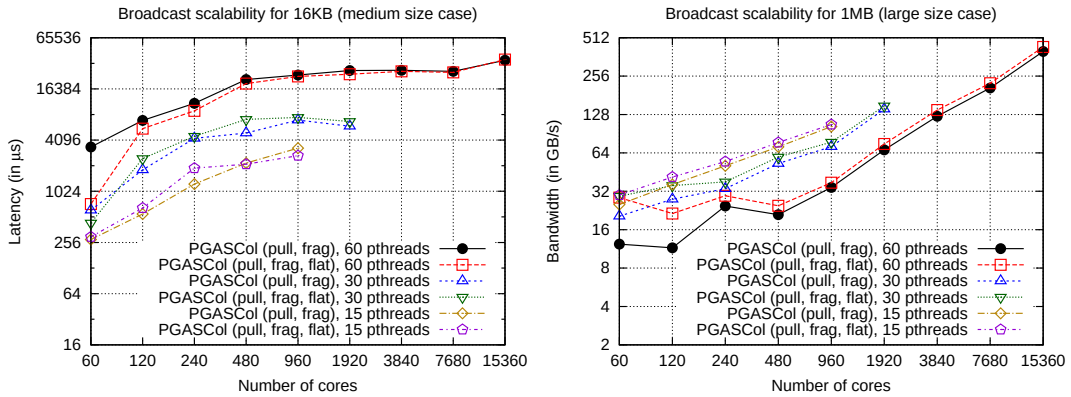

Figure 8. Effects of number of pthreads per process on UPC broadcast performance and scalability on Xeon Phi

4.3. Collective Operations Performance on Xeon versus Xeon Phi

This subsection compares the performance of collective operations in Xeon versus Xeon Phi. The purpose is to assess the impact of the processor used in communications performance. Figures 9, 10 and 11 compare the performance of the most relevant Intel MPI algorithms on Xeon Phi with the performance of the same algorithms on Xeon, using the same number of nodes, for the broadcast, scatter and gather, respectively. Results for MVAPICH2-MIC on Xeon Phi have been also included. Figures 12, 13 and 14 show the results for UPC. The comparison has been made using the same number of nodes, which are fully populated, and therefore the number of cores will be different for Xeon and Xeon Phi. Hence, the Xeon Phi experiments use 60 cores per node, whereas the Xeon experiments use 16 cores per node. All the experiments have been carried out on Stampede, using from 1 to 256 nodes. For each figure, there are two graphs. The top graphs within the figures are focused on latency, whereas the bottom graphs are focused on bandwidth. The MPI figures additionally have results with hybrid setups, using both Xeon and Xeon Phi processors. These setups also use fully populated nodes, ranging from 76 to 19456 cores (1 to 256 nodes).

Figure 9 shows the results for the MPI broadcast. Regarding latency, the results confirm the observations made on previous experiments. Start-up latency is much higher on Xeon Phi than on Xeon. The higher number of cores in Xeon Phi contributes to the higher latency, as more data has to be distributed. However, it is remarkable that the Xeon with 256 nodes is faster than Xeon Phi in a single node (except for the binomial algorithm), even though the number of cores is much

Prepared using cpeauth.cls
DOI: 10.1002/cpe

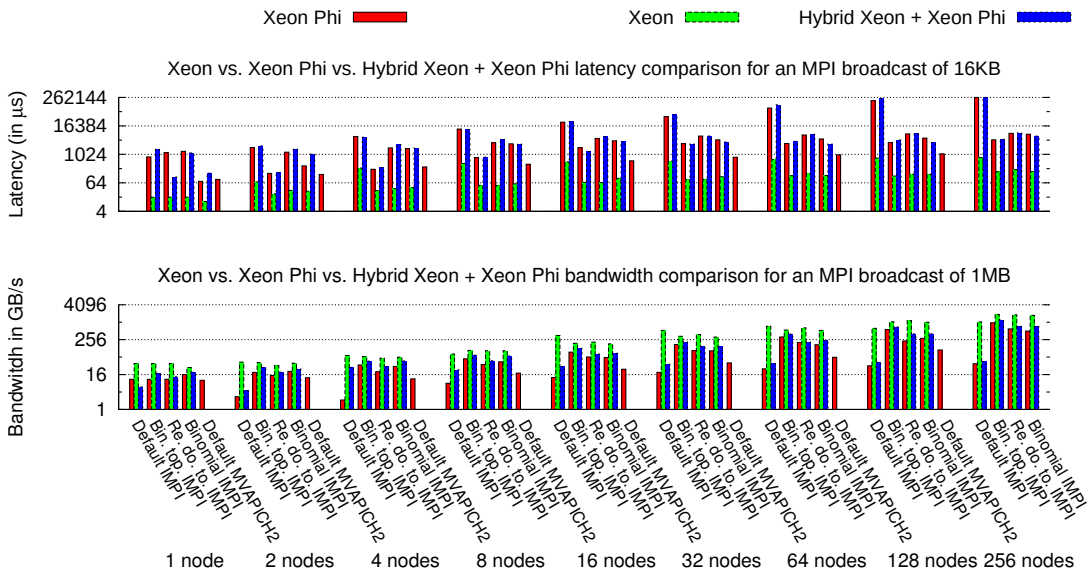

Figure 9. Performance comparison of the most relevant MPI broadcast algorithms, for Xeon, Xeon Phi and hybrid Xeon + Xeon Phi, using the same number of nodes

higher (4096 cores for Xeon vs. 60 cores for Xeon Phi). It is also highly remarkable the impact of the network, where Xeon Phi is heavily affected, especially on the default algorithm of Intel MPI, which performs poorly throughout all the tests. The hybrid Xeon + Xeon Phi setup is limited by the Xeon Phi coprocessors, and therefore its latency is almost equal to the Xeon Phi setup for all the cases. Regarding bandwidth the trend showed by the default algorithm on Intel MPI on Xeon is to scale its performance with the availability of more resources, whereas on Xeon Phi it slows down up to 8 nodes, point where the default algorithm improves, showing that the default thresholds are not set properly on Xeon Phi. The binomial algorithm with topology awareness increases its performance in both Xeon and Xeon Phi, as well as the recursive doubling algorithm with topology awareness. The performance of the binomial algorithm is very similar to the performance of the topology aware binomial. As a general conclusion, the Xeon results are significantly better than the Xeon Phi performance numbers. The performance of hybrid setups is very similar to the performance of Xeon Phi setups regarding latency. Regarding bandwidth the hybrid setups can perform slightly worse than Xeon.

Figure 10 presents the results for MPI scatter. Latency wise the trend is the same as for broadcast. The Xeon experiments have lower latency than the experiments with Xeon Phi for every setup, especially in the binomial algorithm, since the overhead of calculating the optimal tree has a big impact in setups with small messages. Differently from the broadcast case, in scatter the hybrid setups have better latency than pure Xeon Phi setups. Nevertheless, it is still significantly higher compared to the results on Xeon, that performs more than 77$\times$ better than Xeon Phi, for the default algorithm on Intel MPI using 256 nodes. On the bandwidth scenario Xeon is the best performer, with the hybrid setups between Xeon and Xeon Phi. Interestingly the topology binomial algorithm performs equal or better on Xeon than the binomial algorithm, for up to 8 nodes, whereas it is easily outperformed by the binomial algorithm in setups with more than 16 nodes. On Xeon the bandwidth

Prepared using cpeauth.cls
DOI: 10.1002/cpe

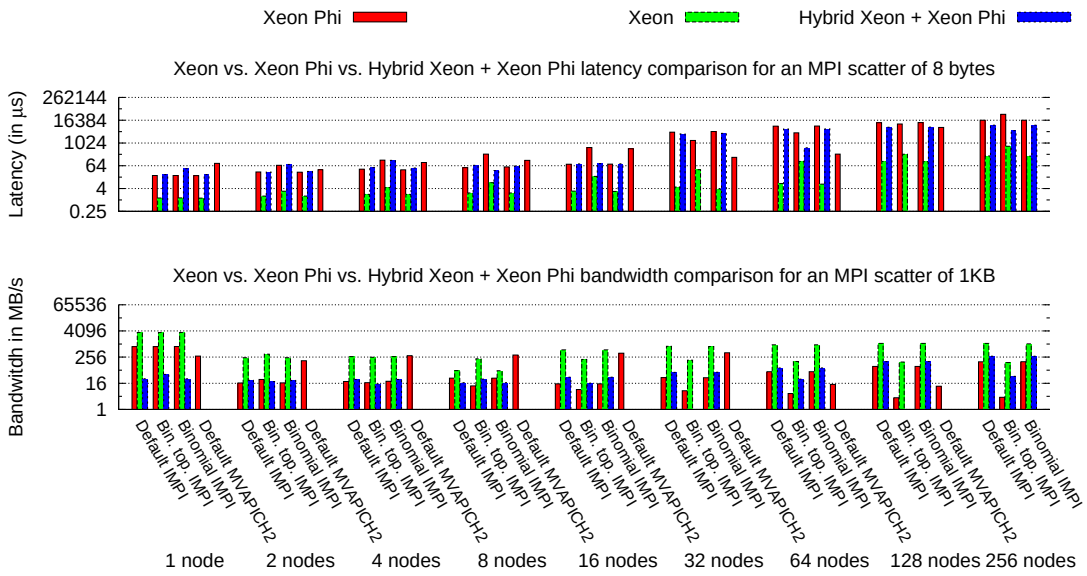

Figure 10. Performance comparison of the most relevant MPI scatter algorithms, for Xeon, Xeon Phi and hybrid Xeon + Xeon Phi, using the same number of nodes

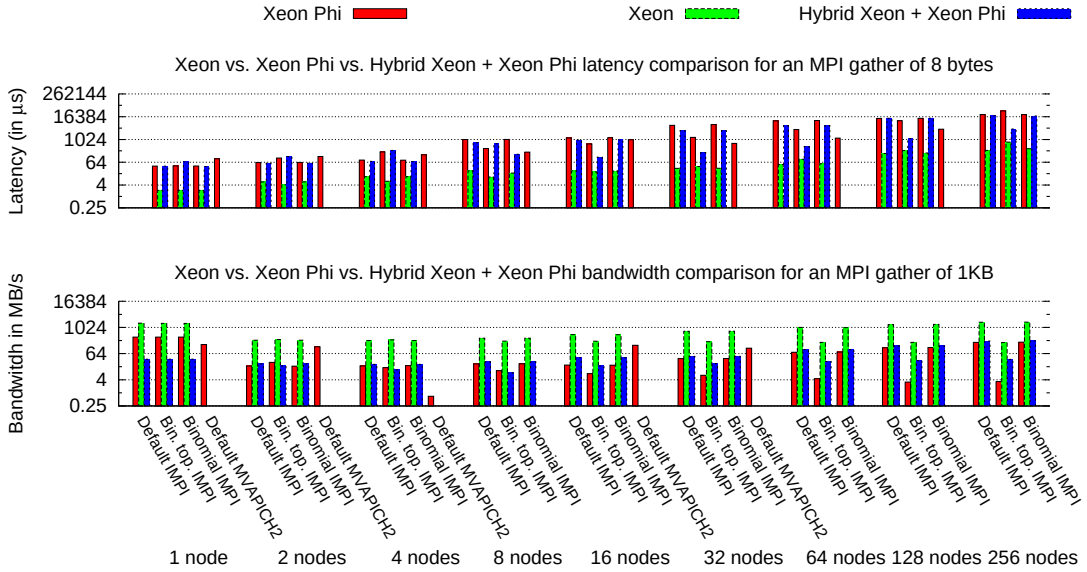

Figure 11. Performance comparison of the most relevant MPI gather algorithms, for Xeon, Xeon Phi and hybrid Xeon + Xeon Phi, using the same number of nodes

reaches its peak already with 64 nodes, whereas on Xeon Phi the bandwidth keeps increasing slowly through all the setups.

Figure 11 shows the results for MPI gather. Like the Xeon Phi analysis done in Subsection 4.1, the analysis for gather is quite similar to the analysis for scatter, with Xeon latencies much lower than on Xeon Phi. The high start-up latencies of Xeon Phi also limits the performance of the hybrid setups. Bandwidth is also much better on Xeon, with the performance of the hybrid setups slightly

DOI: 10.1002/cpe

better than the performance of the Xeon Phi setups, except for the topology binomial algorithm, where the hybrid setups perform up to 6× better than Xeon Phi.

Figure 12 shows the performance of UPC broadcast. As for MPI, for UPC the communication latency is much higher on Xeon Phi. Besides that, UPC latencies are always higher than MPI latencies. Again, Xeon Phi is more affected when using more than one node, with latencies significantly higher. These results confirm that the reason for the suboptimal performance of these algorithms on Xeon Phi is the high latency of the access to the network, as these algorithms rely extensively on short synchronisation messages to coordinate the pipelining. However, the scalability is quite good, with an almost constant latency on Xeon Phi, from 8 to 256 nodes, whereas on Xeon it increases accordingly with the number of cores. Generally, the bandwidth obtained by Xeon Phi setups is worse than Xeon setups, except than when using 256 nodes (15360 cores) and the pull algorithm with flat trees at the node level. Thus, in large messages the same trend as for medium size messages holds true, with Xeon outperforming Xeon Phi.

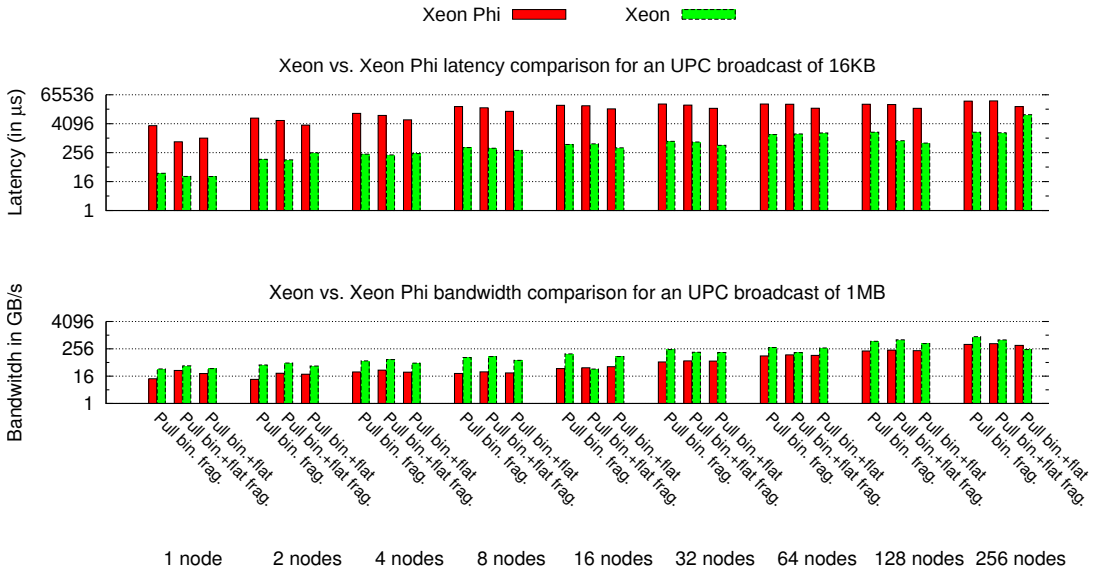

Figure 12. Performance comparison of the most relevant UPC broadcast algorithms, for Xeon and Xeon Phi, using the same number of nodes

In Figure 13 the results for UPC scatter are presented. As for the other setups already discussed, the communication latency for Xeon is much lower than for Xeon Phi. However, as the latency on Xeon Phi remains constant from 16 nodes on, the latency on Xeon keeps increasing. Regarding bandwidth, the results for Xeon and Xeon Phi for the pull binomial algorithm are similar when using a single node, but the performance of Xeon Phi falls significantly when using multiple nodes because of the network access overhead. Here Xeon is the best performer, but Xeon Phi improves when using more nodes, as the bandwidth obtained increases at a better pace than on Xeon, except for the pull algorithm with binomial trees and fixed fragmentation. In this particular case there is a high number of messages, a scenario which is especially penalised by high start-up latencies and therefore its performance does not increase.

Finally, Figure 14 shows the results for UPC gather. It is remarkable the good performance of Xeon Phi with more than 16 nodes, where it performs similarly to Xeon, or even better. Gather

Prepared using cpeauth.cls
DOI: 10.1002/cpe

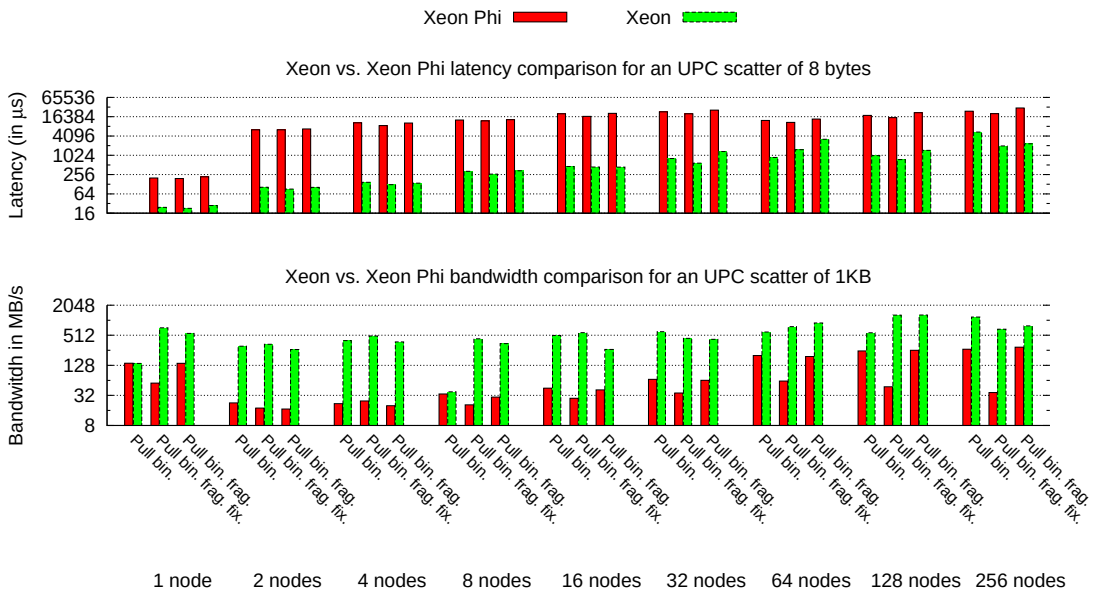

Figure 13. Performance comparison of the most relevant UPC scatter algorithms, for Xeon and Xeon Phi, using the same number of nodes

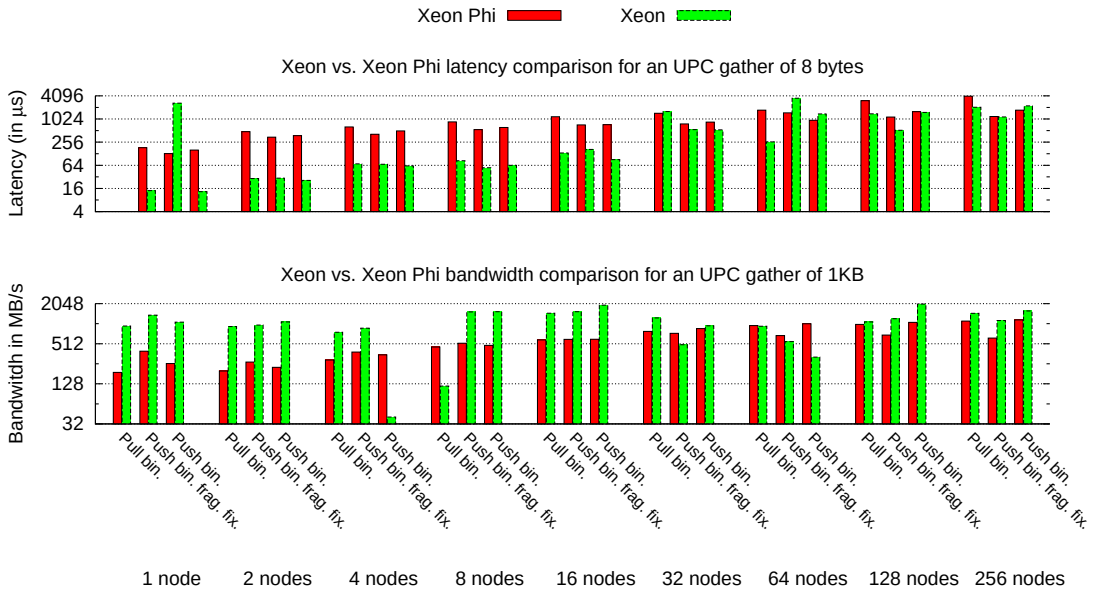

Figure 14. Performance comparison of the most relevant UPC gather algorithms, for Xeon and Xeon Phi, using the same number of nodes

is a bottom-top collective, where leaves of the tree operate independently. In other words, more parallelism is exposed at the beginning of the operation, and copies are done asynchronously towards the root, whereas for scatter less parallelism is exposed at the beginning, and transfers downwards have to wait for the data to arrive. This allows these gather algorithms to perform better than their scatter counterparts. The same effect is noticeable for bandwidth.

Prepared using cpeauth.cls
DOI: 10.1002/cpe

5. CONCLUSION

This work has analysed the performance of different algorithms for representative collective operations, particularly broadcast, scatter and gather, on a large supercomputer with Xeon Phi coprocessors. The performance of the most scalable algorithms in an MPI implementation (Intel MPI) for Xeon Phi has been assessed, as well as an MPI implementation specifically optimised and tuned for Xeon Phi (MVAPICH2-MIC). Moreover, this paper also analyses our proposal of scalable algorithms for PGAS languages, implemented in Unified Parallel C (UPC) and using the first UPC runtime available for Xeon Phi (Berkeley UPC). The potential of PGAS on future architectures has motivated this evaluation. The measured results on Xeon Phi have been compared against those using only main processors and a hybrid configuration with Xeon and Xeon Phi processors, using up to 19456 cores. To the best of our knowledge, this is the first study done at this scale, comparing orthogonally the performance of MPI and UPC collectives, and Xeon and Xeon Phi.

The analysis of the results has shown that: (1) in many cases the default algorithm for Intel MPI is not the best for Xeon Phi, as the thresholds for switching algorithms seem to have been set with main processors (e.g., Xeon) in mind; (2) the collectives operations overhead is typically much higher on Xeon Phi than on Xeon processors. In some cases it is up to two orders of magnitude higher for the same number of nodes. Applications whose performance depends on low latency communications can suffer significant performance degradation on Xeon Phi; (3) specific optimisations on the MVAPICH2-MIC runtime result in measurable benefit. Nevertheless, choosing the correct algorithm plays a major role, and MVAPICH2-MIC is frequently outperformed by less optimised runtimes in certain scenarios; (4) our algorithms implemented on UPC scale and perform well. However, the compiler and runtime are not optimised for Xeon Phi, and therefore better performance could be expected in upcoming releases. It is particularly important the use of multiple processes per node in setups with a large number of cores, minimising the use of pthreads and therefore minimising serialisation of the access to the communication layer, as well as using specific optimisations for Xeon Phi as seen in MVAPICH2-MIC. Finally, (5) even without specific optimisation the proposed algorithms implemented in UPC can outperform the MPI algorithms on Intel MPI and MVAPICH2-MIC. Some of the UPC scatter algorithms outperform the MPI scatter algorithms in bandwidth bound scenarios at large core counts, whereas for gather the performance is better using the UPC algorithms in both latency and bandwidth bound scenarios.

To sum up, both MPI and UPC collective operations are able to scale on clusters of Xeon Phi coprocessors, especially for large message sizes. Xeon Phi shows higher start-up latencies than Xeon, which limits algorithms with multiple synchronisations, but a right choice of algorithms can support its scalability. This study can serve as reference of collective operations scalability for deployments of Xeon Phi coprocessors on supercomputers.

ACKNOWLEDGEMENTS

We gratefully thank Texas Advanced Computing Centre for providing access to the Stampede supercomputer, funded by the National Science Foundation (award OCI-1134872); and Hewlett-Packard and CESGA for their support on the development of the algorithms.

DOI: 10.1002/cpe

REFERENCES

1. Rajovic N, Rico A, Puzovic N, Adeniyi-Jones C, Ramirez A. Tibidabo: Making the Case for an ARM-based HPC System. *Future Generation Computer Systems* 2013; doi:http://dx.doi.org/10.1016/j.future.2013.07.013. . In press.
2. Wang H, Potluri S, Bureddy D, Rosales C, Panda DK. GPU-Aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation. *IEEE Transactions on Parallel and Distributed Systems* 2013; **99**(PrePrints):1, doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.222.
3. Jenkins J, Dinan J, Balaji P, Peterka T, Samatova NF, Thakur R. Processing MPI Derived Datatypes on Noncontiguous GPU-Resident Data. *IEEE Transactions on Parallel and Distributed Systems* 2013; **99**(PrePrints):1, doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.234.
4. Kerbyson DJ, Barker KJ, Vishnu A, Hoisie A. A Performance Comparison of Current HPC systems: Blue Gene/Q, Cray XE6 and InfiniBand Systems. *Future Generation Computer Systems* 2014; **30**:291–304, doi:http://dx.doi.org/10.1016/j.future.2013.06.019.
5. El-Ghazawi TA, Cantonnet F, Yao Y, Annareddy S, Mohamed AS. Benchmarking Parallel Compilers: A UPC Case Study. *Future Generation Computer Systems* 2006; **22**(7), doi:http://dx.doi.org/10.1016/j.future.2006.02.002.
6. Jiang W, Liu J, wook Jin H, Panda DK, Gropp W, Thakur R. High Performance MPI-2 One-Sided Communication over InfiniBand. *Proc. 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'04)*, Chicago (IL), 2004; 531–538, doi:http://dx.doi.org/10.1109/CCGrid.2004.1336648.
7. Z Ryne and S Seidel. Ideas and Specifications for the new One-sided Collective Operations in UPC. http://www.upc.mtu.edu/papers/OnesidedColl.pdf [Last visited: January 2015].
8. Kandalla K, Venkatesh A, Hamidouche K, Potluri S, Bureddy D, Panda DK. Designing Optimized MPI Broadcast and Allreduce for Many Integrated Core (MIC) InfiniBand Clusters. *Proc. 21st Annual Symposium on High-Performance Interconnects (HOTI'13)*, San Jose (CA), 2013, doi:http://dx.doi.org/10.1109/HOTI.2013.26.
9. Potluri S, Bureddy D, Hamidouche K, Venkatesh A, Kandalla K, Subramoni H, Panda DK. MVAPICH-PRISM: A Proxy-based Communication Framework Using InfiniBand and SCIF for Intel MIC Clusters. *Proc. 25th International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*, Denver (CO), 2013, doi:http://dx.doi.org/10.1145/2503210.2503288.
10. Kandalla KC, Subramoni H, Vishnu A, Panda DK. Designing Topology-Aware Collective Communication Algorithms for Large Scale Infiniband Clusters: Case Studies with Scatter and Gather. *Proc. 10th Workshop on Communication Architecture for Clusters (CAC'10)*, Atlanta (GA), 2010; 1–8, doi:http://dx.doi.org/10.1109/IPDPSW.2010.5470853.
11. Gong Y, He B, Zhong J. Network Performance Aware MPI Collective Communication Operations in the Cloud. *IEEE Transactions on Parallel and Distributed Systems* 2013; **99**(PrePrints):1, doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.96.
12. Tu B, Fan J, Zhan J, Zhao X. Performance Analysis and Optimization of MPI Collective Operations on Multi-core Clusters. *Journal of Supercomputing* 2012; **60**(1):141–162, doi:http://dx.doi.org/10.1007/s11227-009-0296-3.
13. Kumar R, Mamidala AR, Panda DK. Scaling Alltoall Collective on Multi-core Systems. *Proc: 8th Workshop on Communication Architecture for Clusters (CAC'08)*, Miami (FL), 2008; 1–8, doi:http://dx.doi.org/10.1109/IPDPS.2008.4536141.
14. Chan E, van de Geijn R, Gropp W, Thakur R. Collective Communication on Architectures that Support Simultaneous Communication over Multiple Links. *Proc. 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'06)*, Manhattan (NY), 2006; 2–11, doi:http://dx.doi.org/10.1145/1122971.1122975.
15. Ramos S, Hoefler T. Modeling Communication in Cache-Coherent SMP Systems: a Case-Study with Xeon Phi. *Proc: 22nd International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC'13)*, New York (NY), 2013; 97–108, doi:http://dx.doi.org/10.1145/2462902.2462916.
16. Nishtala R, Yelick KA. Optimizing Collective Communication on Multicores. *Proc. 1st USENIX Workshop on Hot Topics in Parallelism (HotPar'09)*, Berkeley (CA), 2009; 1–6.
17. Graham RL, Shipman GM. MPI Support for Multi-core Architectures: Optimized Shared Memory Collectives. *Proc. 15th European PVM/MPI Users' Group Meeting (EuroPVM/MPI'08)*, Dublin (Ireland), 2008; 130–140, doi:http://dx.doi.org/10.1007/978-3-540-87475-1_21.
18. Mamidala AR, Kumar R, De D, Panda DK. MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics. *Proc. 8th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'08)*, Lyon (France), 2008; 130–137, doi:http://dx.doi.org/10.1109/CCGRID.2008.87.
19. Kandalla KC, Subramoni H, Santhanaraman G, Koop M, Panda DK. Designing Multi-Leader-Based Allgather Algorithms for Multi-Core Clusters. *Proc. 9th Workshop on Communication Architecture for Clusters (CAC'09)*,

DOI: 10.1002/cpe

Rome (Italy), 2009; 1–8, doi:http://dx.doi.org/10.1109/IPDPS.2009.5160896.
20. Nishtala R, Zheng Y, Hargrove PH, Yelick KA. Tuning Collective Communication for Partitioned Global Address Space Programming Models. *Journal of Parallel Computing* 2011; (37):576–591, doi:http://dx.doi.org/10.1016/j.parco.2011.05.006.
21. Qian Y. Design and Evaluation of Efficient Collective Communications on Modern Interconnects and Multi-core Clusters. PhD Thesis, Electrical & Computer Engineering – Queen's University 2010.
22. Thakur R, Rabenseifner R, Gropp W. Optimization of Collective Communication Operations in MPICH. *International Journal of High Performance Computing Applications* 2005; **19**(1):49–66, doi:http://dx.doi.org/10.1177/1094342005051521.
23. Vadhiyar SS, Fagg GE, Dongarra JJ. Automatically Tuned Collective Communications. *Proc. 2000 ACM/IEEE Conference on Supercomputing (SC'00)*, Dallas (TX), 2000; 1–11, doi:http://dx.doi.org/10.1109/SC.2000.10024.
24. Gupta R, Balaji P, Panda DK, Nieplocha J. Efficient Collective Operations using Remote Memory Operations on VIA-Based Clusters. *Proc. 17th IEEE International Parallel & Distributed Processing Symposium (IPDPS'03)*, Nice (France), 2003; 9, doi:http://dx.doi.org/10.1109/IPDPS.2003.1213135.
25. Sur S, Bondhugula UKR, Mamidala A, Jin HW, Panda DK. High Performance RDMA Based All-to-all Broadcast for Infiniband Clusters. *Proc. 12th International Conference on High Performance Computing (HiPC'05)*, Goa (India), 2005; 148–157, doi:http://dx.doi.org/10.1007/11602569_19.
26. Potluri S, Venkatesh A, Bureddy D, Kandalla K, Panda DK. Efficient Intra-node Communication on Intel-MIC Clusters. *Proc. 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'13)*, Delft (The Netherlands), 2013; 128–135, doi:http://dx.doi.org/10.1109/CCGrid.2013.86.
27. Miao Q, Sun G, Shan J, Chen G. Single Data Copying for MPI Communication Optimization on Shared Memory System. *Proc. 7th International Conference on Computational Science (ICCS'07)*, Beijing (China), 2007; 700–707, doi:http://dx.doi.org/10.1007/978-3-540-72584-8_93.
28. Trahay F, Brunet E, Denis A, Namyst R. A Multithreaded Communication Engine for Multicore Architectures. *Proc: 8th Workshop on Communication Architecture for Clusters (CAC'08)*, Miami (FL), 2008; 1–7, doi:http://dx.doi.org/10.1109/IPDPS.2008.4536139.
29. Brightwell R, Pedretti KT. Optimizing Multi-core MPI Collectives with SMARTMAP. *Proc. 3rd International Workshop on Advanced Distributed and Parallel Network Applications (ADPNA 2009)*, Vienna (Austria), 2009; 370–377, doi:http://doi.ieeecomputersociety.org/10.1109/ICPPW.2009.65.
30. Hoefler T, Siebert C, Rehm W. A Practically Constant-time MPI Broadcast Algorithm for Large-scale InfiniBand Clusters with Multicast. *Proc: 7th Workshop on Communication Architecture for Clusters (CAC'07)*, Long Beach (CA), 2007; 1–8, doi:http://doi.ieeecomputersociety.org/10.1109/IPDPS.2007.370475.
31. Velamati MK, Kumar A, Jayam N, Senthilkumar G, Baruah PK, Sharma R, Kapoor S, Srinivasan A. Optimization of Collective Communication in Intra-cell MPI. *Proc. 14th International Conference on High Performance Computing (HiPC'07)*, Goa (India), 2007; 488–499, doi:http://dx.doi.org/10.1007/978-3-540-77220-0_45.
32. Luo M, Li M, Venkatesh A, Lu X, Panda DK. UPC on MIC: Early Experiences with Native and Symmetric Modes. *Proc. 7th International Conference on PGAS Programming Models (PGAS'13)*, Edinburgh, Scotland (UK), 2013.
33. Mouriño JC, Gómez A, Taboada JM, Landesa L, Bértolo JM, Obelleiro F, Rodríguez JL. High Scalability Multipole Method. Solving Half Billion of Unknowns. *Computer Science - R&D* 2009; **23**(3–4):169–175, doi:http://dx.doi.org/10.1007/s00450-009-0075-7.
34. D Bonachea. Proposal for Extending the UPC Libraries with Explicit Point-to-Point Synchronisation Support. http://upc.lbl.gov/publications/upc_sem.pdf [Last visited: January 2015].
35. Koop MJ, Sridhar JK, Panda DK. Scalable MPI Design over InfiniBand using eXtended Reliable Connection. *Proc. 10th IEEE International Conference on Cluster Computing (Cluster'08)*, Tsukuba (Japan), 2008; 203–212, doi:http://dx.doi.org/10.1109/CLUSTR.2008.4663773.
36. Shipman GM, Poole S, Shamis P, Rabinovitz I. X-SRQ - Improving Scalability and Performance of Multi-core InfiniBand Clusters. *Proc. 15th European PVM/MPI Users' Group Meeting (EuroPVM/MPI'08)*, Dublin (Ireland), 2008; 33–42, doi:http://dx.doi.org/10.1007/978-3-540-87475-1_11.
37. George Washington University. GWU Unified Testing Suite (GUTS). http://threads.hpcl.gwu.edu/sites/guts [Last visited: January 2015].
38. Mallón DA, Taboada GL, Teijeiro C, González-Domínguez J, Gómez A, Wibecan B. Scalable PGAS Collective Operations in NUMA Clusters. *Journal of Cluster Computing* 2014; :1–23doi:http://dx.doi.org/10.1007/s10586-014-0377-9.
39. Stampede Supercomputer at TOP500 List. http://top500.org/system/177931 [Last visited: January 2015].
40. Intel Corporation. Intel MPI Benchmarks. http://software.intel.com/en-us/articles/intel-mpi-benchmarks [Last visited: January 2015].

DOI: 10.1002/cpe

41. Mallón DA, Mouriño JC, Gómez A, Taboada GL, Teijeiro C, Touriño J, Fraguela BB, Doallo R, Wibecan B. UPC Operations Microbenchmarking Suite. *Proc. 25th International Supercomputing Conference (ISC'10), Research Poster*, Hamburg (Germany), 2010. URL: http://upc.cesga.es [Last visited: January 2015].

DOI: 10.1002/cpe