# Designing Multi-Core Aware Inter-Communicator Operations for MPI-2 Dynamic Process Management

Krishna Kandalla, Sreeram Potluri, Miao Luo and Dhabaleswar K. Panda
Department of Computer Science and Engineering, The Ohio State University
Columbus, OH, U.S.A
{kandalla, potluri, luom, panda}@cse.ohio-state.edu

November 17, 2009

## Abstract

Dynamic process management is a feature of MPI-2 that allows an application to spawn new processes during its execution. Communication between the parent and the newly spawned process groups is established using inter-communicators. In this paper, we design a high performance communication framework for the MPI-2 dynamic process management interface over modern multi-core architectures and high performance networks. We propose efficient algorithms for point-to-point and collective operations over inter-communicators. We also propose a set of micro-benchmarks to evaluate these designs. Results show that the new designs outperform the conventional implementations by more than 30% at the micro-benchmark level. Finally, we present a synthetic application built using the NAS Grid Benchmarks to measure communication performance in applications based on multi-component architectures. Using the proposed designs, we observe a 46% improvement in the application's inter-communicator communication time.

Keywords : Message Passing Interface, Dynamic Process Management, InfiniBand, Collective communication.

## 1 Introduction

Modern multi-core architectures and high speed networks are playing a key role in powering several large scale clusters towards exascale computing. Large scale clusters comprising of several hundred compute nodes based on multi-core architectures from Intel [7] and AMD [6] with high performance networks such as InfiniBand and 10GigE [33] have become quite common over the last few years. Several researchers and application developers have evaluated the performance benefits observed with applications on such clusters. However, it is necessary to design software stacks and libraries that can fully leverage the performance benefits offered by such platforms to deliver the best performance to the applications. The Message Passing Interface (MPI) is currently the most dominant model for programming parallel applications. The MPI-1 standard was first defined in 1994 and was designed to include the attractive features of contemporary message passing systems such as PVM [10]. This specification defined a standard interface for communication, providing both point-to-point and collective communication primitives. However, the MPI-1 model was static in nature and mandated that the number of tasks in a job is fixed at launch time. This requirement restricts the applications from spawning additional tasks for compute intensive portions of the applications or to expand and contract with compute node availability. The MPI-2 specification addressed this limitation and added support for dynamic process management. This allows MPI applications to dynamically create and communicate with new processes, thus providing a new paradigm for programming MPI applications.

Dynamic process management is being used in several classes of scientific and engineering applications that are usually deployed on clusters and grid-based systems. Multi-Scale applications such as MD/FE [11], Multi-Component applications such as CFD [9] and parallel image processing applications such as [8] are some of the well-known examples that can benefit from the dynamic process management interface. These applications need to achieve load balance efficiently to ensure that the "idle" periods of the processing units are minimized and the overall system throughput is high. The load imbalance can arise due to the computational imbalance between some of the individual components of the applications [26], or due to a sudden surge in computational or networking requirements [8]. Application designers leverage the advantages of using the Dynamic Process Interface to address such issues to ensure high utilization of the computing systems. Several popular MPI implementations (MVAPICH2[29], OpenMPI[28], MPICH2[27]) support the Dynamic Process Management feature.

The dynamic process interface involves spawning new processes, setting up new connections and managing them during

the execution of a job. Some of the previous studies related to this feature [14, 4] have stressed on the need to minimize the overhead of setting up and managing the new connections and to schedule the spawned processes in a manner that leads to optimal resource utilization. Applications that use the dynamic process interface create inter-communicator objects between the original process set and the spawned process set and use them while exchanging messages between processes belong to different process groups. Basic point-to-point and collective communication operations over inter-communicators have been proposed in [31]. However, it is also necessary to model the communication costs incurred by the processes in the dynamic process interface and to design optimal point-to-point and collective message exchange algorithms over inter-communicators to minimize the communication time. In this paper, we have studied the challenges that are involved in designing a high performance communication framework in the dynamic process interface for clusters with multi-core architectures with InfiniBand as the network fabric.

To summarize, the following are the main problems that we have addressed in our paper :

- *Can we improve the inter-communicator point-to-point communication framework to achieve lower latencies?*
- *The performance of inter-communicator collective operations will strongly impact the application run-times. Is it possible to enhance the inter-communicator collective communication framework for the dynamic process interface?*
- *How do these designs reflect on the performance of the real-world applications that use dynamic process management?*

The rest of the paper is organized as follows: In Section 2, we describe a serious of concepts related to our work including MPI Communicator, InfiniBand, the Dynamic Process Interface. We also give an introduction to the applications that use the dynamic process interface, which we have studied in the following sections. Section 3 presents the various issues involved with designing a high-performance dynamic process management solution. In Section 4, we propose a number of new benchmarks to evaluate the performance of dynamic process management implementations. Section 5 provides an evaluation of the various design options proposed using benchmarks and Section 6 provides an application evaluation. Section 7 cites work related to dynamic process management and finally Section 8 provides conclusions and offers future work in this area.

# 2 Background and Motivation

In this section, we give a brief description about the concept of communicators in MPI, InfiniBand, the Dynamic Process Interface and the class of applications that we have studied in this work.

## 2.1 MPI Communicators

The MPI-1 standard introduced the concept of communicator objects to provide a convenient abstraction object for communication in parallel programs. A communicator is a software construct that defines a group of processes and a context (tag/identifier) for communication within that group. MPI operations use the rank and communicator context information to decide the target rank within the process group. MPI also allows applications to split an existing communicator to create new sub-communicators that contain a specific set of processes. These communicators are referred to as intra-communicators as they are used for communication within that group of processes. The MPI standard also defines another type of communicator, called the inter-communicator. Inter-communicators have a local process group and a remote process group and all communication is always between a process in the local group and a process in the remote group.

The dynamic process interface in MPI-2 allows a process that resides in a specific intra-communicator, to spawn a new set of processes, that are contained in a different intra-communicator. An inter-communicator that comprises of both the intra-communicators is created and facilitates communication between the two process groups.

## 2.2 InfiniBand

InfiniBand is a popular processor and I/O interconnect that has become popular and is enjoying wide success due to low latency (1.0-3.0$\mu$sec), high bandwidth and other features. Over 30% of the Top500 fastest supercomputers show InfiniBand as the interconnect being used. However, we wish to indicate that the algorithms proposed in this paper can be applied to other high speed networks such as 10GigE, as well.

## 2.3 Dynamic Process API

The MPI standard defines three functions to create and join new processes into existing MPI jobs.

- **MPI_Comm_spawn** : This is a collective function called by all the processes in a communicator. When this function is called, the root process of the communicator spawns a set of child processes. All the processes return from the call only when the inter-communicator that connects the existing process group and the newly spawned process group has been created.

- **MPI_Comm_accept/MPI_Comm_connect** : These functions provide a client-server paradigm to facilitate the creation of an inter-communicator between existing parent and child process groups.

- **MPI_Comm_join** : Using this function two processes with an existing TCP/IP connection can establish an inter-communicator and start MPI message exchange.

## 2.4 Inter-Communicator Communication operations

In this section, we give a brief overview of the point-to-point and collective message exchange operations for inter-communicators.

Applications that use the dynamic process interface will involve message exchange operations between processes that belong to the different groups of processes. These operations are done in the context of an inter-communicator that comprises of the two intra-communicator groups. Applications are free to use either point-to-point or collective operations over the inter-communicators.

As explained in [10], inter-communicator communication involves a source process in the local process group and a destination process in the remote group. There is an overhead involved in computing the destination process's rank relative to the remote process group. Owing to this, we expect inter-communicator point-to-point operations to perform slightly worse than their intra-communicator counterparts. As described in [31], the inter-communicator collective operations are implemented on top of point-to-point operations and they might also suffer from such overheads. The inter-communicator collectives also involve slightly different message exchange patterns when compared to their intra-communicator counterparts [10]. Based on the message exchange pattern, the MPI standard categorizes the inter-communicator collective operations as *All-to-All*, *All-to-One*, *One-to-All* and *Other*. *All-to-One* and *One-to-All* collective operations are rooted operations and involve the movement of messages in a specific direction between the two groups of communicators. *All-to-All* operations involve messages being exchanged between processes belonging to both the groups in both the directions. Collective calls that belong to the *Other* category either have a communication pattern that do not fit into the above categories - such as MPI_SCAN or do not involve any explicit movement of messages such as MPI_Barrier.

Point-to-point and collective communication between processes that reside within the same communicator is a very well studied area and several multi-core aware algorithms have already been proposed [24, 16] . These algorithms leverage the performance benefits offered by the modern multi-core architectures and high speed networks and significantly reduce fraction of communication time in many applications. However, applications that use the dynamic process interface involve message exchange operations across both the inter and intra communicators. Currently applications that use the dynamic process support can benefit from the proposed message exchange operations for most intra-communicator message exchange operations. However, these applications also rely on inter-communicator operations to exchange data across different modules and to synchronize. The performance of the inter-communicator collective operations plays a key role in such applications and a poorly designed set of inter-communicator message exchange operations will lead to undesirable bottlenecks and affect the performance of applications. Designing efficient multi-core aware algorithms for inter-communicator point-to-point and collective operations is an important research problem that we have addressed in this paper.

## 2.5 Applications

In this paper, we have considered two major classes of applications that use the dynamic process interface supported by the MPI-2 standard. The Master/Worker paradigm is commonly used in parallel programming. The WaterGAP application [21] and the N-Dimensional functions [23] rely on this paradigm. Applications based on the Master/Worker paradigm involve setting up inter-communicators between the master process and the worker processes. Any message exchange between the Master and the Worker processes will involve inter-communicator communication. Image processing applications such as [8] and CFD based applications use the dynamic process interface to optimize a pipelined architecture. Applications based on the pipelined architectures involve multiple components of the application connected through a pipeline and involve data exchange across these components over inter-communicator. Such applications can achieve better load balancing across by using the dynamic process interface.

As explained in [23], applications that are based on the Master/Worker paradigm, involve the following three steps within each time-step :

- The Master process broadcasts/scatters the data among the set of Worker processes

- The Workers work on their local chunks in a parallel collaborative fashion and send their contributions back to the master, when they are done

- The Master gathers/reduces the data from all the worker processes which is followed by some computation

Multi-component applications such as [26] are based on the MIMD concept and involve different groups of processes executing different executables. The different components also exchange data and synchronize with each other during the execution. Processes executing one kind of executable

can be grouped under an intra-communicator object and one of the processes is assigned as a "Master" process. The inter-component communication phases are essentially inter-communicator based communication operations. The "Master" process also acts as the root of the two communicators during the inter-communicator collective operations. In this paper, we have designed a synthetic application based on the NAS Grid Benchmarks [12] to examine and measure the inter-component communication costs in applications based on the pipelined architecture. We have considered three benchmarks from the NAS suite - BT, SP and LU and constructed a pipeline comprising of these three modules. Such a pattern is commonly found in CFD based applications [9]. The application starts off with five processes - three "Master" processes,one source process and one tail process. The three "Master" processes spawn new process groups and each process group works on a different executable. When one process group completes its execution, we move the data along the pipeline to the next process group. We have designed the inter-group communication using an all-to-one and one-to-all inter-communicator exchange operations.

As indicated earlier, applications that use dynamic process management might not able to leverage the performance benefits offered by the modern multi-core architectures if the inter-communicator message exchange algorithms are not designed in an efficient manner. In the following sections, we describe the challenges involved in designing mult-core aware inter-communicator point-to-point and collective algorithms and the potential performance benefits of using such algorithms.

# 3 Efficient Multi-core Aware Inter-Communicator Communicator Operations

In this section we describe our design for the dynamic process management framework.

## 3.1 Enhanced inter-communicator point-to-point operations

Multi-core computing has now become ubiquitous and the number of processing cores within a compute node is constantly increasing. For most message sizes, the communication between processes that reside within the same compute node is faster than communication between processes that are on different nodes. This is because the processes use the shared memory for exchanging the messages. However, this approach involves an extra-copy of the message being made and for larger messages, the overhead introduced by the extra-copy is undesirable. Also, when processes are performing the copy operation, it invariably leads to cache pollution and it severely degrades the performance. This is particularly true for architectures based on shared-L2 cache based architectures. In [16], authors have proposed utilizing a portable zero-copy based, kernel assisted design called LiMIC, for optimizing the intra-node communication. In this work, we have studied the advantages of using the LiMIC module for optimizing the intra-node communication, in the context of dynamic process management. Consider the message exchange pattern indicated in Figure 1(a). We have four processes within a compute node and two processes that reside on the same socket are sharing the L2 cache. Suppose processes P0 and P3 belong to different intra-communicators and they are involved in an inter-communicator message exchange operation, the shared-memory based approach will involve process P0 copying its data into the memory buffers and process P3 reading the data into its local buffers. As indicated in the figure, this entails moving the message through the L2 cache on both the sockets and this will lead to cache evictions if the size of the message is larger than the size of the L2 cache. We have illustrated the kernel assisted zero-copy based approach in Figure1(b)In the zero-copy based approach, it is possible to avoid the cost of copying the data on the source process and this improves the performance of intra-node inter-communicator message exchange operations. We would also like to point out that LiMIC is one of the zero-copy based approaches. Intra-node message exchange algorithms based on MPICH2's knem [2] can also be used to achieve similar performance gains.

## 3.2 Shared Memory based inter-communicator collectives

In the dynamic process framework, collective operations are performed over processes that are in both the communicator groups - the parent communicator and the spawned communicator. These collective operations are essentially inter-communicator collective operations done on the communicator that comprises of the parent and the spawned communicators. In MVAPICH2, the inter-communicator collective operations are designed on top of the intra-communicator collective calls. In Figure 2(a), we have illustrated the conventional inter-communicator One-to-All Broadcast operation. Processes P0 is the root of the intra-communicator intra-comm1 and process P0 is the root of the second intra-communicator intra-comm2. Lets assume process P0 in intra-comm1 to be the root of the inter-communicator broadcast operation. In the conventional algorithm, the root process of intra-comm1 sends the data to the root of intra-comm2 and this is followed by a binomial exchange algorithm performed on the intra-comm2 intra-communicator object. Suppose the intra-comm2 communicator has P processes, the cost of performing an inter-communicator broadcast operation is dominated by the time required to complete the binomial exchange operation over these P processes. As indicated in [34], if there are P pro-

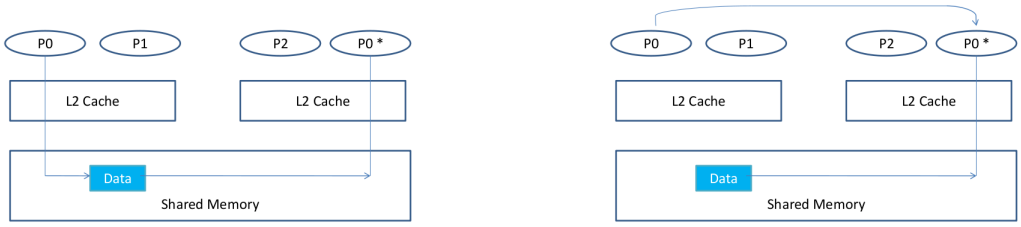


Figure 1: Comparison of the Point-to-point algorithms (a) Conventional Inter-Communicator Point-to-Point algorithms and, (b) Proposed Multi-core aware Inter-communicator Point-to-point algorithms

cesses in intra-comm2, the algorithm will involve logP steps. For small messages, the cost of an inter-communicator broadcast operation is a function of t*(logP) as the bandwidth cost is negligible, where "t" is some cost-function that considers both intra-node and inter-node message exchange operations. Also, the binomial exchange operation in multi-core architectures is not free of network contention as two or more processes in the same compute node might be involved in inter-node message exchange operations at the same time. Lets consider the cost induced by the network contention to be C. The cost of an inter-communicator broadcast operation will be a function of (t*logP + C).

In Figure 2(b), we have illustrated the proposed multi-core aware inter-communicator broadcast algorithm. In this algorithm, the binomial exchange operation is replaced by multi-core aware point-to-point or shared memory based algorithms. In such algorithms, we assign one process per node as the leader process and this process is involved in message exchange operations over the network. Once the leader process of a node has all data, it broadcasts the data to the rest of the processes that are within the same compute node. In MVAPICH2, the second phase of the broadcast operation is based on using either point-to-point binomial exchange operation or the shared-memory based algorithm that allows all the non-leader processes to read that data in parallel. For simplicity, assume that the P processes in intra-comm2 are uniformly distributed across N compute nodes, with n processes in each compute node. The inter-leader exchange operation is based on the binomial exchange algorithm and the cost of this phase will be a function of t_inter*(logN), where "t_inter" is the cost of one inter-node exchange. Also, the inter-leader exchange operation will always involve one process per node exchanging data over the network and is contention free, in fully connected networks. If we consider the shared-memory based intra-node exchange operation, this phase can proceed in parallel across all nodes, without any contention over the network. If we assume the cost of the intra-node exchange operation to be S, the cost of the proposed multi-core aware inter-communicator broadcast operation will be a function of (t_inter*(logN) + S). So, we can see that the cost of the shared-memory aware inter-communicator broadcast operation has the potential to perform better than the default binomial exchange algorithms.

# 4 Designing Benchmarks for Dynamic Process Management

In [31], authors have proposed a set basic inter-communicator collective operations and have provided a few benchmarks to measure their performance. We have augmented these by developing benchmarks for a few other inter-communicator collectives and point-to-point operations. The benchmarks are similar to the existing OSU Benchmark suite [25] released with the MVAPICH/MVAPICH2 software.

## 4.1 Inter-communicator point-to-point latency

Point-to-point inter-communicator operations involves the movement of data from a local group to a remote group. This inter-group message latency is an important metric as Master/Worker applications that rely on asynchronous progress rely on making a lot of inter-communicator point-to-point calls. Also, as indicated earlier, inter-communicator collective operations are usually implemented on top of point-to-point operations. With inter-communicators, message delivery has an additional overhead of mapping from the (local process group, rank) to the (remote process group, rank). In some designs, such as ours, no connections are setup between ranks of the local and remote process groups. Connections are setup on-demand, when the ranks really need to communicate. This connection establishment time can be excluded from measurement by using a warmup loop, as we are mainly interested in

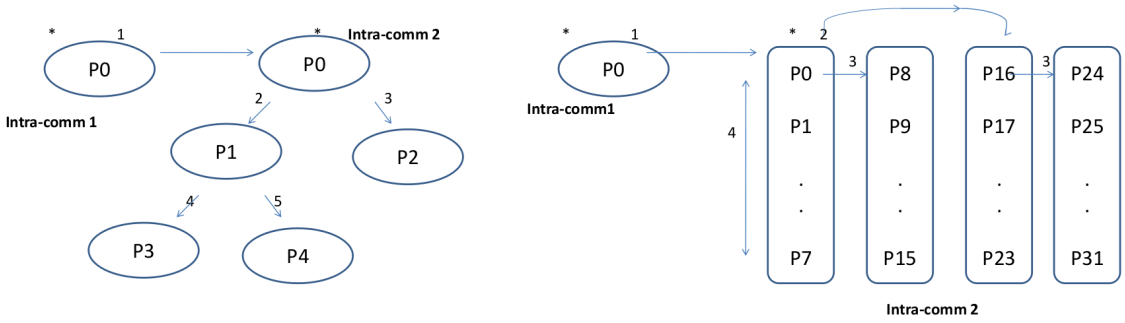


Figure 2: Comparison of the One-to-All Broadcast algorithms (a) Conventional Inter-Communicator Broadcast algorithm and, (b) Proposed Multi-core aware Inter-Communicator Broadcast operation

the communication time. Our benchmark calculates an average latency between two processes from different groups but are on the same node.

### 4.2 Inter-communicator collectives latency

As explained in Section 3, the time taken spent in the collective operations significantly impacts the overall performance of the applications. Applications that utilize the dynamic process management features perform collective operations that are done over inter-communicators that comprises of the different process groups. In Section 3, we proposed an efficient inter-communicator collective framework that utilizes the optimized shared memory based collective algorithms. It is necessary to design micro-benchmarks that measure the performance of the various inter-communicator collective algorithms. In this paper, we have designed a benchmark suite to measure the performance of a few of the inter-communicator collective operations including MPI_Bcast, MPI_Reduce and MPI_Allreduce. Since the connections between processes are set up only when the they first start sending messages to each other, it is essential to design the benchmarks to amortize the cost of the connection setup phase. In our benchmarks, for each message size, we call the inter-communicator collective operations several times and collect the average time each process takes to complete one collective operation. For *rooted* collective calls, it is also important to measure the minimum and maximum amount of time spent within the collective call, across the set of all the processes in the inter-communicator. This is because of the asymmetric nature of the message exchanges for such collective calls. In MVAPICH2, we have shared memory based collective algorithms for MPI_Bcast, MPI_Reduce, MPI_Allreduce and MPI_Barrier. In the following section we have demonstrated the benefits of using these algorithms for inter-communicator collectives in the context of communication in a dynamic environment. In these experiments, we have considered one process in the original process group and this process spawns new processes and performance inter-communicator collective operations. We chose this mode as this reflects the choice of applications in our paper. We can have any number of processes in the spawning communicator and perform inter-communicator operations with processes across both the groups.

## 5 Performance Evaluation

### 5.1 Experimental Platform

We use a 64-node Xeon cluster with each node having 8 cores and 6 GB RAM. The nodes are equipped with InfiniBand DDR HCAs and 1 GigE NICs. All the nodes are connected to a single file server accessed by the NFS (Network File System) protocol. We run our collective benchmarks using an 8x8 layout with block allocation of ranks. Our designs were implemented in the MVAPICH2 library.

### 5.2 Inter-Communicator Point-to-point Latency

The inter-group latency is a basic latency test to measure the difference between intra-communicator latency and inter-communicator latency.

In Figure 3 we compare the latency for inter-communicator and intra communicator point-to-point operations for the default version of MVAPICH2. As explained earlier, the inter-communicator latency is slightly higher than the intra communicator exchange latency.

In Figure 4, we compare the inter-communicator latency between the default version of MVAPICH2 and our proposed design that utilizes the LiMIC module. We can see that

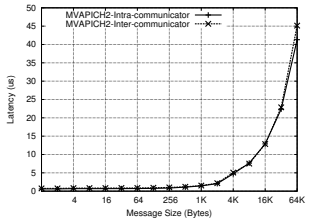


Figure 3: Inter-group vs Intra-group latency

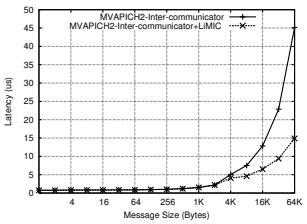


Figure 4: (b) Inter-group with and without LiMIC

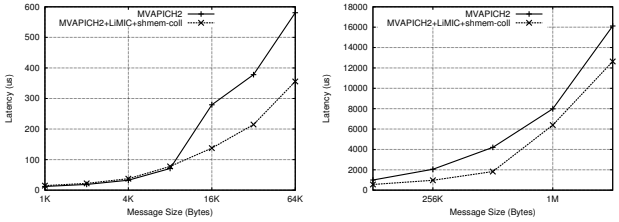


Figure 6: Broadcast Latency for 63 Spawned Processes: (a) Small Messages and (b) Large Messages

our proposed design performs significantly better the default inter-communicator version. The performance improvement is about 67% for 64KB message size.

## 5.3 Inter-communicator collectives latency

### 5.3.1 MPI_Allreduce

This benchmark measures the time taken for an inter-communicator MPI_Allreduce operation to complete. In an inter-communicator MPI_Allreduce operation between two communicators A and B, the result of the reduction of the data provided by processes in group B will be stored at all the processes in group A and vice-versa. Since MVAPICH2 offers an efficient shared memory based algorithm for MPI_Allreduce, we have explored the performance implications of having a shared-memory enabled inter-communication MPI_Allreduce operation. In this experiment the parent process spawns 63 child processes and the processes are distributed uniformly across 8 nodes. The results in Figure 5 show the performance benefits of using shared memory based inter-communicator MPI_Allreduce algorithm along with the kernel-based zero copy mechanism being used for all intra-node large message exchanges. The improvement is about 36% at 512KB message size.

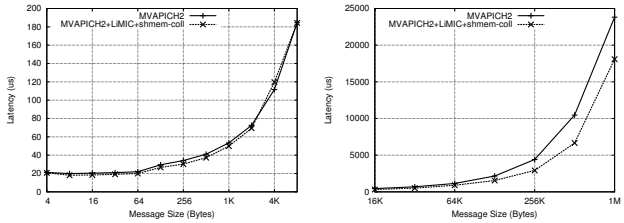


Figure 5: Allreduce Latency for 63 Spawned Processes: (a) Small Messages and (b) Large Messages

### 5.3.2 MPI_Bcast

In an inter-communicator broadcast, one of the groups defines the root and data is sent from the root to all the processes in the other group. In MVAPICH2, we perform this operation by having one point-to-point exchange between the root of the parent communicator and the root of the spawned communicator. This root process does an intra-communicator MPI_Bcast. In our proposed design, the intra-communicator MPI_Bcast phase uses the efficient shared memory broadcast algorithm. Our benchmark involves one process in the parent communicator and it spawns 63 processes and performs MPI_Bcast over this inter-communicator. The proposed schemes show notable effect on performance for message sizes beyond 16KBytes. As shown in Figure 6(b), we see a 56% improvement in latency for 512KB message size and a 22% improvement for 2MB message size.

### 5.3.3 MPI_Reduce

The inter-communicator MPI_Reduce operation involves all the processes specifying the root of the operation. Suppose the root is in the intra-communicator A, the data from all the processes in the intra-communicator B is ultimately accumulated at the root. In MVAPICH2, this operation involves a local intra-communicator MPI_Reduce performed on the remote communicator and this data is sent to the root through a point-to-point call. MVAPICH2 supports an optimized shared memory based algorithm for MPI_Reduce with intra-communicators. In our proposed design, we have used this algorithm for the reduction phase on the remote communicator. We also observed that process skew generated during back-to-back calls to MPI_Reduce resulted in widely varying timing results. To address this issue, in our benchmark, we call MPI_Barrier, before each call to MPI_Reduce. However, the time spent in MPI_Barrier is not considered for our analysis. In Figure 7, we can see that for medium and larger messages, our proposed framework delivers better performance than the default inter-communicator schemes in MVAPICH2. The new framework has a 30% improvement at 512KB message size.

# 6 Synthetic Application Description and Evaluation

As explained in Section 2.5, we have constructed a pipeline architecture by considering three benchmarks in the NAS suite to simulate the nature of inter-communicator communication in CFD based applications. The three benchmarks that we

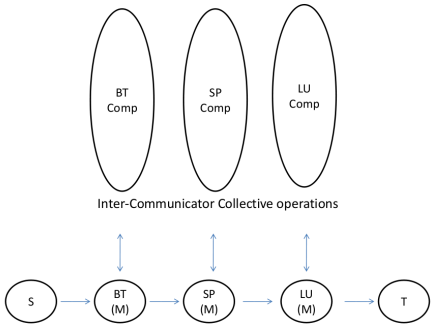


Figure 8: Synthetic DPM Application

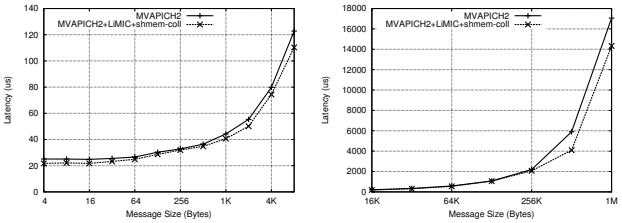


Figure 7: Reduce latency for 63 spawned processes: (a) Small Messages and (b) Large Messages

have considered are Class B - BT, SP and LU. As illustrated in Figure 8, we split the original MPI_COMM_WORLD process group comprising of the three master processes, the source and the tail processes into five different intra-communicators so that each process is contained in one intra-communicator. Each master process spawns a new process group using the MPI-2 DPM operations and sets up an inter-communicator between itself and all the spawned processes. The source creates data and sends it to the BT component's master process which does an inter-communicator one-to-all (MPI_Bcast) collective operation. Once the BT job gets done, the data from each process is accumulated at BT's master process through an inter-communicator all-to-one (MPI_Reduce) collective operation. BT's master process then moves the data along to SP's master process which performs the same set of inter-communicator collective operations. Since we are interested in measuring the time spent in the various inter-communicator collective operations, we have inserted timers around those operations. We have also measured the overall application run-time that includes the time required for each component to complete its job. This application can be extended to have a true pipeline with the source constantly sending data to the BT module and each of the modules grow and shrink with varying compute requirements. In our experiments, the pipelined application is launched with 64 processes in each of the BT, SP and LU modules. Since we are mainly interested in the inter-component communication time, we have not included the costs involved in setting up the connections. Table 1 compares the communication and total execution times of the application using the default MVAPICH2 and the library with our new designs. We see a 46% improvement in the inter-communicator communication times. As the application spends most of the time in computation, we see an improvement of about 2 seconds in the overall execution time. We would like to point out that this improvement is with one iteration through the pipeline. Since most applications belonging to this architecture are long running and involve several iterations through the pipeline, we expect to see larger benefits in the overall run-times. We have also observed that as the size of the process groups increases, the application run-times have also improved with to our proposed communication framework. We are interested in observing the benefits of our proposed designs on a larger scale. We plan to run a few experiments on large scale production level clusters such as the TACC Ranger [3] and include the results in camera-ready version of the paper.

# 7 Related Work

In this section, we describe briefly the relevant research work. The architecture of a dynamic process creation framework for MPI was described by Gropp and Lusk [15]. The MPI-2 standard [10] defined the process creation and management interface. The standard defined only the process creation interface leaving the job scheduling to the MPI implementation. Marcia Cera et al [4] have explored the issue of improving scheduling of dynamic processes. Their solutions are aimed at load balancing jobs across nodes of a cluster and providing novel ways of scheduling dynamic processes across a cluster. Gabriel et al [13] evaluated the dynamic process interfaces of several MPI-2 implementations. They measure the bandwidth achieved in point-to-point message exchange over traditional versus dynamic communicators. Silva et al, [31] proposed some of the basic inter-communicator collective operations and compared their performance against their intra-communicator counterparts. Kim et al. [18] explored the design and implementation of dynamic process management for grid-enabled MPICH.

Table 1: Application Execution Times

| | MVAPICH2 | MVAPICH2+LiMIC+shmem-coll |
|---|---|---|
| Inter-communicator Communication Time | 18,315 $\mu$s | 9,767 $\mu$s |
| Total Execution Time | 92.84 s | 90.46 s |

Several applications have been modified to leverage the advantages of using the dynamic process interface. Some of the prominent applications include [5, 23, 8, 21, 11] and so on.

In [22], authors have proposed the concept of an Application Agent to introduce the concept of dynamic process management in Grid based systems.

Several designs that utilize shared memory and RDMA features to achieve high performance for intra-communicator collective operations have been proposed. Some of the notable works include [24, 20, 17, 1, 32, 19, 30, 35]. Efficient intra-node point-to-point designs have been proposed in [16].

# 8 Conclusions

MPI-2 Dynamic process interface allows dynamic spawning of processes and is used in several classes of scientific and engineering applications. Communication between the parent and newly spawned process groups happens over inter-communications. MPI Communication operations over intra-communicators are well studied and optimized but optimizing point-to-point and collective communication operations over inter-communicators is an important issue that needs to be addressed in the context of dynamic process interface. In this paper, we have proposed a new framework for inter-communicator point-to-point and collective operations that leverage the use of kernel based zero copy intra-node message transfers and efficient shared memory based collective algorithms to achieve high performance. We implemented our designs and evaluated them on MVAPICH2, a popular MPI implementation for InfiniBand. Evaluations shows that Kernel-based zero copy mechanism (LiMIC) and shared-memory based collective algorithms can be exploited to achieve considerable performance gains in inter-communicator point-to-point and collective communication operations in the dynamic process management framework.

# Acknowledgments

This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation grants #CNS-0403342, #CCF-0702675, #CCF-0833169, #CCF-0916302 and #OCI-0926691; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Intel, Mellanox, Cisco, QLogic, and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, Advanced Clustering, Appro, QLogic, and Sun Microsystems. We would like to thank Dr.K.Tomko, whose guidance has been invaluable in bringing out this paper.

# References

[1] H.-W. Jin A. Mamidala, L. Chai and D. K. Panda. Efficient smp-aware mpi-level broadcast over infiniband's hardware multicast. In *IPDPS*. IEEE, 2006.

[2] Darius Buntinas, Brice Goglin, David Goodell, Guillaume Mercier, and Stphanie Moreaud. Cache-Efficient, Intranode, Large-Message MPI Communication with MPICH2-Nemesis. In *Proceedings of the 38th International Conference on Parallel Processing (ICPP-2009), Vienna, Austria, September 2009*, 2009.

[3] TACC: Texas Advanced Computing Center. http://www.tacc.utexas.edu/resources/hpc/.

[4] Márcia C. Cera, Guilherme P. Pezzi, Elton N. Mathias, Nicolas Maillard, and Philippe Olivier Alexandre Navaux. Improving The Dynamic Creation of Processes in MPI-2. In *PVM/MPI*, pages 247–255, 2006.

[5] A. Clematis, D. D.Agostino, and A. Galizia. A parallel image processing server for distributed applications. In *Parallel Computing: Current and Future Issues of High-End Computing, Proceedings of the International Conference ParCo 2005, NIC Series, Vol. 33,pp. 607-614, 2006*, 2006.

[6] AMD Corporation. http://www.amd.com/.

[7] Intel Corporation. http://www.intel.com/.

[8] David Cronk, Graham Fagg, Susan Emeny, and Scot Tucker. Dynamic process management for pipelined applications. In *dod-ugc, pp.250-253, 2005 Users Group Conference (DOD-UGC'05)*, 2005.

[9] David H. Bailey, NASA Ames Research Center. http://www.netlib.org/parkbench/html/npb.html.

[10] MPI Forum. MPI: A Message Passing Interface, Version 2.2. 2009.

[11] B. Fox, P. Liu, C. Lu, and H. P. Lee. Parallel multiscale computation using the message passing interface.

In *icppw, pp.199, 2003 International Conference on Parallel Processing Workshops (ICPPW'03)*, 2003.

[12] Michael Frumkin and Rob F. Van der Wijngaart. Nas grid benchmarks: A tool for grid space exploration. *Cluster Computing*, 5(3):247–255, 2002.

[13] Edgar Gabriel, Graham E. Fagg, and Jack J. Dongarra. Evaluating the Performance of MPI-2 Dynamic Communicators and One-Sided Communication. In *Lecture Notes in Computer Science*, pages 88–97, 2003.

[14] Tejus Gangadharappa, Matthew Koop, and Dhabaleswar K. Panda. Designing and evaluating mpi-2 dynamic process management support. In *Parallel Programming Models Workshop, ICPP 2009, The 38th International Conference On Parallel Processing (ICPP-2009)*, 2009.

[15] W. Gropp and E. Lusk. Dynamic process management in an MPI setting. In *SPDP '95: Proceedings of the 7th IEEE Symposium on Parallel and Distributeed Processing*, page 530, Washington, DC, USA, 1995. IEEE Computer Society.

[16] H. W. Jin, S. Sur, L. Chai, and D. K. Panda. Lightweight kernel-level primitives for high-performance MPI intranode communication over multi-core systems. In *IEEE International Conference on Cluster Computing*, 2007.

[17] K. Kandalla, H. Subramoni, G. Santhanaraman, M. Koop, and D. K. Panda. Designing Multi-Leader-Based Allgather Algorithms for Multi-Core Clusters. In *IPDPS*. IEEE, 2009.

[18] Sangbum Kim, Namyoon Woo, Young Yeom, Taesoon Park, and Hyoung-Woo Park. Design and Implementation of Dynamic Process Management for Grid-Enabled MPICH. In *PVM/MPI*, pages 653–656, 2003.

[19] Sushmitha P. Kini, Jiuxing Liu, Jiesheng Wu, Pete Wyckoff, and Dhabaleswar K. Panda. Fast and scalable barrier using rdma and multicast mechanisms for infiniband-based clusters. In *In EuroPVM/MPI*, pages 369–378, 2003.

[20] Rahul Kumar, Amith R. Mamidala, and Dhabaleswar K. Panda. Scaling alltoall collective on multi-core systems. In *IPDPS*, pages 1–8. IEEE, 2008.

[21] Claudia Leopold and Michael Sub. Observations on mpi-2 support for hybrid master/slave applications in dynamic and heterogeneous environments. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface (285-292), Lecture Notes In Computer Science, Volume 4192/2006*, 2006.

[22] Hao Liu, Nazir, A. Sorensen, and S.-A. Supporting dynamic parallel application execution in the grid. In *Fourth International Conference on Semantics, Knowledge and Grid, 2008. SKG '08,pp. 420-423 ISBN 978-0-7695-3401-5*, 2008.

[23] Elsa M. Macias and Alvaro Suarez. Solving engineering applications with lamgac over mpi-2. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface (173-183), Lecture Notes in Computer Science, Volume 2474/2002*, 2002.

[24] A. R. Mamidala, R. Kumar, D. De, and D. K. Panda. MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics. In *Int'l Symposium on Cluster Computing and the Grid (CCGrid), Lyon, France*, pages 130–137, 2008.

[25] OSU Microbenchmarks. http://mvapich.cse.ohio-state.edu/benchmarks/.

[26] Community Climate System Model. http://www.ccsm.ucar.edu/.

[27] MPICH2. http://www.mcs.anl.gov/research/projects/mpich2/.

[28] OpenMPI. http://www.open-mpi.org/.

[29] MVAPICH2: High Performance MPI over InfiniBand and iWARP. http://mvapich.cse.ohio-state.edu/.

[30] Richard L. Graham and Galen M. Shipman. MPI Support for Multi-core Architectures: Optimized Shared Memory Collectives. In *EuroPVM/MPI*, 2008.

[31] Pedro Silva and Joao Gabriel Silva. Implementing mpi-2 extended collective operations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface pp.681, Lecture Notes in Computer Science, Volume 1697/1999*, 1999.

[32] S. Sur, U. Bondhugula, A. Mamidala, H.-W. Jin, and D. K. Panda. High performance rdma based all-to-all broadcast for infiniband clusters. In *HiPC*. IEEE, 2005.

[33] Mellanox Technologies. InfiniBand and TCP in the Data-Center.

[34] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of Collective Communication Operations in MPICH. In *International Journal of High Performance Computing Applications*, pages (19)1:49–66, Spring 2005.

[35] Qian Ying and Afsahi Ahmad. Efficient shared memory and rdma based collectives on multi-rail qsnetii smp clusters. volume 11, pages 341–354, December 2008.