

Performance Analysis of User-Level PIM Communication in the Data IntensiVe Architecture (DIVA) System

Sumit Dharampal Mediratta, Jeffrey Draper

USC Information Sciences Institute, Marina del Rey, CA-90292, USA
{sumitm, draper}@isi.edu

Abstract: The performance of user-level messaging in PIM (Processing-In-Memory) to PIM communication is modeled and analyzed for the DIVA (Data IntensiVe Architecture) system. Six benchmarks have been used for this purpose, two from each category, namely single message transfer, parallel transfer and collective communication, as described for the PMB (Pallas MPI Benchmarks). The benchmarks used are PingPong, PingPing, SendReceive, Exchange, Barrier synchronization and AllToAll personalized exchange. The main significance of this work lies in the evaluation of an implementation of system-wide support for memory-to-memory and memory-to-host communication via a *parcel* buffer (used as a network interface). Another remarkable feature of this evaluation lies in presenting an optimal algorithm for Barrier synchronization and an optimal algorithm, with full channel utilization, for AllToAll personalized exchange for the bi-directional ring configuration of up to 8 DIVA PIMs in the memory system of a Hewlett-Packard's zx6000 server. The algorithms presented can be scaled for higher number of PIM chips with a little degradation in performance over the optimal solution. Our analysis shows that the currently employed communication mechanism can be used very efficiently for collective communication operations, and it also exposes the bottlenecks in the current design for future improvements.

1 Introduction

The user-level messaging performance of high-end architectures has always been a topic of interest to the hardware and software designers of clusters of SMP or PC nodes, and engineering and scientific communities related to the high-performance computing field. Much work has been done on evaluating performance of many commercial interconnects such as Myrinet, Quadrics and Infiniband [1], for middleware layers such as MPI (Message Passing Interface) and vendor-supplied communication primitives [2], and more recently of networks on chips [3]. Evaluating user-level messaging performance has also become crucial for PIM systems such as DIVA¹ [4], which aims to mitigate the processor-memory speed gap. DIVA targets two important classes of bandwidth-limited applications: multimedia and irregular

¹ This research was supported by DARPA contracts F30602-98-2-0180 and F33615-03-C-4105.

applications, including sparse-matrix and pointer computations. DIVA accelerates both classes of applications by performing computation directly in memory, requiring novel underlying hardware structures [5][6][7].

PIM systems like DIVA have unique communication requirements because of the dense packing requirements of the memory systems and the need to provide uniform communication mechanism among heterogeneous components (PIM node processors and host processor) [4]. So, PIM systems like DIVA rely upon relatively lightweight communication protocols [8] and network [9] to provide effective memory-to-memory and memory-to-host communication. To further standardize the process of performance comparison of various high-end architectures, a set of well-defined MPI benchmarks, known as PMB (Pallas MPI Benchmarks) [10], has been developed. This paper reports an evaluation for six of these benchmarks executing on a DIVA PIM system. Instead of using MPI primitives for this purpose, native communication primitives have been developed, as increasing PIM user-level messaging performance requires judicious use of underlying system architecture and interconnection network capabilities by the software. The benchmark results obtained can be considered as the optimum performance data by the potential developers of middleware layers for DIVA, ranging from explicit message-passing to shared-memory models because of the flexibility provided by the DIVA communication mechanism. The results are also of significance to potential PIM users and other PIM system architects.

Another significance of this work is in the quantification of the performance metrics in the implementation of system-wide support for memory-to-memory communication via a *parcel* buffer for the first time. A parcel is similar to an active message [11] as it is a relatively lightweight communication mechanism containing a reference to a function to be invoked when the parcel is received. Parcels are distinguished from active messages in that the destination of a parcel is an object in memory, not a specific processor. For more DIVA architectural and communication mechanism details, please refer to [4][8][9].

Another importance of this evaluation lies in presenting optimal algorithms for collective communication routines, namely Barrier synchronization and AllToAll personalized exchange for the bi-directional ring configuration of up to 8 DIVA PIMs. The allowable network size is of 2,4 or 8 PIMs. The number of PIMs is limited by the current capacity of the zx6000's memory system. An efficient implementation of the AllToAll personalized exchange benchmark is an important achievement, as it is one of the most complicated and time-consuming communication operations in high performance computing.

In this benchmark, a personalized (or different) message is sent to every node in the network by every other node. Usually one of two types of algorithms is used to perform this operation – *Direct* or *Indirect*. Direct algorithms involve the transfer of all messages in several contention free phases directly from source node to the destination node, without involving any intermediate nodes [12]. Indirect algorithms involve the intermediate nodes in the message transfer, and a message combination step is added before the combined message is forwarded to the destination node. The indirect algorithms perform better for the cases where message length is small or communication startup time is large compared to the link transmission time [12]. This is because of the reduction in the number of phases required for indirect algorithms. Many algorithms are available for wormhole routed torus networks [12][13][14],

which can be tailored for implementation in the DIVA PIM system environment. However, the deficiencies of these algorithms due to their indirect nature [12][13] or the underlying assumptions (like use of only one virtual channel by [14]) motivated the development of the presented algorithm, which is of *Direct* type because of the reasons discussed in section 2.6.

Section 2 describes the different benchmarks analyzed, algorithms and procedure used, and gives expressions for the performance metrics; section 3 presents the plots of the performance metrics with variation in message length and system size, as applicable, and explains the results; finally, section 4 concludes this paper with some suggestions for potential improvements in the design.

2 Benchmarks – Implementation Algorithms and Analysis

The performance of six benchmarks, namely PingPong, PingPing, SendReceive, Exchange, Barrier synchronization and AllToAll personalized exchange, is analyzed for the above described PIM communication mechanism. Two benchmarks from each category, namely single message transfer, parallel transfer and collective communication (as specified in [10]) are chosen. For PingPong, PingPing, SendReceive, and Exchange, both the timing expressions and throughput calculations are given. For barrier synchronization and AlltoAll personalized exchange only timing expressions are given. These calculations are in accordance with the rules specified by [10].

2.1 PingPong

This benchmark measures the efficiency of a *single message transfer* between two processes (in this case two PIM nodes). A single message is sent from one node to another node and received back. The simplicity of bi-directional connection of two PIM nodes, for this purpose, is apparent in [9].

Send and *Receive* communication primitives have been implemented at the assembly level. The processor clock cycles ($\text{Cycle}_{\text{proc}}$) taken by each native communication primitive have been used in the performance analysis. [5][6][7] explain DIVA PIM processing logic in detail. The clock cycles taken for these primitives is fairly deterministic, as the DIVA instruction set is very generic and uses only memory operations (*Load* and *Store*) for accessing the network interface (PBuf). The parcel payload size is 32 bytes, so any message greater than 32 bytes can be divided into multiple parcels and transferred using one pair of *Send* and *Receive*. This can be done in $\lceil m/32 \rceil$ *Sends* and *Receives*, where m is message length in bytes.

The bottleneck in this communication mechanism is the boundary between the PBuf and 352-bit packet serializer (send bridge). The serializer is held for 11 PiRC clock cycles ($\text{Cycle}_{\text{PiRC}}$) to convert a packet into eleven 32-bit flits, and it takes an additional 6 $\text{Cycle}_{\text{proc}}$ of handshaking to move the next packet into the serializer and start its serialization. In this implementation of DIVA, $\text{Cycle}_{\text{PiRC}}$ is twice that of $\text{Cycle}_{\text{proc}}$, and thus this bottleneck takes 28 $\text{Cycle}_{\text{proc}}$. The time taken by *Send* and

Receive operation ($25 \text{ Cycle}_{\text{proc}}$ each) is evident for one packet send, but the time taken by other *Send* commands overlaps with the above described bottleneck as another parcel can be written into the PBuf while the serializer is transferring the previous parcel. So, throughput for PingPong is dictated by the serializer bottleneck.

Below is a timing expression for the time taken (Δt) by an m -byte message transfer from one PIM node to another and back to the first node.

$$\Delta t = 2(56 + 28 \lceil m/32 \rceil) \text{ Cycle}_{\text{proc}} \quad (1)$$

Based on standard throughput calculations for PingPong [10], the throughput (Γ) is given by the expression below for an m -byte message transfer in $(\Delta t/2)$ seconds.

$$\Gamma = m / (\Delta t/2) = 2m / \Delta t \quad (2)$$

2.2 PingPing

This benchmark also belongs to the *single message transfer* category. The importance here is on the outgoing message being obstructed by the incoming message. A single message is sent from each node to the other node concurrently. Unlike PingPong, the message is not expected to be returned to the sender.

```

if (  $\lfloor \lceil m/32 \rceil / 3 \rfloor \geq 1$  )
{
    for ( i = 1; i =  $\lfloor \lceil m/32 \rceil / 3 \rfloor$ ; i = i+3 )
    {
        Send packet i;
        Send packet i+1;
        Send packet i+2;
        Receive packet i;
        Receive packet i+1;
        Receive packet i+2;
    }
}

if (  $\lceil m/32 \rceil \% 3 \geq 1$  )
{
    for ( i = 1; i =  $\lceil m/32 \rceil \% 3$ ; i = i+1 )
    {
        Send packet i;
    }
    for ( i = 1; i =  $\lceil m/32 \rceil \% 3$ ; i = i+1 )
    {
        Receive packet i;
    }
}

```

Figure 1. Algorithm for PingPing

Fig. 1 shows the algorithm for one node. The progression of two simultaneous sends at the two PIM nodes (because of bi-directional channels) is used here to overcome the above bottleneck and hide the latency of the communication network. The idea simply makes use of the idle time available or latency of the network (wasted in PingPong waiting for the packet) for other *Send* operations on each PIM node before the packet arrives from the other node. A simple analysis of the DIVA communication mechanism revealed that using 3 continuous *Send* operations, followed by 3 continuous *Receive* operations, can best overcome the serializer bottleneck and hide the latency of the network, and the resulting time taken (Δt) depends totally on the execution of *Send* and *Receive* commands. The approach here is to defer the *Receive* operation until the maximum number of packets can be received without any delay due to the serializer bottleneck (because they are buffered at the receiving node). The time is used for sending other packets otherwise.

The timing expression for PingPing is given below.

$$\Delta t = [50 \lceil \lceil m/32 \rceil / 3 \rceil + (28k + 56)q] \text{ Cycle}_{\text{proc}} \quad (3)$$

$$k = \lceil m/32 \rceil \% 3 \text{ and,}$$

$$q = \begin{cases} 0 & \text{if } k = 0 \\ 1 & \text{if } k \neq 0 \end{cases}$$

The throughput (Γ) is given below for an m -byte message transfer per Δt seconds. Please notice that throughput is not scaled by a factor of 2 (actual transfer rate is $2m$ bytes per Δt seconds) because PingPing is a single message transfer benchmark.

$$\Gamma = m / \Delta t \quad (4)$$

2.3 SendReceive

This benchmark belongs to the *parallel message transfer* category. The benchmarks belonging to this category aim to measure performance under global load, and performance is measured for the simultaneous execution of processes at different nodes. In a SendReceive configuration, each node n sends a message to its neighbor PIM node $(n+1) \bmod N$ and receives a message from another neighbor node $(n-1) \bmod N$. The algorithm for the SendReceive is similar to algorithm provided in fig. 1, except for the fact that each PIM node n sends a message to its neighbor node $(n+1) \bmod N$ and receives a message from another neighbor $(n-1) \bmod N$.

The timing expression for SendReceive remains essentially similar to that of PingPing, with one difference: the time taken (Δt) now depends on the maximum length message to be transferred among any pair of nodes.

$$\Delta t = [50 \lceil \lceil \max(m_n)/32 \rceil / 3 \rceil + (28k + 56)q] \text{ Cycle}_{\text{proc}} \quad (5)$$

Throughput (Γ) is calculated taking into account the number of messages (N_{Msg}) injected and received by a particular node [10].

$$\Gamma = N_{\text{Msg}} m / \Delta t = 2m / \Delta t \quad (6)$$

2.4 Exchange

This benchmark also belongs to the *parallel message transfer* category. In an Exchange configuration, each PIM node n sends and receives a message to and from its neighbor nodes $(n+1) \bmod N$ and $(n-1) \bmod N$. This can be viewed as two phases of SendReceive. In the first phase, each PIM node n sends a message to its neighbor node $(n+1) \bmod N$ and receives a message from another neighbor $(n-1) \bmod N$. In the second phase, each node n sends a message to node $(n-1) \bmod N$ and receives a message from another node $(n+1) \bmod N$. The timing expression is given below.

$$\Delta t = \{ [50 \lceil \lceil \max(m_{n,0})/32 \rceil / 3 \rceil + (28k_0 + 56)q_0] + [50 \lceil \lceil \max(m_{n,1})/32 \rceil / 3 \rceil + (28k_1 + 56)q_1] \} \text{ Cycle}_{\text{proc}} \quad (7)$$

For throughput (Γ) calculations, N_{Msg} is 4 because each node sends a message to both of its neighbors and receives a message from both.

$$\Gamma = N_{Msg} m / \Delta t = 4m / \Delta t \quad (8)$$

2.5 Barrier Synchronization

This benchmark belongs to the *collective communication* category, where a message is communicated among a group of processes simultaneously. Barrier Synchronization is one of the most important operations required in high performance parallel computing. Various solutions have been presented thus far to solve this problem efficiently, either by providing hardware support [15], implementing algorithms in software [16][17] or hybrid approaches [18][19][20]. The DIVA communication mechanism provides no hardware support for collective communication. Thus, an efficient algorithm is required to implement this important operation. Although one of many available algorithms [16][17] can be tailored for the DIVA platform for this purpose, a unique algorithm is desirable to make efficient use of the hardware resources available in a PIM environment.

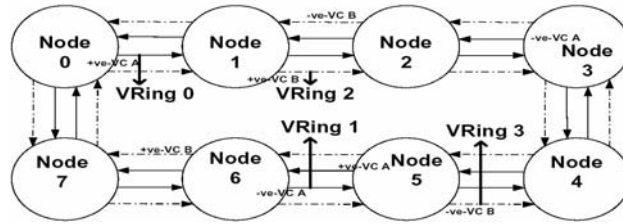


Figure 2. 8-node network explaining the concept of Virtual Rings

DIVA PIM nodes are connected in a bi-directional ring configuration in a memory system. The concept of a Virtual Ring is introduced for collective communication algorithms. A *Virtual Ring* is defined as the complete traversal on a particular virtual channel (VC) of a physical ring (in a particular direction, i.e. +ve or -ve). Thus, there can be a maximum of four Virtual Rings in a DIVA system. VRing0, VRing1, VRing2 and VRing3 are defined as the traversal on VC A, VC A, VC B, and VC B virtual channels of +ve, -ve, +ve and -ve direction physical rings respectively (fig. 2). The nodes are labeled, as n , in an incremental manner from 0 to $N-1$ (where N is the total number of nodes in the ring).

The presented algorithm reduces the required barrier message transfer stages (phases) to $\log_2 N$ by forming a balanced binary tree. The number of phases (P) is optimum because a message can be distributed optimally among N nodes (without any multicast or broadcast capability) by forming a balanced binary tree. Thus,

$$P = \log_2 N \quad (9)$$

In each phase p , a node n sends barrier message *Barrier* to the destination node (or Send node), $S_{(n,p)}$, given by the expression below.

$$S_{(n,p)} = \begin{cases} [n + (N/2)(1/2^p)] \bmod N & \forall n : n \in N_1 \\ [n - (N/2)(1/2^p)] \bmod N & \forall n : n \in N_2 \end{cases} \quad (10)$$

Where, $N_1 = \{0, 2, \dots, N-2\}$ & $N_2 = \{1, 3, \dots, N-1\}$

Here, a barrier message *Barrier* consists of a single packet (256 bit payload). The barrier message received needs to be combined with the local barrier message and treated as a local barrier message for further phases. In other words, a message combination step is needed before the algorithm can begin the next phase at a particular node. The result is called the *Combined Synchronization Signature*. Further, to detect the completion of a required barrier message in a particular phase an *Expected Phase Signature* (again 256 bit wide) for each phase p is maintained at every node n .

The out of order arrival of packets at a particular node must also be addressed, since different nodes may start the barrier operation at different times. Barrier messages from nodes expected in later phases can arrive and be buffered at the PBuf and received by the current node in any order. This issue is resolved by checking whether the current phase is completed (i.e. whether expected barrier message has already been received) before entering into the *Receive* operation. Otherwise, a process will wait for the packet forever that has already been received before. These issues led to a modification of the *Receive* command for barrier synchronization. The *Send* command remains the same as for the previous operations.

Below is the expression for the virtual ring $R_{(n,p)}$ to be used by node n in phase p .

$$\begin{aligned} R_{(n,0)} &= n \bmod N_{VRings} \\ R_{(n,p)} &= R_{(n,0)} \quad \forall p: p \in \{1, \dots, P-1\} \end{aligned} \quad (11)$$

The ring $R_{(n,p)}$ used does not change with phase p . This means that contention for links is not an issue for this algorithm, even if execution of *Barrier Send* and *Receive* operations by different nodes gets out of synchronization. The channel utilization is full in the first phase. After that channels are available between two nodes, but not required by the algorithm. However, every virtual ring is used in all phases as messages are sent on minimal paths, which are disjoint in a particular phase.

The timing expression for the lower bound (when every node starts the barrier operation at the same time) on the time taken to reach complete synchronization is given below. This expression basically accumulates the single message transfer time in each phase, which is similar to the PingPong timing expression (except *Receive* takes 39 cycles). The calculations required for the destination node and ring to be used in each phase can be computed by the kernel in the initialization phase for a user program, which in turn initializes the route cache [8] so that user-level object addresses may be translated appropriately.

$$\Delta t_{lower_bound} = [96 P + 2N (1 - (1/2)^P)] \text{ Cycle}_{proc} \quad (12)$$

The timing expression for the upper bound is also given below. Here, T_n is defined as the start time of barrier operation on node n . This worst case arises when all potential senders (in different phases) of *Barrier* to node n , which started barrier operation at $\min \{T_n\}$, start the barrier operation at a time $\max \{T_n\}$.

$$\Delta t_{upper_bound} = \Delta t_{lower_bound} + \max \{T_n\} - \min \{T_n\} \quad (13)$$

2.6 AllToAll Personalized Exchange

This benchmark also belongs to the *collective communication* category, as discussed above. The presented algorithm is of the *Direct* type. This type of algorithm was preferred for DIVA, since the communication startup time and latency can be *completely* hidden (for up to 8 hop distance) by using the previously presented method of 3 continuous *Sends* and *Receives* (section 2.2), with only message transfer dominating. Although DIVA tries to reduce communication by sending pointers, functions and arguments only, messages lengths can be large for applications where data use is not localized. Moreover, the message *combination and forwarding* step, used in Indirect algorithms, would have been simply overhead because the same message could have been sent to the destination node directly instead of some intermediate node.

This algorithm achieves a lower bound on the number of phases P possible for the communication of all messages with full utilization of all channels and in terms of messages required to be sent by one node. A total of $N(N-1)$ (every node sends to each node except itself) messages are required to be sent over $(N_{VC} \times N_{Directions})$ - channels, which is equivalent to N_{VRings} . Here, N_{VC} is the number of virtual channels, $N_{Directions}$ is the number of directions in the ring, and product of the former two gives, N_{VRings} , the total number of virtual rings in the network. Messages are exchanged (3 continuous *Sends* and *Receives* method), rather than only sent, in each phase. This further reduces the number of phases by a half (factor of 2 in the denominator). These messages can be exchanged on any of the N_{VRings} rings, following non-minimal paths also. The result is the expression below.

$$\begin{aligned} P &= [N(N-1)] / 2 N_{VC} N_{Directions} \\ &= [N(N-1)] / 2 N_{VRings} \end{aligned} \quad (14)$$

The above value of P is equal to $N-1$ for $[N=8, N_{VRings} = 4]$, $[N=4, N_{VRings} = 2]$ and $[N=2, N_{VRings} = 1]$. Here, N_{VRings} changes with N because required number of virtual rings decreases with N .

In each phase p , a node n sends a message the destination node, $S_{(n,p)}$.

$$S_{(n,p)} = \begin{cases} (N - n - p - 1) \bmod (N - 1) & \forall n : n \in N_1 \text{ \& if } (N - n - p - 1) \bmod (N - 1) \neq n \\ N - 1 & \forall n : n \in N_1 \text{ \& if } (N - n - p - 1) \bmod (N - 1) = n \\ (3p) \bmod (N - 1) & n : n = N - 1 \end{cases} \quad (15)$$

Where, $N_1 = \{0, 1, 2, \dots, N-2\}$

Below is the expression for the virtual ring $R_{(n,p)}$, to be used by node n in phase p .

$$R_{(n,0)} = \begin{cases} n & \forall n : n \in N_1 \\ N - n - 1 & \forall n : n \in N_2, N_3 \end{cases} \quad (16)$$

$$R_{(n,p)} = \begin{cases} R_{(n,p-1)} & \forall n : n \in N_1 \text{ \& if } (N - n - p - 1) \bmod (N - 1) \geq n \\ (R_{(n,p-1)} - 1) \bmod N_{VRings} & \forall n : n \in N_1 \text{ \& if } (N - n - p - 1) \bmod (N - 1) < n \\ (R_{(n,p-1)} - 1) \bmod N_{VRings} \text{ \& } j = 0 & \forall n : n \in N_2 \text{ \& if } (N - n - p - 1) \bmod (N - 1) < n \text{ \& } j = 0 \\ (R_{(n,p-1)} - 1) \bmod N_{VRings} \text{ \& } j = 1 & \forall n : n \in N_2 \text{ \& if } (N - n - p - 1) \bmod (N - 1) = n \text{ \& } j = 0 \\ R_{(n,p-1)} & \forall n : n \in N_3 \text{ \& if } j = 1 \\ R_{(n,p-1)} & \forall n : n \in N_3 \text{ \& if } p = \text{Even} \\ (R_{(n,p-1)} - 1) \bmod N_{VRings} & \forall n : n \in N_3 \text{ \& if } p = \text{Odd} \end{cases}$$

Where, $N_1 = \{0, 1, \dots, N/2-1\}$, $N_2 = \{N/2, N/2+1, \dots, N-2\}$ & $N_3 = \{N-1\}$

Contention for the virtual rings (or links) at the source nodes will arise for this algorithm if message lengths used by different nodes are unequal in one phase. In such a case, it would be desirable for some nodes to start their next phase early, and thus increase the traffic in the network (because of fair flow control in DIVA [9]), but this would lead to contention. This contention problem will not harm the functionality, but will increase the time taken by the algorithm because message transfer times in the allocated phases are ideally overlapped (instead of cumulative) with other message transfers. Thus, synchronization between phases is desired for AllToAll direct algorithms [12][14] for better performance. This synchronization can be achieved statically by a compiler by padding small messages with dummy messages, to make the message lengths consistent. Alternatively, a barrier command can be used after each phase. Obviously, there is a tradeoff between using a barrier operation for synchronization and simply allowing the AllToAll operation to proceed asynchronously. If the time overhead due to a barrier operation is larger than the penalty due to asynchronous treatment of differing message lengths, then the latter method is preferred.

The channel utilization is full in all phases, as messages are sent on the non-minimal disjoint paths in a particular phase. The expression for the time taken to perform a complete personalized exchange on a synchronous run of the algorithm is given below. This expression accumulates the single message transfer time in each phase, which is the PingPing time for the largest message.

$$\Delta t = \sum_{p=0}^{P-1} [50 \lceil \lceil \max(m_{n,p})/32 \rceil / 3 \rceil + (28 k_p + 56) q_p] \text{Cycle}_{\text{proc}} \quad (17)$$

3 Performance Results and Analysis

The timing expressions for five of the benchmarks are plotted with respect to variation in the message length (fig. 3). The processor speed of 140 MHz ($\text{Cycle}_{\text{proc}}$ of 7.14ns) is chosen for this purpose, corresponding to the actual speed achievable for DIVA-II PIM chips assembled in DIMM boards and integrated in the memory system of an Hewlett-Packard Itanium2-based zx6000 server. The message length is varied from 1 byte to 4096 bytes in powers of 2, i.e. 2^i for $i=0$ to 12. Barrier synchronization is not shown in this graph because it is defined only for a single message length of 32 bytes. The throughput expressions are plotted for the *single* and *parallel message transfer* category benchmarks in fig. 4, as throughput is not reported for the *collective communication* category.

As seen in fig. 3, PingPong time is double that of PingPing time for message lengths up to 64 bytes, but then the difference grows with increase in the message length. This is because of the communication latency hiding technique used for PingPing, which results in reasonable time savings for large message lengths. The PingPong throughput (fig. 4) is equal to the PingPing throughput for up to two sends, and then it decreases because of the above reason. Throughput, in general, increases initially and then tends to saturate for higher message lengths. Because the impact of communication latency involved in the timing expressions of all benchmarks (for

non-multiple of 3 message lengths in PingPing based expressions) becomes lesser for larger message lengths, throughput reaches the capacity limit of the underlying network. The time required for SendReceive is the same as that for PingPing, but throughput is double because total message turnover count is double (it belongs to the *parallel message transfer* category). The time taken by Exchange is twice that of SendReceive, but throughput is the same because total message turnover count is also double.

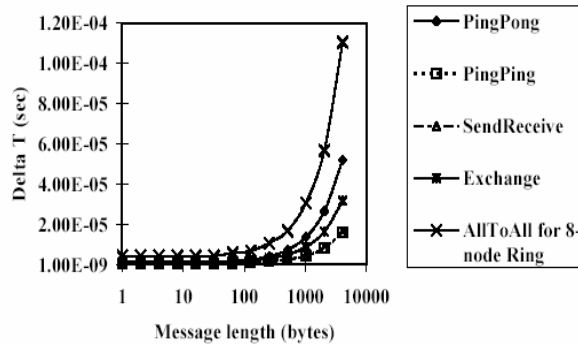


Figure 3. Time taken for different message lengths (except for Barrier)

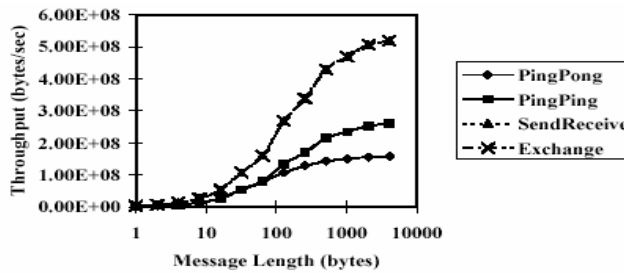


Figure 4. Throughput achieved for different message lengths

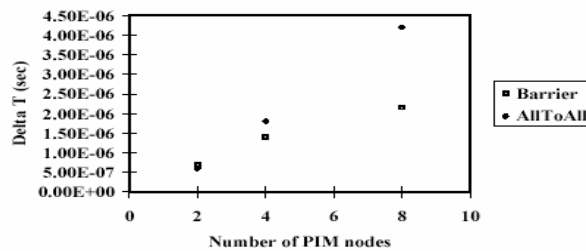


Figure 5. Time taken by Barrier and AllToAll for different number of nodes

Fig. 5 shows the plots for the time needed for the Barrier operation (lower bound) and AllToAll communication with variation in the total number of PIM nodes, i.e. for $N=2,4,$ and $8,$ connected in a bi-directional ring configuration, as is the case for

DIVA. Barrier takes more time for $N=2$ than AllToAll, because the Barrier *Receive* command takes more clock cycles due to synchronization signature matching for each phase. Barrier is implemented as an Indirect algorithm, but this does not imply that an Indirect algorithm will work better for AllToAll, as discussed in section 2.6.

DIVA PIMs have been assembled into DIMM boards (see fig. 8 in [8]) and inserted into the memory system of a Hewlett-Packard Itanium2-based zx6000 server². Instrumented measurement experiment was conducted for PingPong benchmark for a 32-byte message transfer to validate the analysis. This experiment tested the path of parcel flow from and to the *user space* through the pbuf and PiRC. The analysis was found to be in perfect agreement with the measurement on the hardware system for this case. This experiment also verified the analyzed clock cycles taken by *Send* and *Receive* communication primitives. This indicates that analyzed performance results for other benchmarks should also be in close agreement with measurement on the hardware system.

The results cannot be fairly compared with those provided in [10] for MPI implementations on specific machines. Our analysis assumes the DIVA specific optimized implementations of communication primitives, and hence, a comparison with the results of generic MPI implementations is not appropriate. Given that DIVA is a unique architecture using PIMs, it is difficult to draw fair comparisons with general-purpose architectures, especially when considering the limited silicon area used for implementing DIVA as opposed to full-scale multiprocessors. Also, there are currently no other PIM communication results for comparison. It is simply noted that DIVA achieves sub-microsecond user-level messaging, including software overheads, in contention-free cases and also performs reasonably well for collective communication using lightweight network structures and protocols [8][9]. For more details on how other architectures perform for these communication operations, the interested reader can refer to [10] and other survey literature.

4 Conclusion and Possible Improvements

Every design, in spite of the best efforts of the designer, has some scope of improvement, and such is the case with DIVA. First, if a dedicated pbuf bus is used, pbuf load and store operations would take only one clock cycle, as they would not require arbitration through a memory controller. Secondly, RDMA capability, which has become standard in most state-of-the-art interconnection networks, is desirable, so that processor intervention is not needed in the normal case. At a minimum, a remote memory write capability would free the receiving node processor from waiting for *Receive* data (polling mode) or context switching costs (interrupt mode). Thirdly, the interaction between the send part of the pbuf and serializer, which is the bottleneck in the current design, can be greatly improved. Fourthly, providing two serializers and doubling the FIFO space in the send part of the pbuf would result in greater per-node throughput in many cases. Finally, providing some hardware support for collective

² We thank Tim Barrett for providing DIVA PIM testing platform on zx6000 server.

communication and using a hybrid approach will definitely help systems containing larger number of PIM chips.

To summarize, user-level messaging performance in PIM to PIM communication is modeled and analyzed for the DIVA system in this paper. The benchmarks used for this purpose are PingPong, PingPing, SendReceive, Exchange, Barrier synchronization and AllToAll personalized exchange. A significant part of this evaluation lies in the formulation of optimal algorithms for Barrier synchronization and AllToAll personalized exchange for the bi-directional ring configuration of upto 8 DIVA PIMs in the memory system of an HP zx6000 server. The expressions for timing and throughput, as applicable, are derived for the above benchmarks, and results are thoroughly analyzed to provide insight. The results show that the currently employed communication mechanism can be used very efficiently, for collective communication operations also.

References

- [1] J. Liu, B. Chandrasekara, et al. Microbenchmark performance comparison of high-speed cluster interconnects. *IEEE Micro*, Volume 24, Jan-Feb 2004, pp 42-51
- [2] P.H. Worley. MPI performance evaluation and characterization using a compact application benchmark code. *MPI Developer's Conference*, July 1996, pp 170-177
- [3] S.G. Pestana, et al. Cost-performance trade-offs in networks on chip: a simulation-based approach. *Design, Automation and Test in Europe Conference and Exhibition*, Volume 2, Feb 2004, pp 764-769
- [4] J. Draper, et al. The Architecture of the DIVA Processing In Memory Chip. *Supercomputing*, June 2002
- [5] J. Draper, et al. Implementation of a 32-bit RISC processor for the Data-Intensive Architecture processing-in-memory chip. *Application-Specific Systems, Architectures, and Processors*, 2002
- [6] J. Draper, et al. Implementation of a 256-bit wide word processor for the Data-Intensive Architecture processing-in-memory chip. *28th European Solid-State Circuit Conference*, September 2002
- [7] S. Mediratta, et al. A 0.18um CMOS implementation of an area efficient precise exception handling unit for processing-in-memory systems. *Midwest Symposium on Circuits and Systems*, July 2004
- [8] S. Mediratta, C. Steele et al. An area efficient and protected network interface for processing-in-memory systems. *International Symposium on Circuits and Systems*, May 2005
- [9] S. Mediratta, J. sondeen, J. Draper. An area efficient router for Data-Intensive Architecture (DIVA) system. *International Conference on VLSI Design*, Jan 2004.
- [10] Pallas MPI Benchmarks. <http://www.pallas.com/e/products/pmb/documents.htm>
- [11] T. V. Eicken, et al. Active messages: a mechanism for integrated communication and computation. *ISCA*, May 1992.
- [12] Y.C. Tseng, et al. Bandwidth-optimal complete exchange on wormhole-routed 2D/3D torus networks: a diagonal-propagation approach. *IEEE Transactions on Parallel and Distributed Systems*, April 1997
- [13] S.G. Kim, et al. Complete exchange algorithms in wormhole-routed torus networks: a divide-and-conquer strategy. *Parallel Architectures, Algorithms, and Networks*, June 1999, pp 296-301
- [14] S. Hinrichs, C. Kosak, et al. An architecture for optimal all-to-all personalized communication. *ACM Symposium on Parallel Algorithms and Architectures*, 1994, pp 310-319
- [15] R.E. Kessler, J.L. Schwarzmeier. Cray T3D: a new dimension for Cray research. *Compton, Digest of Papers*, Feb 1993, pp 176-182
- [16] Y. Sun, P.Y.S. Cheung. Barrier synchronization on wormhole-routed networks. *IEEE Transactions on Parallel and Distributed Systems*, June 2001, pp 583-597
- [17] J.S. Yang, C.T. King. Designing tree-based barrier synchronization on 2D mesh networks. *IEEE Transactions on Parallel and Distributed Systems*, Volume 9, June 1998, pp 526-534
- [18] D.K. Panda. Fast barrier synchronization in wormhole k-ary n-cube networks with multideestination worms. *HPCA*, Jan 1995, pp 200-209
- [19] R. Sivaram., C.B. Stunkel, D.K. Panda. A reliable hardware barrier synchronization scheme. *Parallel Processing Symposium*, April 1997, pp 274-280
- [20] J.F. Martinez, J Torrellas. Speculative synchronization: programmability and performance for parallel codes. *IEEE Micro*, Nov-Dec 2003, pp 126-134