

Efficient Offloading of Collective Communications in Large-Scale Systems

José Carlos Sancho, Darren J. Kerbyson, Kevin J. Barker

Performance and Architecture Laboratory (PAL)
Computer Science for HPC (CCS-1)
Los Alamos National Laboratory, NM 87545, USA
{jcsancho,djk,kjbarker}@lanl.gov

Abstract—In parallel applications communication overheads generally increase as the processor count increases and in particular, collective communication operations can become a critical limiting factor in achieving high performance. In this paper we explore a novel technique to boost application performance by dedicating some processors in the system to collective operations. We demonstrate the viability and efficiency of this approach for the *Allreduce* collective operation on a state-of-the-art cluster. Experimental results show that the collective latency can be reduced by 30% and that the communication overhead per processor is also very low, at 1.6 μ s, which represents one order of magnitude higher performance than with conventional implementations. Moreover, results on a large-scale scientific application (POP) show that this approach achieves 15% higher performance on 640 processors than when using the default collective implementation.

I. INTRODUCTION

The single program multiple data (SPMD) programming model is generally the preferred programming model in parallel scientific applications as they usually exhibit a rich degree of data-parallelism. In this model the global simulation domain must be divided across the available processing elements where similar operations are performed on different sub-grids at the same time. Unfortunately, when running at large-scale the communication costs can become a critical limiting factor to achieve high performance. This is due to the communication costs typically increasing in direct proportion to the processor count. This is especially true with collective communication operations such as the *Allreduce* primitive [1].

In scientific computing the *Allreduce* collective is heavily used in solving a sparse system of linear equations in many scientific applications [2]. The basic operation of this collective consists of several communication and computation steps to combine one or more values from every processor and distribute the result back to all processors. Some high-performance cluster interconnection networks, such as Quadrics QS-Net and InfiniBand, provide hardware support in their network interface cards (NICs) for some collective operations in order to accelerate and offload the message handling from the main processor. Offloading is important because it allows for overlapping communication with other communication/computation activities of the application. However, up to now the applicability of these network co-processors to carry out collectives primitives has been very limited as they do not

perform floating point operations in hardware that are often required.

Moreover, custom-designed supercomputers such as the Blue Gene/L [3] and the Intel Paragon XP/S [4] machines dedicate some of their processing elements to specifically perform communication related operations. For example, in the Blue Gene/L machine there is a mode of operation (known as co-processor mode) in which a processor is paired with another processor dedicated to handle its communication tasks (collective and point-to-point). This mode of operation is fully supported in hardware, and thus it is transparent to the application requiring no changes to the conventional SPMD programming model. This approach is more appealing in large-scale systems as the use of a NIC may become a major potential bottleneck for a node as the node size, in terms of number of processing elements, increases due the expected rapid expansion in number of cores per socket. Enhancing network interface cards to reduce this congestion is not a scalable solution as it will encounter power and PCI bandwidth constraints. On the other hand, host processors have the advantages of having a much faster processing capabilities, a faster access to memory, and a higher proximity to other processors within the node than their processor counterparts in the NIC. Therefore, host processors are more suitable to accelerate and offload communication operations. However when using communication co-processors, the typical scheme employed uses a fixed number of dedicated communication processors (DCPs), and often takes a conservative approach where the number of DCPs can be very large (for example, half of the available processors in the Blue Gene/L machine) which clearly has the disadvantage of reducing the capability of the system for application processing.

In this paper, we explore a novel approach to increase application performance in large-scale systems by efficiently dedicating one or more processors in the system to perform collective communications, and thus act as support to the other processors running the application. The number of dedicated collective processors in our approach that we term CDCP (*Configurable Dedicated Collective Processors*) is configurable to each application. The number of DCPs required depends on the characteristics of the application which allows us to drastically reduce the overhead of DCPs in the system with respect to conventional approaches. Moreover, there is no

additional hardware required to implement this approach as it can be constructed using the standard MPI communication library. Thus, providing portability and cost-effectiveness to many systems.

This new approach is a substantial departure from the SPMD programming model to the MPMD (multiple program multiple data) as not all processors execute the same work-flow of the application. This transformation to a MPMD is totally transparent to the application that, from the point of view of the programmer, it is still SPMD. We demonstrate in this work that parallelizing the application work-flow in this way significantly achieves better performance than the classical SPMD parallelization of the application. In this research we quantify the possible performance benefits and empirically demonstrate the viability and efficiency of this scheme in a current commodity large-scale cluster. The prototype implementation focuses on the *Allreduce* collective. The results show that the CDCP approach is able to significantly increase the performance of the *Allreduce* – its latency can be reduced by up to 30% on 512 processors. In addition, the CDCP can effectively hide the communication costs of the collective primitives with other communication/computation activities as the overhead per processor of the CDCP approach is very low at 1.6 μ s, and the required time to fully hide the collective costs is 3 \times lower than that required for other implementations. Moreover, results from *Parallel Ocean Program* (POP) show that the CDCP approach significantly increases its performance by 15% with respect to the conventional SPMD scheme when using the same number of processors. This indicates that the CDCP approach is able to provide a more efficient use of the computational resources in the system than the one achieved by the SPMD model.

The rest of this paper is organized as follows. Section II summarizes the most relevant previous work on improving the performance of collectives. In Section III we briefly describe the typical implementation of the *Allreduce* collective. Section IV describes our non-blocking approach for collectives. Section V describes the experimental results. And finally, conclusions are given in Section VI.

II. RELATED WORK

There has been a lot of research focused on improving the performance of collectives in the past due to their high impact on application performance. We consider the most relevant into the three sub-sections below.

– **Collectives Assisted by NIC Processors** The first collectives offloaded to NICs were the *Barrier* and *Broadcast* primitives [5]. They were implemented in Asynchronous Transfer Mode (ATM) network adapters. The NIC was able to significantly accelerate these operations because they are not computation bound and the NIC was able to provide a much faster response than the host processor. More recently other research has explored the offloading of collectives including *Allreduce* and *Reduce* [6], [7]. Integer and floating point operations were evaluated for various collective data sizes. Because the NIC didn't have hardware floating point,

those operations were emulated using software libraries on the embedded processors. Results showed that although this scheme provides a better tolerance of the process skew, the performance of these collectives were significantly worse than their counterpart implementation in the host processors. The major cause for the degradation was the slower speed of the NIC processors with respect to the host processor that resulted in slower floating point performance for those collectives. This behavior is likely to also occur in current HPC networks as the gap in performance between the NIC and host processors remains high. We believe that this trend is likely to continue in the future due to the restricted power of NICs and the limited I/O bandwidth within a node.

In addition, there have been studies that improved collective performance by using Remote Direct Memory Access (RDMA) that is available in most modern networks. Gupta showed that collectives can take advantage of this feature to significantly increase their performance [8]. Tipparaju [9] and Wu [10] combined the RDMA capabilities with shared memory support available within an SMP node to accelerate collectives. All these techniques complement our own approach as they can be incorporated to improve the performance for specific systems.

– **Collectives Assisted by Dedicated Host Processors** Two machines that are representative of this scheme are the Intel Paragon XP/S [4] and the Blue Gene/L [3]. In both machines one or more of the host processors can be dedicated to carry out the communication operations of the other's. The Blue Gene/L machine contains only two processors per node while the number of processors per node in the Intel Paragon XP/S varied from 2 to 5 depending on the machine. Thus, the ratio between DCPs and application processors is between 1:1 and 1:4. Our approach also dedicates a set of processors to act as communication processors. However, a main difference is that the number of dedicated processors is flexible and can be configured to the needs of each application.

– **Non-blocking Collective Libraries** Recently, a communication library was developed by Hoefler called *NBC* that also provides support for non-blocking collectives [11]. Non-blocking collectives are not in the MPI standard today, but there are currently active discussions in the community to decide whether it should be integrated into the standard or not in the future. Preliminary results on the *NBC* library showed that such a technique is a viable approach to improve the performance of some scientific applications on small-scale clusters encouraging the inclusion of non-blocking collectives in the MPI standard. Our approach also provides support for non-blocking collectives, but unlike the *NBC* scheme, we don't execute the application on the processors that the collective operation is offloaded in order to maximize the parallelism between collective operations and other application communication/computation activities.

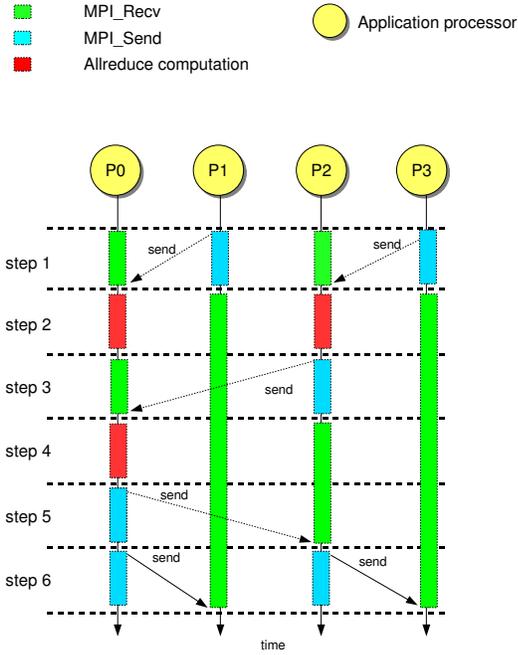


Fig. 1. Conventional implementation of the *Allreduce* collective.

III. BACKGROUND

A. The Basic *Allreduce* Implementation

The *Allreduce* collective operation performs a calculation (such as summation, maximum, or minimum) on data located on each processor and then distributes the result back to every processor. Various efficient algorithms consisting of a series of computation and communication steps have been developed to perform this operation. The most commonly used algorithm employs a binomial tree because it has a regular structure, and thus is easily implemented. This algorithm typically requires a total of $2 \times \log(N)$ communication steps and $\log(N)$ computation steps where N is the number of application processors (assumed here to be a power of two for simplicity). Generally, because the high performance of today's processors, the communication costs represent a greater proportion of the overall collective cost. For illustration purposes, Figure 1 depicts the steps required on a 4-processor system using the blocking MPI Send/Recv functions. Two binomial trees are required to complete the communications: one to reduce the data to a single root and a second to distribute the result. This last distribution can be done in one communication operation when the network has hardware support for data broadcast. Also, note that there are dependencies between steps, for example *step 4* in Figure 1 cannot begin until *step 3* completes.

B. The Non-blocking *Allreduce* Implementation

In general, collective operations allow applications to overlap communication with available computation through the use

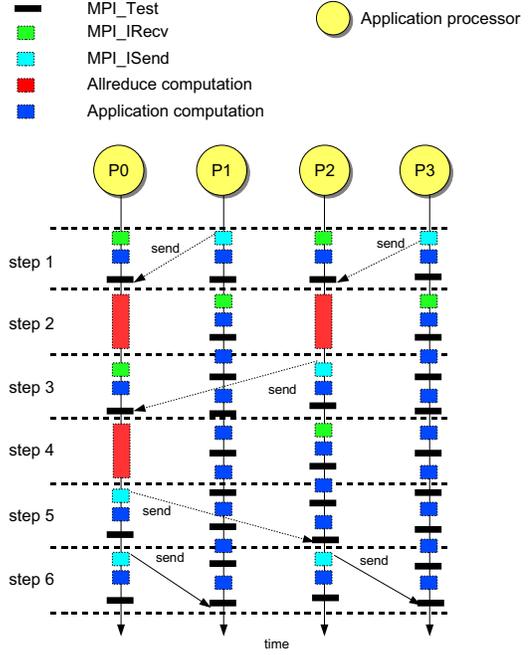


Fig. 2. Non-blocking implementation of the *Allreduce* collective (NBC library).

of non-blocking send/receive functions provided by the message passing library. Application performance can be benefited as the processor wastes fewer cycles waiting to receive data and therefore can perform application computation in advance of, rather than after, the collective operation's completion. Figure 1 illustrates this approach. P_1 initiates the receive operation in *step 2* although the data is not received until *step 6*. During this elapsed time the processor is idle even though it could perform useful computation. Note that a data dependency analysis is required in order to ensure correctness when overlapping in this manner.

The optimal amount of computation that can be scheduled between the initiation of the receive operation and the arrival of data is not obvious. It varies from one processor to another and is unknown prior to execution. For instance, in Figure 1, processor P_0 can absorb less computation than processor P_1 . Polling or interrupt based methods can be used to determine the arrival of data and make progress on the collective. The NBC library employs the polling mechanism because generally provides more responsiveness than the interrupt based methods. This is due to the fact that the operating system is usually not responsive enough to deal with incoming messages.

Figure 2 depicts the non-blocking *Allreduce* implementation for a 4-processor system in the NBC library. Although processors are able to advance the application's execution by overlapping communication with available computation, the execution of the collective is performed by the same processors that are run the application and suffer the overheads associated

with repeatedly sending and receiving messages, calculating the reduction, and polling the MPI library.

IV. THE CDCP SCHEME

In this section, we describe our proposed technique for the MPI *Allreduce* collective as well as its implementation.

A. Algorithm Description

The CDCP (*Configurable Dedicated Collective Processors*) technique divides the processors into two sets in order to execute an application: (1) the set of *application processors*, referred to as *APP* where the main work-flow of the application is executed, and (2) the set of (*Dedicated Collective Processors*), referred to as *DCP* that actually execute the steps of the collective operations.

A mapping function assigns a unique DCP to each APP processor, $f : APP \Rightarrow DCP$. The number of DCPs may be one or more. When using more than one DCP, f is used to distribute the DCPs evenly among the APPs in order to reduce contention when multiple APPs simultaneously access the DCPs.

Each application processor is involved in the following operations:

- Send data to its corresponding DCP, and
- Receive data back from the same DCP.

In the case of a reduction collective, the operations undertaken by each DCP are:

- Receive data from all the APPs assigned to it,
- Calculate the partial reduction on the data received,
- Communicate with other DCPs, if any, to complete the reduction, and
- Distribute the result to the corresponding APPs.

Note that although this scheme can also be implemented using threads eliminating the need for DCPs, we decided to use processors rather than threads for performance purposes. Processors provide more responsiveness, less overhead (no context switching due to multi-threading), and also reduces the load imbalance produced by the calculation of the collective in the application processors.

In the CDCP scheme, two different processor distributions, between *APP* and *DCP* could be considered. The first is to assume that the application runs on $N = APP$ processors and additional processors would be used for the DCPs. The second does not require additional processors in the system and the DCPs are taken from the total processors used, i.e. $N = |APP| + |DCP|$. This later mode is used in the following analysis, that is the number of APP processors decreases as the number of DCPs increases. In both distributions, the *DCP ratio* is defined as the ratio between $|DCP|$ and $|APP|$. It represents the number of application processors assigned to each DCP, and thus determines the level of contention that a DCP will see — this is an important consideration that impacts on the performance of the CDCP scheme as will be seen later. There is an equivalence between the DCP ratio and the total number of DCPs for a particular run and distribution

mode. For example, a DCP ratio of 1:4 on a 640-processor run corresponds to $|APP| = 640$ and $|DCP| = 160$ in the case of the first distribution mode, whereas in the second distribution mode there would be 512 and 128 processors respectively. Note that in the second mode the parallelism available to the application is reduced from 640 to 512 processors in order to accommodate the DCPs.

To illustrate the CDCP approach, Figure 3 depicts the steps required for an Allreduce operation on the same number of processors as shown in Figure 1 but the number of application processors is reduced by one in order to accommodate one DCP. The application processors initiate the collective operation by sending data to the waiting DCP. Once the data is transmitted, the application processors are able to advance the computation until, at some point, they are required to wait for the completion of the collective. Note that most of the communication and computation required to calculate the reduction have moved from the application processors to the DCP. This minimizes the performance impact on the application and maximizes the parallelism between the collective operation and the application work-flow.

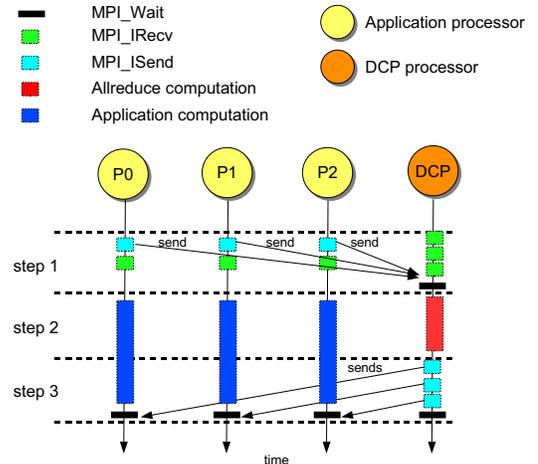


Fig. 3. Steps required to complete the non-blocking implementation of the *Allreduce* operation in the CDCP scheme using a 1:3 DCP ratio.

Among the DCPs, collectives are completed in the traditional way. Operations performed by the application processors are minimal, requiring only the initiation of the operation and a final synchronization. Note that the number of steps required to complete the *Allreduce* in the Figure 3 is reduced to half the steps required in the traditional implementation (6 steps, see Figure 1). The number of steps in the CDCP scheme is given by

$$2 \times \log(|DCP|) + \log(|DCP|) + 2$$

which corresponds to the steps required for the *Allreduce* calculation among the DCPs and the two extra steps represent the initiation and a final synchronization. It can be proved that DCP ratios equal or larger than 1:2 reduce the number of steps required compared to the traditional implementation.

In principle, reducing the number of steps might be beneficial to performance not only because of the reduction of the communication/computation operations, but also for a better tolerance to process skew due to the reduction in communication operations. However, reducing the number of steps implies that the DCP ratio is higher which in turn increases the contention in the DCPs. This contention might be high enough to offset the performance improvement achieved via the reduction in the number of steps. Thus there is a trade-off between the reduction in the number of steps and the contention generated. There is an optimal DCP ratio that reduces the number of steps that at the same time keeps the contention low — this will be analyzed in Section V.

B. Implementation Description

Because MPI implements an SPMD model, one of the main issues in its use for implementing collectives using CDCP is in the separation of the *APP* and *DCP* processor groups. A mechanism is therefore required to assign different tasks to different processors at application launch time while still allowing for communication between groups.

To implement this, we utilize the PMPI profiling interface supported by the MPI message passing library [12]. PMPI can be used to intercept all calls to the MPI library, allowing for insertion of code that can be executed either before or after an MPI call. These “wrapper” functions do not require any modification of the application source code.

At job launch time, an identical executable image will begin on all APP and DCP processors. At MPI initialization time, a PMPI wrapper routine assigns processors to each group based on information passed via environment variables. Two MPI communicators are created, allowing for independent communication between the *APP* and *DCP* groups. These communicators are substituted for the original global communicator (`MPI_COMM_WORLD`) by PMPI wrapper routines, and allow seamless communication between application processors. This is fully transparent to the application and the programmer too. Although, the underlying implementation actually follows a MPMD programming model this is not visible to the programmer, and thus from a programming point of view, the application still follows the SPMD model.

The CDCP library provides two additional functions to the MPI library named `DCP_Allreduce` and `DCP_Wait`. The former allows the application to initiate the non-blocking *Allreduce* operation and the later is to synchronize with its completion. The `DCP_Allreduce` is implemented using the non-blocking send/receives (`MPI_Isend` and `MPI_Irecv`). The synchronization routine is implemented using `MPI_Wait` to wait for the completion of the `MPI_Irecv` and `MPI_Isend` calls.

Moreover, in order to minimize the contention in accessing multiple DCPs from multiple application processors in systems using multi-processor nodes (and multi-core sockets), the DCPs are physically distributed across the nodes in the system. This distribution is illustrated in Figure 4 for an 8-processor system where $|APP| = 6$ and $|DCP| = 2$. Note that when a DCP occurs within a particular node, the other application

processors in that node are assigned to it in order to further accelerate the collective.

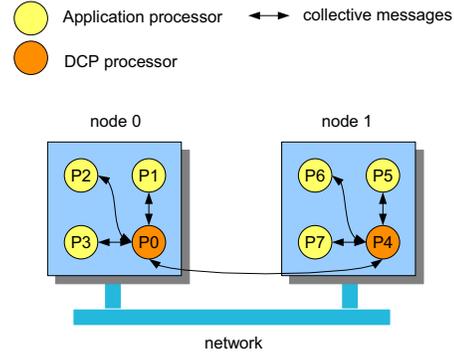


Fig. 4. Distribution of two DCPs across an 8-processor, 2-node system.

V. EVALUATION

A. Experimental Set-up

The system used in evaluating the CDCP scheme is representative of current high-performance computing platforms, with a peak performance of 4 TFlops. The system contains 256 compute-nodes interconnected with a Voltaire 288-port InfiniBand 4x SDR switch. Each node contains 2 sockets that share a NIC. Each socket contains a dual-core 2.0GHz AMD Opteron processor resulting with a total of 1,024 cores in the system. At the time of writing this paper, 640 cores were made available for this analysis. Both cores in a socket share 2GB of main memory and each node has 4GB in total. Although cores and processors are not the same concept—cores must share main memory—in this paper we use the terms *processor* and *core* interchangeably since a core can be viewed as an independent processor in parallel programming. Each node runs the Linux operating system (kernel 2.6.15.1, SMP version). The MPI message passing library used was MVAPICH-0.9.9-1168 [13]. The achievable inter-node latency for small messages is 4 μ s, and the uni-directional bandwidth is 950 MB/s for large messages on this cluster.

We evaluated the performance of the *Allreduce* based on the CDCP scheme and also for comparison purposes we evaluated the blocking scheme provided by the MVAPICH library, and the non-blocking scheme provided by the NBC library [11]. In order to better highlight the benefit of the CDCP scheme, we designed microbenchmarks that allows us to easily explore different parameters. Various collective sizes are considered in this analysis ranging from 8 up to 1024 bytes. In particular, the 8-byte size is typical in many scientific applications when using the *Allreduce* collective operation [14]. A summation is used in the *Allreduce* collective. The reported timing data are averages over 1,000 iterations.

We also evaluated the CDCP scheme on a large-scale scientific application, the Parallel Ocean Program (POP) version 2.0 [15]. This application has been also recently analyzed the

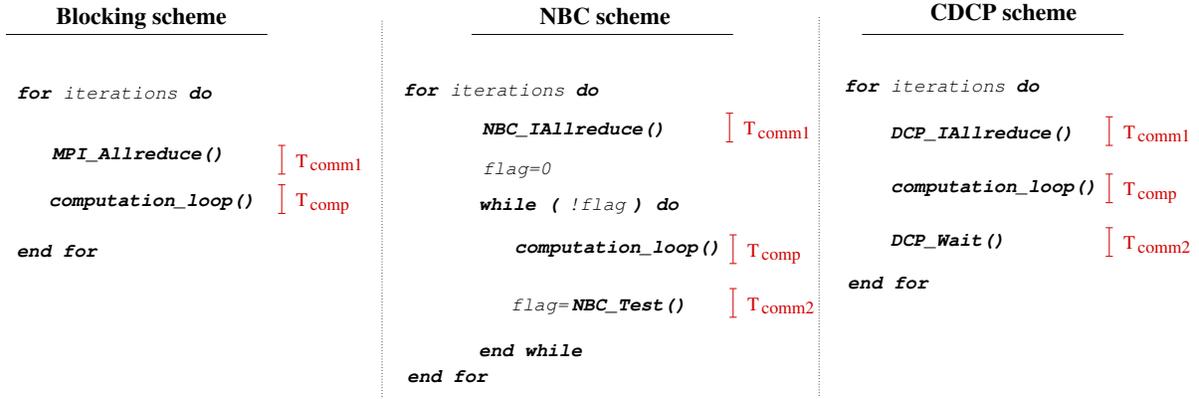


Fig. 5. Microbenchmarks for the blocking, NBC, and CDCP schemes showing time measurements.

overlapping of the point-to-point communications operations with available computation in [16]. POP is an ocean modeling code that represents water regions as 3D regular grids. POP belongs to the climate modeling applications that has been identified as an increasingly important area for high performance computing. This application runs in strong scaling mode where the global problem size remains constant and the spatial sub-grid size per processor decreases as the number of processors increases. The problem $x1$ that consists of a global problem size of $320 \times 384 \times 40$ grid points was used. For this evaluation we considered the second distribution mode for the CDCP scheme. This is the worse case because the available parallelism of the application is reduced.

B. Microbenchmarks

Figure 5 shows the pseudo-code of the microbenchmarks used in the evaluation. These microbenchmarks allows us to evaluate the behavior of the CDCP scheme in two different scenarios: in applications that exhibit some available computation/communication that can be performed at the same time as the collectives (i.e. with overlap), and in applications in which there is no such overlap. When considering the overlap case, some computation is placed between the initiation of the collective and the final synchronization for the completion. Computation times ranging from $0\mu s$ up to $200\mu s$ are considered in this case.

In these microbenchmarks we measure two important metrics: the collective latency and collective overhead. Both metrics are defined as the summation of the times T_{comm1} and T_{comm2} , but the collective latency is the time when there is no computation available to overlap ($T_{comp} = 0$). The collective latency measures the total duration of the collective which is important for applications that does not exhibit any overlap. For those applications is important to reduce the collective latency since they must wait for the completion of the collective.

On the other hand, the collective overhead is relevant for applications that exhibit some overlap. In this case, the performance of the CDCP scheme is given by the collective overhead. This overhead can be decreased when overlapping

the collective with some computation/communication activities of the application. There is a lower bound in the collective overhead as some communication costs cannot be overlapped such as the overhead of calling to the MPI library. In these evaluations we used the first distribution mode in the CDCP scheme in which additional processors are required to accommodate the DCPs in order to compare the performance of the collective in an equal job size with respect to the other schemes evaluated.

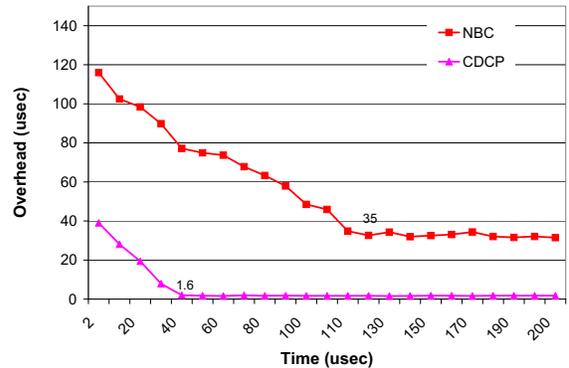


Fig. 6. Collective overhead per processor for the NBC and CDCP schemes when overlapping with available computation (512-processor job and 8-byte collective).

C. Overlapping Computation

In this section, we examined the efficiency of the CDCP scheme to overlap the communication costs of the collective with computation. We compared the performance of the CDCP scheme with the NBC scheme as it also allows for overlapping communication with computation.

Figure 6 shows the collective overhead per processor for the NBC and CDCP schemes for various computation times on a 512-processor system and using a DCP ratio of 1:4. As can be seen, as the computation time is increased the collective overhead gradually decreases in both schemes down to a

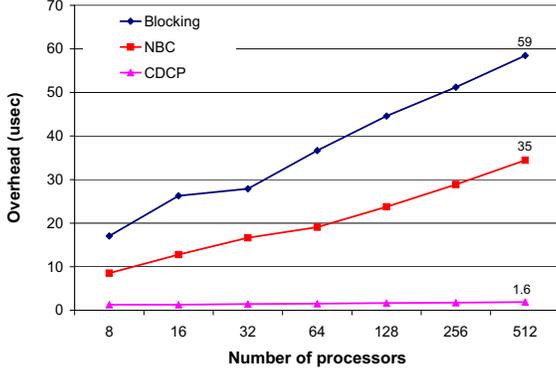


Fig. 7. Lower bound of the collective overhead per processor for the blocking, NBC, and CDCP schemes for various job sizes (8-byte collective).

lower-bound — the level at which no further communication costs can be hidden. Two important observations can be made in these results. First, the CDCP scheme achieves the lower bound earlier than the NBC scheme. In particular, the CDCP scheme achieves the lower bound at $48\mu\text{s}$ while for the NBC library the lower bound is reached at $122\mu\text{s}$. The cause for this is the high overhead introduced at small computation times in the NBC scheme due to the high frequency of the polling that prevents it to quickly reduce the communication costs. However, in the CDCP scheme there is no such polling, and thus it can rapidly decrease the communication costs. The improvement in the computation time by the CDCP scheme is a factor of roughly $3\times$. Reducing the computation time is important because at large-scale the amount of computation available to be overlapped with the collective may be small, and the CDCP scheme can effectively exploit that computation to hide the communication costs. And the second observation is that the lower bound reached by the CDCP scheme is substantially lower than the one reached by the NBC scheme. The lower bound for CDCP scheme is $1.6\mu\text{s}$ while for the NBC scheme is one order of magnitude higher, at $35\mu\text{s}$. Again, the reason is due to the overhead of polling in the NBC scheme — this is analyzed in more detail below.

D. Collective Overhead

Figure 7 shows the lower bound of the collective overhead per processor for the blocking, NBC, and CDCP schemes for various job sizes while considering a DCP ratio of 1:4. As can be seen, the CDCP scheme consistently achieves the lowest collective overhead regardless of the job size. In particular, this overhead is $1.6\mu\text{s}$ for a 8-processor job whereas the collective overhead for the NBC and the blocking scheme is $8.5\mu\text{s}$ and $17\mu\text{s}$, respectively. Moreover, the collective overhead does not vary significantly as the job size increases unlike the NBC and blocking schemes. For these schemes

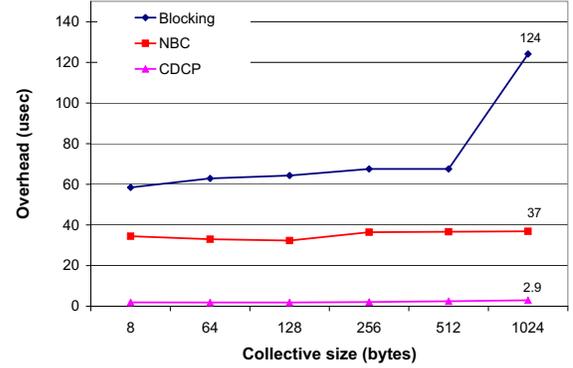


Fig. 8. Lower bound of the collective overhead per processor for the blocking, NBC, and CDCP schemes for various collective sizes (512-processor job).

the overhead is increased by a factor of $4\times$ and $3\times$ on the largest job size evaluated (512 processors). The reason for this is because the blocking and the NBC schemes execute the collective operation on the same processors that the application uses and suffers all the overheads associated to complete the collectives. This effect increases with the job size as described in Section III. However, in the CDCP scheme most of these operations have moved to the DCPs and the application processors are uniquely suffering a constant overhead of the initiation and a final synchronization which is independent of the job size.

Moreover, the increase in the collective size does not affect the relative performance of CDCP scheme with respect to the other implementations as can be observed in Figure 8. Although, the overhead of the CDCP scheme is increased up to $2.9\mu\text{s}$ on a 1024-byte collective size the performance of the CDCP scheme is still one order of magnitude higher than the other schemes.

Therefore, the greater scalability and lower overhead of the CDCP scheme makes it more appropriate for large-scale systems in order to efficiently exploit the potential overlap that is often present in applications.

E. Collective Latency

Figure 9 shows the collective latency for the blocking, NBC, and CDCP schemes for various job sizes while considering a DCP ratio of 1:4. As it can be observed the CDCP scheme achieves significantly lower latency. For instance, for a 512-processor job a reduction of 30% and $3\times$ is achieved by the CDCP scheme compared to the blocking and NBC schemes. The improvement ratio of the CDCP scheme compared to the blocking scheme is consistent among the system sizes evaluated suggesting that a similar improvement will also be found at larger processor counts. This improvement is mostly due to the DCP ratio of 1:4 used that reduces the number of communication steps to complete the collective. To

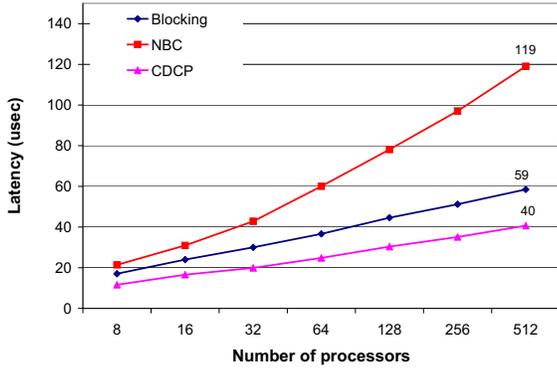


Fig. 9. Collective latency for the blocking, NBC, and CDCP schemes for various job sizes (8-byte collective).

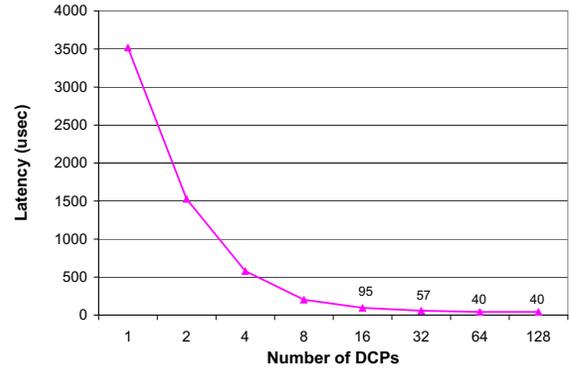


Fig. 10. Collective latency for the CDCP scheme for various numbers of DCPs in a 512-processor job (8-byte collective).

give more insight on the impact on performance of the DCP ratio, Figure 10 shows the collective latency for the CDCP scheme on a 512-processor job when scaling the number of DCPs in the system. As can be seen, although the case of 1 DCP in the system (DCP ratio of 1:512) should achieve the lowest number of steps, it is not efficient due to the higher contention generated in the DCP inevitably increases the collective latency (3500 μ s) with respect to the 128-DCP case (40 μ s). Increasing the number of DCPs drastically reduces the contention as can be also seen in Figure 10. In particular, starting at 32 DCPs (DCP ratio 1:16) the CDCP scheme is able to improve the performance compared to the blocking scheme (see also Figure 9). And the optimal DCP ratio is achieved on 64 DCPs (DCP ratio 1:8) where the collective latency reaches the lowest value, of 40 μ s.

Figure 11 shows the case of increasing the collective size for the blocking, NBC, and CDCP schemes for the 512-processor job. As can be seen, increasing the collective size negatively affects the performance of the CDCP scheme due to the higher contention generated in the DCP from sending/receiving larger messages. The performance of the CDCP scheme is reduced by 9% on a 1024-byte collective size. However, large collective sizes are not very common in scientific applications.

The collective latency can be collected by using these microbenchmarks for various collective sizes, job sizes, and various DCP ratios in order to automatically determine the optimal DCP ratio for a particular system and application. The CDCP scheme can be automatically configured to the optimal DCP ratio for the largest collective size of the application that will be the worse case. Applications could be benefited from the CDCP scheme because the reduced collective latency and overhead that is achieved in the optimal DCP ratio. Moreover, in certain cases the optimal DCP ratio can be relaxed for some particular applications. These applications are the ones that can fully overlap the collective with some computation/communication activities of the application. In

this case, because the collective can be fully overlapped the DCP ratio can be higher than the optimal DCP ratio reducing the number of DCPs required in the system. In order to not degrade the application performance the DCP ratio should be selected to the one that the collective latency is equal or lower than the amount of computation/communication time available to overlap. For example, in case that the application exhibit 95 μ s for overlapping then we can use 16 DCPs (DCP ratio 1:32) instead of the 64 DCPs (DCP ratio 1:8) required for applications that do not exhibit any overlap (see Figure 10).

Based on the optimum DCP ratio the CDCP scheme can increase the collective performance, but at the expenses of reducing available parallelism of the application due to the DCPs. For some applications the reduction in parallelism may have a higher impact on performance than the improvement in the collective performance. Therefore, not all applications will be benefited by the CDCP scheme, and the criteria basically depends on the total amount of computation time in the application versus the time spent in the collectives. This will be discussed in more detail below.

Given that the performance of an application can be generally described as the summation of the computation time (T_{comp}), the collective time (T_{coll}), and the point-to-point communication time (T_{point}), and also given the fact that the computation time is increased by a factor of the DCP ratio and the performance of the collective can be increased by a factor of 30%, then the performance of an application in the CDCP scheme can be given by

$$(1 + DCP_ratio)T_{comp} + 0.7T_{coll} + T_{point}$$

It can be proved that in order for the CDCP scheme improve the performance of an application then it must fulfill the following condition for the case of non-overlapped collectives:

$$T_{coll} > \frac{DCP_ratio}{0.3} \times T_{comp}$$

and for the case of overlapped collectives:

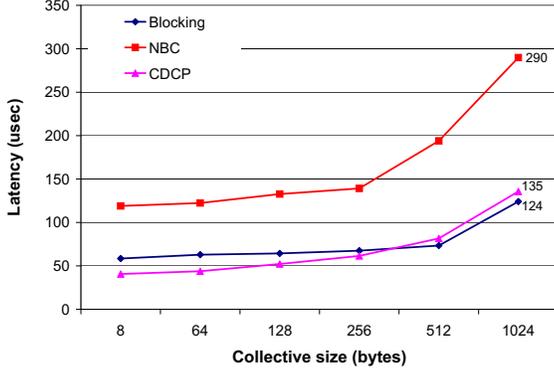


Fig. 11. Collective latency for the blocking, NBC, and CDCP schemes for various collective sizes (512-processor job).

$$T_{coll} > DCP_ratio \times T_{comp}$$

In essence, the collective time should be larger than a fraction of the computation time in order to improve the application performance. We believe that this condition is fulfilled for most applications running in strong-scaling mode because the collective costs often become the limiting factor in the performance specially at large processor counts.

F. Evaluation on a Scientific Application

The CDCP scheme was evaluated using the Parallel Ocean Program (POP). The run-time of POP is basically dominated by the barotropic phase, more specifically by the conjugate gradient solver (CG) that is used to solve a two-dimensional system of linear equations. Each iteration of this solver is composed on a series of communication and computation phases that are performed on the local sub-grid. At the end of an iteration there is a correction and convergence test phase that is performed every ten iterations. There are a total of five communication phases, three of which are collective operations (*Allreduce*) and two are a set of multiple point-to-point communication operations.

We applied the CDCP scheme to the three collective operations in order to overlap these operations with the available computation and communication phases. The first collective is overlapped with the computation of the solution correction in the correction and convergence test phase. Finally, the second and the third collectives are overlapped with the point-to-point communications performed in the main iteration as well as in the correction and convergence test phase, respectively. No more computation/communication can be overlapped for these collectives due to data dependencies. The timing data presented in this section corresponds to the average execution time of a set of ten iterations of the solver over 10,000 iterations.

Figure 12 shows the execution time of POP for the blocking, NBC, and CDCP schemes when scaling the number of pro-

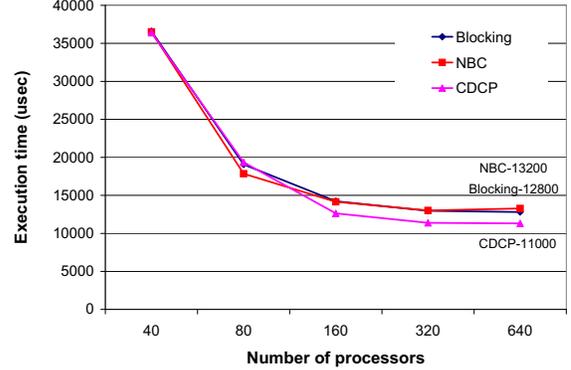


Fig. 12. Execution time for POP when scaling the number of processors.

cessors. A DCP ratio of 1:4 is used in the CDCP scheme. As can be seen, the performance of POP is significantly increased when using the CDCP scheme on processor counts of 160 and larger. In particular, on 640 processors the CDCP scheme achieves an improvement of 15% and 16% with respect to the blocking and NBC schemes respectively. This behavior is explained by the fact that the CDCP scheme reduces the application execution time when the total time of the collectives is higher than a certain fraction of the total computation time as described in Section V-E. This condition starts to occur at 160 processors for the POP application.

In addition, scaling the application to larger processor counts in the blocking and NBC schemes does not contribute to improved performance as can be also observed in Figure 12. This is because at large processor counts the increased communication costs offset the advantage of using further parallelism for the application. However, the CDCP scheme is able to provide increased performance as it can effectively overlap the collective with other communications. This result is very important since it clearly illustrates that although the CDCP scheme reduces the parallelism of the application it can effectively make better use of the computational resources in the system with respect to other schemes.

Figure 13 shows the execution time of POP for the CDCP scheme for various number of DCPs on 512 application processors. As can be seen, when increasing the number of DCPs the execution time asymptotically decreases to achieving the best performance on both 64 DCPs (DCP ratio of 1:8) and 128 DCPs (DCP ratio of 1:4). However, also note that starting when only 8 DCPs are used (DCP ratio 1:64), the performance improvement is still significant, achieving 95% the performance that it is achieved when using 64 DCPs. This result suggests that the amount of resources needed in the CDCP scheme can be small — representing only a 1.5% of the total application processors (512 processors).

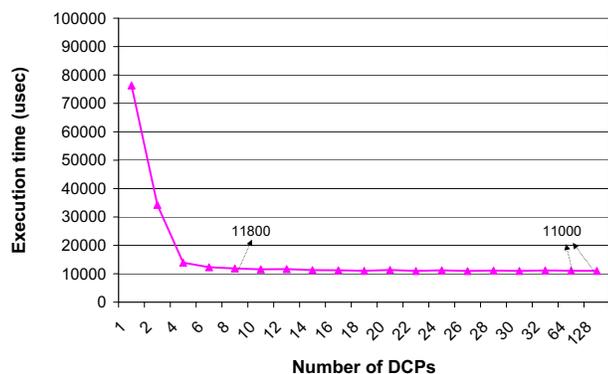


Fig. 13. Execution time for POP when scaling the number DCPs on 512 application processors.

VI. CONCLUSIONS

In this paper we have presented an approach that can significantly increase the performance of collective operations. The approach, that we term CDCP (*Configurable Dedicated Collective Processors*), uses a multiple program multiple data (MPMD) programming model to offload the collective to a number of available processors that are not used to run the application. The viability and efficiency of this scheme has been demonstrated for the *Allreduce* collective on a current large-scale cluster.

This atypical approach to calculate the collectives provides the combined advantage of reducing the collective latency while at the same time provides support for fully offloading its processing requirements. Results show that the collective overhead per processor is one order of magnitude lower ($1.6\ \mu\text{s}$) than over published methods, and also that the collective latency can be reduced by 30% with respect to the default implementation.

This combination results in a substantially improved performance of applications at large scale. Analysis of an important large-scale scientific application (POP) showed that the CDCP scheme significantly increases its performance by 15% on 640 processors without requiring additional processing resources or special purpose hardware. Thus, the CDCP scheme can be considered as a scalable and effective technique to further increase the performance of scientific applications at large-scale. In addition, the study of POP also revealed that one important source of unexploited parallelism resides in the parallelism between communication operations. On the other hand, the parallelism between communication and computation may be small and in some cases nonviable to exploit due to data dependencies. The typical SPMD model was inefficient or insufficient to exploit this type of parallelism, and a MPMD model like the CDCP scheme was crucial to harness this parallelism.

In future work, we plan to support other collective primitives in the CDCP scheme such as the *MPI_Alltoall*. This is limiting factor on the performance on some applications such as multidimensional Fast Fourier Transform algorithms. We also

plan to analyze the benefit of the CDCP scheme to tolerate process skew (or O/S jitter) which has been shown to be a critical factor that degrades collective performance at large processor counts in many systems.

REFERENCES

- [1] R. Rabenseifner, "Automatic MPI Counter Profiling," in *proceedings of the 42nd Cray User Group Conference, CUG SUMMIT 2000*, May 2000.
- [2] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, 1952.
- [3] N. R. Adiga and et al, "An Overview of the BlueGene/L Supercomputer," in *proceedings of the Supercomputing, also IBM research report RC22570 (W0209-033)*, Nov. 16–22, 2002.
- [4] R. Esser and R. Knecht, "Intel Paragon XP/S - Architecture and Software Environment," in *Supercomputer '93: Anwendungen, Architekturen, Trends, Seminar*. London, UK: Springer-Verlag, Jun. 1993.
- [5] Y. Huang and P. K. McKinley, "Efficient Collective Operations with ATM Network Interface Support," in *proceedings of the International Conference on Parallel Processing*, Aug. 1996.
- [6] A. Moody, J. Fernandez, F. Petrini, and D. K. Panda, "Scalable NIC-based Reduction on Large-scale Clusters," in *proceedings of the ACM/IEEE Conference on Supercomputing*, Nov. 2003.
- [7] D. Buntinas and D. Panda, "NIC-Based Reduction in Myrinet Clusters: Is It Beneficial," in *proceedings of the SAN-02 Workshop (in conjunction with HPCA)*, Feb. 2003.
- [8] R. Gupta, P. Balaji, D. K. Panda, and J. Nieplocha, "Efficient Collective Operations Using Remote Memory Operations on VIA-Based Clusters," in *proceedings of the International Symposium on Parallel and Distributed Processing*, Apr. 2003.
- [9] V. Tipparaju, J. Nieplocha, and D. Panda, "Fast Collective Operations Using Shared and Remote Memory Access Protocols on Clusters," in *proceedings of the International Symposium on Parallel and Distributed Processing*, Apr. 2003.
- [10] M. S. Wu, R. A. Kendall, and K. Wright, "Optimizing Collective Communications on SMP Clusters," in *proceedings of the International Conference on Parallel Processing*, Jun. 2005.
- [11] T. Hoefler, J. Squyres, W. Rehm, and A. Lumsdaine, "A Case for Non-Blocking Collective Operations," in *proceedings of the workshop Frontier on High Performance Computing and Networking in conjunction with ISPA'06*, Dec. 2006.
- [12] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference (Vol. 1) - 2nd Edition*. Cambridge, MA: MIT Press (c), 1998.
- [13] "MVAPICH: MPI over InfiniBand and iWARP," <http://mvapich.cse.ohio-state.edu/>.
- [14] J. S. Vetter and F. Mueller, "Communication Characteristics of Large-scale Scientific Applications for Contemporary Cluster Architectures," *Journal on Parallel Distributed Computers*, vol. 63, no. 9, pp. 853–865, 2003.
- [15] D. J. Kerbyson and P. W. Jones, "A Performance Model of the Parallel Ocean Program," *International Journal of High Performance Computing Applications*, vol. 35, no. 3, pp. 261–276, 2005.
- [16] J. C. Sancho, K. J. Barker, D. J. Kerbyson, and K. Davis, "Quantifying the Potential Benefit of Overlapping Communication and Computation in Large-Scale Scientific Applications," in *proceedings of the Supercomputing Conference*, Nov. 2006.