# Partitioned Reduction for Heterogeneous Environments

Alessio De Rango*, Gladys Utrera†, Marisa Gil†, Xavier Martorell†, Andrea Giordano‡, Donato D'Ambrosio§, Giuseppe Mendicino*
*Department of Environmental Engineering (DIAm) University of Calabria Rende (CS), 87036, Italy
alessio.derango@unical.it
†Computer Architecture Department Universitat Politècnica de Catalunya.BarcelonaTECH Barcelona, 08034, Spain
{gladys.utrera, marisa.gil, xavier.martorell}@upc.edu
‡Institute of High Performance Computing (ICAR) National Research Counsil (CNR) Rende (CS), Italy
andrea.giordano@icar.cnr.it
§Department of Mathematics and Computer Science (DEMACS) University of Calabria Rende (CS), 87036, Italy
donato.dambrosio@unical.it
*Department of Environmental Engineering (DIAm) University of Calabria Rende (CS), 87036, Italy
giuseppe.mendicino@unical.it

***Abstract*—Nowadays, performance in HPC applications focuses on MPI efficiency as the de facto message-passing library to exploit parallelism. Features such as multithread and communication and processing overlap are continuously studied to adapt to new platforms and a more significant number of processing units like GPU platforms. In this sense, recently, the MPI-4.0 standard introduced the partitioned point-to-point communication primitives to potentiate computation and communication overlapping. This paper introduces an innovative extension to MPI, specifically addressing partitioned communication for MPI-reduction primitives. Traditional reduction tasks conventionally involve processing the complete input vector following the conclusion of GPU computations. In contrast, our proposed methodology exploits message partitioning to process reduction tasks in real-time incrementally. This approach allows the system to process individual partitions of the input vector as they become available, removing the necessity to await the full completion of GPU computations before initiating the reduction. Our results demonstrate promising benefits, particularly for large message sizes. However, it is essential to acknowledge that optimizations at synchronization points remain potential bottlenecks, requiring meticulous analysis and consideration.**

***Index Terms*—MPI, partitioned communication, GPU programming, distributed computing**

## I. Introduction

Achieving high performance in a High-Performance Computing (HPC) cluster means maximizing the parallelism obtained from the high number of processing elements. For this, the message-passing paradigm is essential as the core foundation for distribution and, until now, is achieved with the Message Passing Interface (MPI) library - often combined with other shared-memory, thread-based programming models. A mixture of programming models and heterogeneous processing elements is the usual ecosystem in which HPC applications are developed and run. The way the MPI library exploits parallelism and its efficiency in using communication and processing resources are critical to obtaining the expected performance of HPC applications. These applications currently solve very varied problems (from human behaviour simulations to weather forecasting and structure design), as are the platforms on which they run: varied in terms of the heterogeneity of their components and the ever-increasing number of processing components. One of the most studied factors is the overlap between communication and processing to mask the time dedicated to communication as much as possible and thus minimise it. For example, large messages require the corresponding reservation of memory space or, in case they are split into smaller ones, synchronisation among them, waiting for barriers to ensure the arrival of all of them before continuing. One of the critical factors in the performance of applications that work with parallelism with message passing is the time spent in synchronization at a barrier, waiting to complete all the expected work. This barrier can be waiting for all the processes involved in a job to finish or for a long message divided into parts to arrive complete. To deal with similar situations, the MPI-4.0 standard includes point-to-point primitives, implementing partitioned communication to break the waiting barriers [1]. This way, partitioned messages are pipelined and can be processed as soon as they become available.

Our foundation is our latest research efforts, specifically in evaluating diverse algorithm reduction proposals [2]. We considered various reduction methodologies, aiming to enhance efficiency and scalability. One standout approach [3], rooted in message partitioning, has inspired a novel methodology that we believe holds immense potential for the real-time processing of reduction tasks. This methodology is particularly tailored for scenarios where input vectors are generated through GPU computations and has recently become of special interest [4].

In this context, we propose a reduction algorithm that leverages message partitioning to incrementally process reduction tasks in real-time. This innovative approach enables the system to handle individual partitions of the input vector

 https://dx.doi.org/10.1109/PDP62718.2024.00047

as they become available, eliminating the need to await the full completion of GPU computations before initiating the reduction process.

Our experimental results highlight promising advantages, particularly in scenarios involving large message sizes. However, it is crucial to recognize that optimizations at synchronization points pose potential bottlenecks, demanding thorough analysis and consideration. As we delve into the intricacies of our proposed methodology, we address these challenges to comprehensively evaluate and refine its performance, particularly in the context of GPU-accelerated high-performance computing applications.

The contributions of this work are:

- Design and implementation of a partitioned reduction algorithm when working with GPUS.
- Evaluation of the proof of concept varying the message size and number of nodes, on a GPU cluster using OpenMPI library. Comparison of our approach against the native reduction algorithm and another from literature.

The rest of the document is organized as follows: we present first, to situate the reader, a description of works related to our proposal. Next, there is a description of our proposed algorithm for the reduction collective. After that, we discuss the methodology followed to evaluate the proposal and the execution framework used. Then, we show the experimental results and analyze the outputs obtained. Finally, there are the conclusions and future work.

## II. Related work

Since the introduction of partitioned communication in the standard MPI 4.0 [5], several proposals appeared to apply it to implement point-to-point and collective communication primitives. The main objective is to achieve the maximum of computation communication overlapping. The proposals are generally built on the combination of the primitives that prepare and manage the partitioned messages, making some adaptation and/or generalization [6].

Recently, in the work by Gillis et al. [4], they propose primitives to implement the partitioned communication on top of the MPICH library. They investigate the benefits of pipelined communication in the context of a hybrid approximation MPI+threads. They conclude that large messages are more suitable for this approximation.

However, the idea of working with partitions of messages has already been presented in several works previously [3], [7], [8], trying to exploit the increasing number of cores in multicores clusters at the intra-node level. In particular, in the work by Traff et al., they propose to work with the partitions concurrently. They obtain performance benefits over native implementations because of the message size reduction, i.e. partitions and parallel communication.

To our knowledge, this is the first work, even as a proof of concept, to design and propose an MPI collective algorithm based on the idea of partitioned communication extended to GPUs. In fact, our proposal is not based on pipelined communication but on message partitioning and concurrent communication.

```
/* computation offload to GPU
   on data with n elements */
copy_from_host_to_device (data, n);
computation ();
/* wait for results back from GPU */
gpu_synchronization ();
copy_from_device_to_host (data, n);
/* reduction */
sync_allreduce (data, n);
```

Fig. 1. Code excerpt that shows the native approach

## III. Proposal

This section is devoted to describing our proposed MPI reduction mechanism.

Figure 4 describes the native approach, i.e. using the native MPI library implementation, and figures 5 show the mechanism followed by our proposal. The figures depict the execution in several steps of a code with a pattern like the one shown in figure 1 for the traditional approach.

In the *native* approach, we can see that in *step 1*, the data is copied from the host to the device to offload the computation. Then *step 2* represents the waiting time until the computation finishes and the data is copied back from the device to the host. Finally, *step 3* and *step 4* show that once the data is in the host, a global blocking reduction is performed among all the tasks involved (*MPI_Allreduce*).

A pseudocode of the native mechanism which we use as a reference is shown in figure 2.

Our proposal consists of a set of primitives listed in 2 and used in the pseudocode from figure 3. The set of primitives prepares the mechanism in the *_init* function, where a callback function is installed, and an MPI group is defined. The MPI group defines the set of processes that will carry out the reduction of each k-partition following the `Multilane` approach [3]. The callback function is used to wait for each k-partition to finish computation from the GPU. Then, the *_do_partitioned_reduction* performs each partition's asynchronous intra-node and inter-node reduction. Finally, the message is rebuilt from the processes of the intra-node group (MPI_Allgather).

## IV. Methodology

This section is devoted to describing the set of experiments carried out to evaluate our proposal and compare it with other approaches.

As a proof of concept work, we build a set of experiments with simple code patterns to compare the different mechanisms based on the codes shown in figures 1 and 3. We perform a loop with this pattern and measure the total execution time to obtain an average time. The input vector is shuffled at each iteration to avoid distortion due to cache reuse.

```
/* initialization function to prepare
   the partitioned reduction */
_init (n){
  install_callback_function (_CB_function);
  create_mpi_intra_socket_group(my_local_rank,
                                size);
  size_k = n / size;
}

/* callback function invoked whenever a
   partition is available from GPU */
_CB_function () {
  copy_from_device_to_host (data+k*size_k, k);
  /* intra-node ireduce by rank k */
  async_reduction (data+k*size_k, size_k, k);
}

/* the partitioned reduction */
_do_partitioned_reduction (data, n) {
    p=0;
    while (p < size) {
       /* wait for ireduction to complete */
       waitany_async_reduction (k);
       if (k = my_local_rank) {
           /* inter-node allreduce */
           async_allreduction (data+k*size_k,
                               size_k);
       }
       p++;
    }
    /* complete iallreduce */
    wait_async_allreduction ();
    gpu_synchronize();
    rebuild_data_result (data, n);
}
```

Fig. 2. Code excerpt that shows the set of primitives that implement our approach

```
/* prepare partitioned reduction */
_init(n);
copy_from_host_to_device (data, n);
/* offload computation */
computation ();
/* perform partitioned reduction */
_do_partitioned_reduction(data, n);
```

Fig. 3. Code excerpt that uses the primitives from our proposal

The computation phase consists of floating point operations performed on the input vector and generating a new output vector.

The executions were performed on a cluster based on IBM Power9 processors, with a Linux Operating System and an Infiniband interconnection network. Each compute node has 2 x IBM Power9 8335-GTH @ 2.4GHz (3.0GHz on turbo, 20 cores and 4 threads/core, totalizing 160 threads per node; 512GB of main memory and 4 x GPU NVIDIA V100 (Volta) with 16GB HBM2. For the MPI library, we use OpenMPI version 3.0.0.

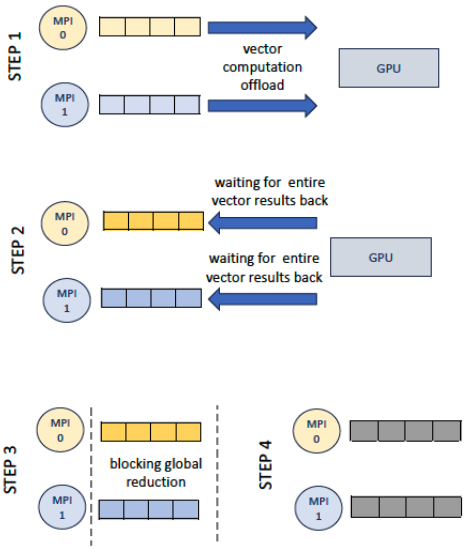


Fig. 4. GPU calculation + Reduction: native approach (Native algorithm or Multilane)

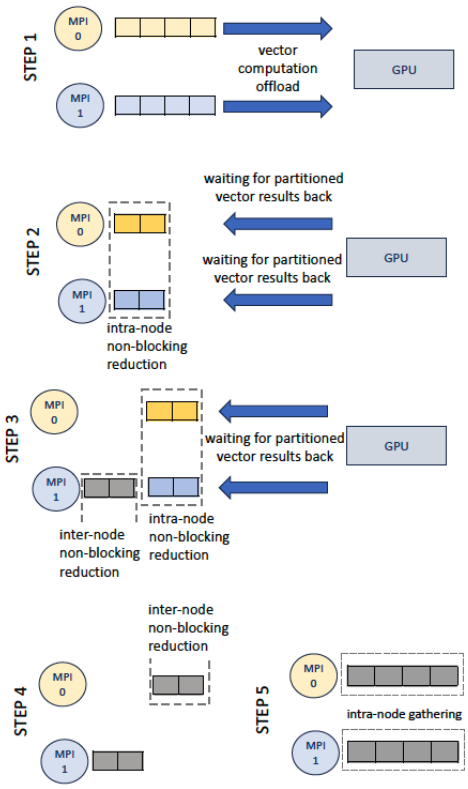


Fig. 5. GPU calculation + Reduction: our approach

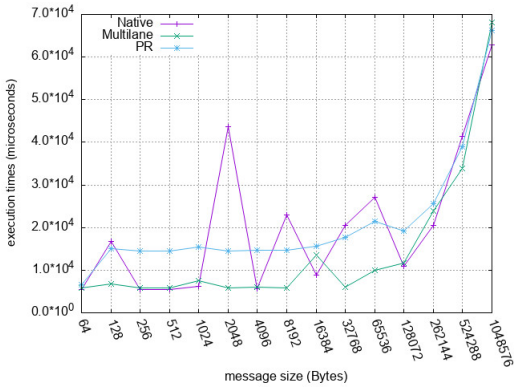


Fig. 6. Execution times on 4 nodes with 40 processes per node for message sizes between 32Bytes and 1 MByte

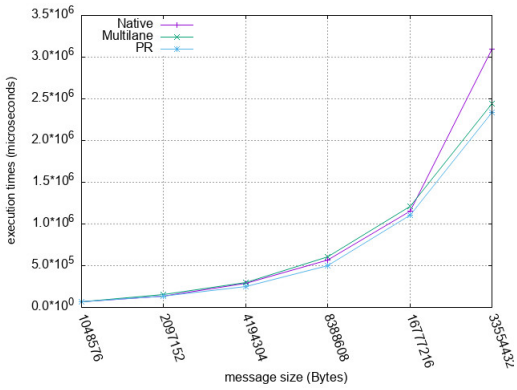


Fig. 7. Execution times on 4 nodes with 40 processes per node for message sizes between 1MByte and 33MBytes

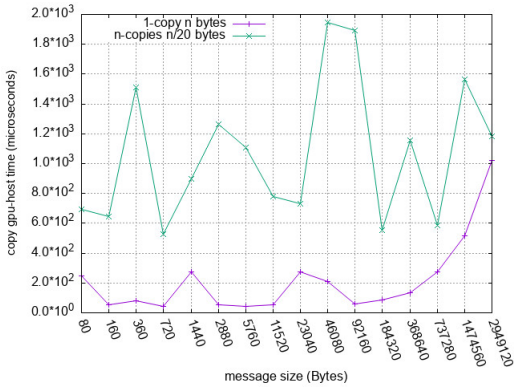


Fig. 8. Execution time comparison for copy from device to host of data between 40 Bytes and 1 MByte

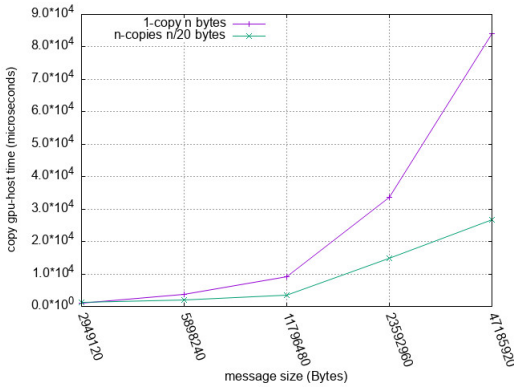


Fig. 9. Execution time comparison for copy from device to host of data between 1 MByte and 47 MBytes

## V. Experimental results

In order to test scalability we test on 1, 2 and 4 nodes, configuring 40 processes per node and with MPI intra-node sockets of 20 processes. We don't use hyperthreading to not distort the results.

For the native approach, we use the reduction algorithm from the current library in use and the Multilane algorithm.

Our results show that no matter the number of processes, that below 1 MB the Multilane approach is the one that obtains better performance. Moreover, we observe that the algorithm from the OpenMPI library is very unstable, generating very different values for the execution times between various repetitions of the same experiment, as shown in figure 6.

It seems that our approach needs to perform several copy-back operations (as many as partitions), which are more costly than one single large copy. Due to the completeness of the partition computation, the overlapping of computation and communication benefit does not compensate till the message is large enough and depends on the amount of computation.

For message sizes over 1 MB, we obtain an improvement of up to 21% over Multilane and several times over the native approach.

## VI. Conclusion

In this work, we present a proof of concept of a mechanism, named PR (Partitioned Reduction) that performs reduction by message partitioning in a scenario where the input vector comes from GPUs. The main objective is to start the reduction as soon as any partition of the input vector is available, instead of waiting for the whole vector to finish the offloaded calculation. The reduction algorithm is based on the idea of hierarchical reduction, where the processes within an intra-socket group, are in charge of intra and inter-node reduction of just a partition of their message. Ultimately, the message is rebuilt from the final partition results at the intra-node level.

We obtained performance improvements for message sizes over 1 MBytes. It seems that for smaller message sizes, the penalty due to the repeated copies from the device to the host of the partitions exceeds the benefit obtained because of overlapping computation and communication.

## Acknowledgment

This work was supported by the Spanish Ministry of Education (PID2019-107255GB-C22) and the Generalitat de Catalunya (2021-SGR-01007). This work was partially supported by PON "Research and Innovation" 2014-2020 CUP H25F21001220006, C. I. 1062_R22_INNOVAZIONE

## REFERENCES


[1] M. G. Dosanjh, A. Worley, D. Schafer, P. Soundararajan, S. Ghafoor, A. Skjellum, P. V. Bangalore, and R. E. Grant, "Implementation and evaluation of mpi 4.0 partitioned communication libraries," *Parallel Computing*, vol. 108, p. 102827, 2021.

[2] G. Utrera, M. Gil, and X. Martorell, "Analyzing the performance of hierarchical collective algorithms on arm-based multicore clusters," in *2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2022, pp. 230–233.

[3] J. L. Träff and S. Hunold, "Decomposing mpi collectives for exploiting multi-lane communication," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, 2020, pp. 270–280.

[4] T. Gillis, K. Raffenetti, H. Zhou, Y. Guo, and R. Thakur, "Quantifying the performance benefits of partitioned communication in mpi," in *Proceedings of the 52nd International Conference on Parallel Processing*, ser. ICPP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 285–294. [Online]. Available: https://doi.org/10.1145/3605573.3605599

[5] MPI-Forum, "Mpi: A message-passing interface standard 2020 draft specification. techincal report," University of Tennesse, Knoxville, TN, USA, Tech. Rep., 2020.

[6] D. J. Holmes, A. Skjellum, J. Jaeger, R. E. Grant, P. V. Bangalore, M. G. Dosanjh, A. Bienz, and D. Schafer, "Partitioned collective communication," in *2021 Workshop on Exascale MPI (ExaMPI)*, 2021, pp. 9–17.

[7] M. Bayatpour, S. Chakraborty, H. Subramoni, X. Lu, and D. K. D. Panda, "Scalable reduction collectives with data partitioning-based multi-leader design," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126954

[8] M. Bayatpour, J. Maqbool Hashmi, S. Chakraborty, H. Subramoni, P. Kousha, and D. K. Panda, "Salar: Scalable and adaptive designs for large message reduction collectives," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018, pp. 12–23.