An MPI interface for application and hardware aware Cartesian topology optimization

Christoph Niethammer
High-Performance Computing Center Stuttgart
Stuttgart, Germany
niethammer@hlrs.de

Rolf Rabenseifner
High-Performance Computing Center Stuttgart
Stuttgart, Germany
rabenseifner@hlrs.de

ABSTRACT

Many scientific applications perform computations on a Cartesian grid. The common approach for the parallelization of these applications with MPI is domain decomposition. To help developers with the mapping of MPI processes to subdomains, the MPI standard provides the concept of process topologies. However, the current interface causes problems and requires too much care in its usage: MPI_Dims_create does not take into account the application topology and most implementations of MPI_Cart_create do not consider the underlying network topology and node architecture. To overcome these shortcomings, we defined a new interface that includes application-aware weights to address the communication needs of grid-based applications. The new interface provides a hardware-aware factorization of the processes together with an optimized process mapping onto the underlying hardware resources. The paper describes the underlying implementation, which uses a new multi-level factorization and decomposition approach minimizing slow inter-node communication. Benchmark results show the significant performance gains on multi node NUMA systems.

CCS CONCEPTS

• **Computing methodologies** → **Parallel programming languages**.

KEYWORDS

MPI, Cartesian communication, grid, mapping

ACM Reference Format:
Christoph Niethammer and Rolf Rabenseifner. 2019. An MPI interface for application and hardware aware Cartesian topology optimization. In *26th European MPI Users' Group Meeting (EuroMPI 2019), September 11–13, 2019, Zürich, Switzerland*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3343211.3343217

1 INTRODUCTION

Many scientific applications are implemented through computations on a Cartesian grid. The common approach for the parallelization of these applications with MPI is domain decomposition. To help developers with the mapping of MPI processes to subdomains, the MPI standard provides the concept of process topologies.

Two useful functions for Cartesian-type topologies are MPI_Dims_create and MPI_Cart_create. While the first function helps finding a factorization for a Cartesian process grid from a given number of processes, the second function creates an MPI Cartesian communicator from a given Cartesian process grid. However, this interface requires care in its usage as neither MPI_Dims_create takes into account the application topology nor MPI_Cart_create takes care of the underlying network topology and node architecture of the system. This results into a problem for today's multi node NUMA systems because of the limited communication bandwidth at the different hardware levels, namely inter-node, inter-socket and inter-core communication.

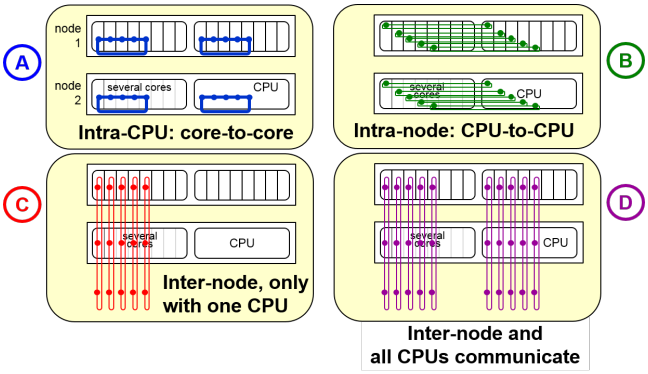
Figure 1 shows different communication schemes (a) and the corresponding duplex accumulated ring bandwidth per node (b) for varying number of processes per node and process placements. As expected, the limit of accumulated intra-CPU and intra-node bandwidth (green and blue) is 8 times larger than the limit of accumulated node-to-node bandwidth (red and purple). To achieve good performance it is therefore essential to make a better (or the best) usage of the available bandwidth at the different levels: Inter-node communication must be reduced in favor of intra-node communication.

This paper addresses the optimization of communication patterns that are analogous to the halo communication in a Cartesian domain decomposition of a data mesh for a given application, as shown in Figure 2. For given number of processes N on given multi-level hierarchical hardware resources, number of dimensions, and data mesh and halo sizes of the application, this paper describes how to achieve an optimized factorization of N into the dimensions d_i of a Cartesian virtual process grid, and a mapping of the application processes on the hardware resources in such a way that the total communication time is minimized. Moreover, this paper describes how an MPI library interface can be designed for this purpose and how it can be implemented.

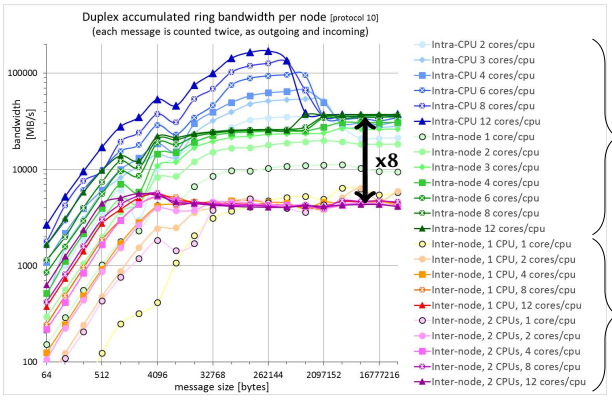
The paper is organized as follows: Section 2 will discuss related work. The background of the existing problem is outlined in Section 2.1. Section 3 proposes a multi-level mapping strategy to provide an application- and hardware-aware, optimized Cartesian process topology. Sections 4 and 5 propose how this new strategy can be included into the MPI standard and the MPI libraries, and show details about our implementation of this process mapping. Sections 6 and 7 present an example and benchmark results, including hints for some further developments. Section 8 concludes this paper.

EuroMPI 2019, September 11–13, 2019, Zürich, Switzerland

ACM ISBN 978-1-4503-7175-9/19/09…$15.00
https://doi.org/10.1145/3343211.3343217



(a) Communication schemes



(b) Duplex accumulated bandwidth per node

Figure 1: Duplex accumulated bandwidth benchmark: Messages are sent bidirectionally in rings.

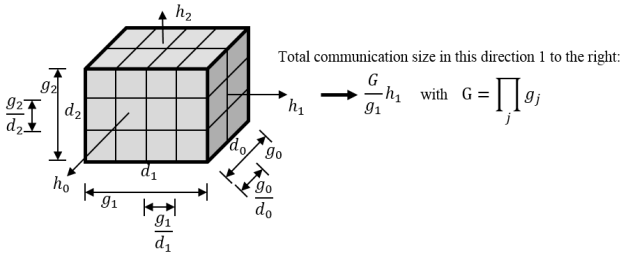


Figure 2: The global application data mesh with $G = \prod_i g_i$ elements, and its domain decomposition on $N = \prod_i d_i$ processes. The halo width is h_i.

2 STATE OF THE ART

In order to specify the communication characteristics of an application, the current MPI 3.1 standard [1] includes the MPI topology mechanism. Therein, the MPI interface provides a convenient way of process naming. The relation between communicating processes builds topological patterns, which are referred to as virtual topology. These topologies can be described with the provided naming mechanism.

Currently, the MPI standard supports three different virtual topologies: Cartesian, graph, and distributed graph. For each topology a specialized interface is provided. The Cartesian topology interface allows to create corresponding communicators based on the desired process grid dimensions, which implicitly define the virtual topology. In contrast, the two graph topology interfaces are based on the explicit user's definition of the virtual topology graph to create a corresponding communicator.

While the graph topology interfaces additionally allow to specify weights reflecting the amount or frequency of communication between processes, the Cartesian topology lacks this feature, always assuming equal weights. This clearly restricts the current optimization opportunities to choose an optimal factorization of the given number of processes into the sizes of the dimensions of the Cartesian process topology (for a given number of dimensions), but also to reorder the process ranks to minimize the communication time along the edges in the virtual Cartesian grid.

The difference between the two graph interfaces lies in their scalability: In the graph interface, the entire topology information is held by each process, while in the distributed graph interface, each process holds only the topology information related to the direct neighbors in the graph. Therefore, the graph interface has the disadvantage of lacking scalability due to a quadratically increasing memory footprint.

Beside the virtual topology, we must consider the topology of the underlying physical hardware. At the moment, this topology information is not exposed to the user via the MPI interface. However, the MPI standard states that the virtual topology information can assist an MPI runtime in the efficient mapping of processes onto the hardware. A variety of approaches to solve such a problem of embedding the virtual topology graph into the hardware topology graph were developed in the past. The embedding problem is often also combined with the load balancing of the computations. A variety of partitioning libraries exist to solve this task, such as METIS, ParMETIS [12], SCOTCH [16], PT-SCOTCH [7], or others [10]. Other approaches try to solve this problem using information collected from application runs to optimize the placement of the processes at launch time, e.g., via the mpich rank reorder method provided on Cray systems [2].

While the embedding problem for general graphs is NP-hard, polynomial solutions exist for relevant, special cases. One of those special cases includes Cartesian topologies. One approach is therefore to handle Cartesian topologies as graphs and to apply similar heuristics as for graph topologies [14]. Of course, such an approach does not take into account the opportunities of optimizing the Cartesian factorization. Research has also been done for special optimization of collective operations on hierarchical networks, e.g., [14, 18]. Note that an MPI library can choose a different rank-remapping for each collective operation, i.e., independent of the mapping used for Cartesian neighbor communication, as, e.g., proposed in this paper. The importance of topology-aware optimizations on the way to exascale was also shown before in [6]. Depending on the cluster network itself, the re-ranking may also be optimized for the hardware grid topology of the cluster network [4, 5, 19].

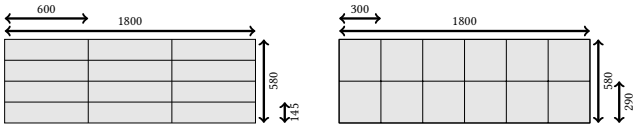


Figure 3: Decomposition of a data mesh with 580×1800 grid points onto 12 processes. Left: Suggestion from MPI_Dims_create. Right: Optimal distribution.

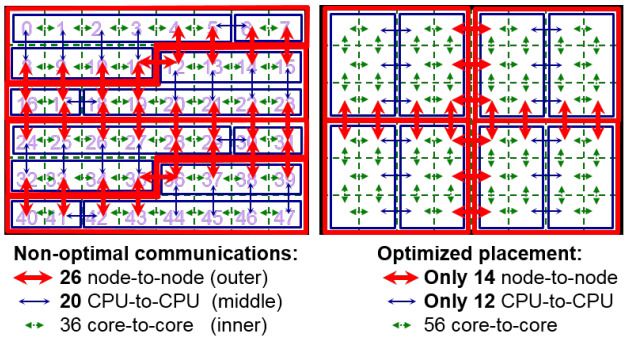


Figure 4: Process mapping to nodes, sockets and cores. Left: Non-optimal default mapping behavior of MPI_Cart_create. Right: Optimized placement suggested by the multi-level decomposition in this work.

2.1 Background

At this point, there are three problems with the MPI API: First, MPI_Dims_create computes only a process grid with dimensions as close to each other as possible, e.g., based on the algorithm in [20]. If the underlying data mesh of a simulation is not close to a quadratic or cubic shape, the decomposition becomes non-optimal [3]. Figure 3 shows an example of a 580×1800 mesh, which shall be distributed across 12 processes. MPI_Dims_create will suggest a 4×3 process grid for the virtual topology of the 12 processes. However, assuming that all 12 processes are connected with links that have the same performance, such a distribution is not optimal. In this case, a 2×6 decomposition would be better, as it would lead to almost quadratic subdomains and therefore less communication at the subdomain boundaries.

The second problem arises from calculating the factorization of the number of MPI processes independently from the knowledge of the underlying hardware.

The third problem arises from not optimized implementations of MPI_Cart_create, which map the processes sequentially to the processor grid as shown on the left side of Figure 4. This leads to an unnecessary high amount of slow inter-node communication. To compare, an optimized mapping, which takes into account the network topology and the system architecture, is shown on the right. This mapping problem is n general non-trivial, as shown in [11].

All three problems are closely coupled and therefore have to be solved together.

3 MAPPING STRATEGY

Hereinafter, the application topology is given as a d-dimensional Cartesian mesh with a total of $G = \prod_{i=0}^{d-1} g_i$ elements, where g_i are the data mesh dimensions in directions $i \in [0, d-1]$. The halo width in direction i is h_i, see Figure 2. The target system shall consist of N nodes with P cores per node.

To achieve optimal domain decomposition and MPI process mapping for a given hardware topology, we present in the following a multi-level optimization approach. The benchmark in Figure 1 shows that the node-to-node communication is dominant in the whole communication overhead. Therefore, this communication must be minimized first, i.e., our approach starts at the node level and ends at the core level.

On each hardware level, from the coarsest to the finest, i.e., from the slowest communication path to the fastest, the factorization of the given amount of nodes into the d dimensions is calculated in such a way that the surface per node in each direction is as evenly balanced as possible. This approach is different from other approaches based solely on graph mapping, because we influence the factorization and therefore also the domain decomposition complying with the given hardware constraints.

3.1 Grid decomposition at the node level

The application shall be run on N nodes. The domain decomposition nodes form a d-dimensional Cartesian node topology with n_i nodes in the i-th dimension. It holds

$$N = \prod_{i=0}^{d-1} n_i \ . \tag{1}$$

The communication within the application requires for each node the exchange of halo data with its neighbors. The amount of data to be transferred depends on the subdomain surface determined by its dimensions, so that the communication costs c can be described as

$$c = 2 \sum_{i=0}^{d-1} c_i h_i \prod_{\substack{j=0 \\ j \neq i}}^{d-1} \frac{g_j}{n_j} = 2 \frac{G}{N} \sum_{i=0}^{d-1} \frac{c_i h_i}{g_i} n_i \ . \tag{2}$$

The factors c_i are additional cost factors (e.g., representing different communication costs due to contiguous or strided data in different directions).

The present goal is to reduce the inter-node communication costs. This is achieved by finding a factorization $(n_i)_{i=0..d-1}$, which minimizes the sum in (2) under the condition (1).

3.2 Grid decomposition at the core level

After achieving an optimized node mapping, the same approach is applied at the next hardware topology level for the subdomains. The submesh at the new level has mesh dimensions $g'_i = g_i / n_i$. Assuming the core level as the next level, we construct a Cartesian core topology with dimensions p_i. The communication costs c' at this level are now given by

$$c' = 2 \frac{G'}{P} \sum_{i=0}^{d-1} \frac{c_i h_i}{g'_i} p_i = 2 \frac{G}{NP} \sum_{i=0}^{d-1} \frac{c_i h_i}{g_i} n_i p_i \text{ with } P = \prod_{i=0}^{d-1} p_i \tag{3}$$

Again, we search a factorization $(p_i)_{i=0..d-1}$ that minimizes the term in (3).

3.3 Multi-level decomposition

The approach shown so far for the node and core level can be applied hierarchically to any number of hardware topology levels by subsequently minimizing the communication costs

$$c^{(l)} = 2\frac{G}{\prod_{k=0}^{l} N^{(k)}} \sum_{i=0}^{d-1} \frac{c_i h_i}{g_i} \prod_{k=0}^{l} n_i^{(k)} \tag{4}$$

$$\text{with } N^{(l)} = \prod_{i=0}^{d-1} n_i^{(l)} \,, \tag{5}$$

starting from hardware topology level $l = 0$. Input requirements for this algorithm are the number of dimensions d and the number $N^{(l)}$ of nodes on each hardware level l of L levels, e.g., the number of ccNUMA nodes, the number of NUMA domains within each ccNUMA node, and the number of cores per NUMA domain. The product $\prod_{l=0}^{L-1} N^{(l)}$ must be equal to the total number of MPI processes N. The resulting dimensions $(d_i)_{i=0..d-1}$ of the Cartesian process grid is defined by the product of the $n_i^{(l)}$ over all hardware levels: $d_i = \prod_{l=0}^{L-1} n_i^{(l)}$.

Based on this decomposition, a rank reordering can be finally performed to create a new optimized MPI communicator. A result is shown in Figure 4 with 2 dimensions and three levels, and $(N^{(l)}) = (4, 2, 6)$.

A first implementation[1] showed execution times for the factorization in the order of Jesper Träff's algorithm [20].

4 EXTENSIONS TO THE EXISTING MPI INTERFACE

The current MPI interface consists mainly of the two routines: MPI_Dims_create, with the number of processes N and dimensions d as input, and the factorization $(n_i)_{i=0..d-1}$ as output, and MPI_Cart_create, with a communicator consisting of a group of processes on the given hardware resources, an appropriate factorization, and a reorder flag as input, and a communicator reflecting the Cartesian topology with the goal of an optimized reordering of the processes if reorder=true as output. The major problem is that the input of MPI_Dims_create does neither include any hardware information nor any information about the ratio of the data mesh sizes of the application, $(g_i)_{i=0..d-1}$.

Consequently, the proposed interface is a combination of both routines together with an additional input argument $(w_i)_{i=0..d-1}$ representing the communication needs of the application's data mesh. As seen above, optimization on each hardware level requires the minimization of the sum $G\sum_{i=0}^{d-1} \frac{c_i h_i}{g_i} n_i'$ with $n_i' = \prod_{k=0}^{l} n_i^{(k)}$. Consequently, we define the weights w_i as

$$w_i = G\frac{c_i h_i}{g_i}. \tag{6}$$

[1] Our poster from 2018 [15] and a free implementation from Feb. 2019 in a form suitable for MPI libraries are available at fs.hlrs.de/projects/par/mpi/EuroMPI2018-Cartesian and latest results from this paper on fs.hlrs.de/projects/par/mpi/EuroMPI2019-Cartesian

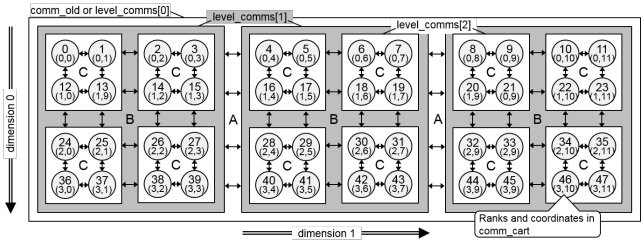


Figure 5: Example with a comm_old with 48 processes, $d = 2$ dimensions, $(w_i) = (1./4, 1./12)$, and 2 split types, e.g., splitting into 3 ccNUMA nodes, each consisting of 4 CPUs, with each containing 4 cores. The level_comms[i] show this splitting of comm_old into hierarchical subcommunicators. The renumbering in comm_cart should first minimize the communication A (here, the communication between the ccNUMA nodes), then B (CPU to CPU communication), and last C (core to core communication)

With this definition, the weights reflect the total communication cost in one direction, i.e., before the data mesh is divided into subdomains as shown in Figure 2, i.e., the total communication size $G\frac{h_i}{g_i}$ multiplied this additional cost factor c_i.

This interface reflects the hardware resources through the communicator input argument and the application communication needs through the weights input argument. In our proposal for MPI-4.0 and in our implementation, we pass to the new routine the weights as an array of double precision numbers. For the case of equal weights, we reserve the constant MPI_WEIGHTS_EQUAL to be passed as an alternative.

This interface is in principle independent from details regarding the hardware resources. On the other hand, the proposed algorithm above requires that the hardware is symmetric in structure, i.e., that on each level l, the number of units $N^{(l)}$ of the underlying level is always identical, such that the communicator with N processes can be characterized through $N = \prod_{l=0}^{L-1} N^{(l)}$.

4.1 Further interfaces

Further interfaces and info arguments can be defined to directly describe which hardware levels should be taken into account for the multi-level factorization of N. This case can be defined through split-types or split info arguments valid for MPI_Comm_split_type, or by directly providing a hierarchical set of communicators as shown in Figure 5.

A weighted factorization MPI_Dims_create_weighted should also be provided as an MPI interface. Compared to MPI_Dims_create, the argument list should additionally include the weights array and an info argument.

This routine should find a factorization $(n_i)_{i=0..d-1}$ of N, which minimizes the sum $\sum_{i=0}^{d-1} w_i n_i$.

5 IMPLEMENTATION

5.1 MPI_Dims_create_weighted

One of the key functions is MPI_Dims_create_weighted. It can be implemented based on the brute-force algorithm developed by Jesper Träff and Felix Lübbe [20]. Important is that this algorithm does not traverse the whole space of possible factorizations, but only traverses all factorizations with not strictly decreasing factors. As selection criterion it uses only the difference between first and last factor, $d_0 - d_{d-1}$, which is minimized.

For weighted factorization, one has to modify the algorithm as follows.

First, the weights have to be sorted from the smallest to the largest value, then the factorization must be realized with the sum-criterion, and finally the sorting must be reverted such that the d_i fall in the correct sequence.

Second, as there can exist multiple solutions at this point, an additional criteria is used, which targets another aspect of optimization. If the sum criterion allows two different factorizations with the same minimal sum, then the factorization with minimal difference between largest and smallest factor shall be chosen. This second criterion was the only one used in [20]. In this fashion more "cubic shape"-like factorizations are favored. If there are still two solutions with the same sum and the same difference, then one may choose the factorization with the smallest largest factor. For example, with equal data mesh sizes g_i, 360 nodes can be factorized into 3 dimensions as 360=10x6x6 and 360=9x8x5, both having equal sum $\sum n_i = 22$ and difference $n_2 - n_0 = 4$. The first and second criterion would allow both results, but the third would choose 9x8x5. For $N \leq$ 10,000,000 and d between 2 and 10, there exist 4,630 factorizations with identical sum and difference, e.g., with $N \leq 10{,}000$ and $d = 3$, 22x14x12=21x16x11, 21x16x15=20x18x14, and 26x16x15=24x20x13, or for $d = 4$, 10xFx6x6=9x8xFx5 with F=6, 7, 8, or 9. Furthermore, there are 6066 results in which the difference-criterion alone would have implied a factorization that has a larger (and therefore worse) sum, but only two cases for N below 50,000, for example N=35200 = 40x40x22 (with $\sum$=102, Δ=18) = 44x32x25 (with $\sum$=101, Δ=19). Although these examples are rare cases, it is recommended to use all three criteria.

5.2 MPI_Cart_create_weighted

This routine has to choose which hardware levels are used. This may be based on information detected at the installation of the MPI library. Then, the comm_old can be accordingly split. The next step is to analyze whether on each level each subcommunicator has the same size. If this criterion is fulfilled, then also on each level each communicator has the same number of subcommunicators.

If the hardware has the described symmetric structure, then condition (4) under the constraint (5) can be fulfilled by L times calling MPI_Dims_create_weighted. The result is the matrix of factors $(n_i^{(l)})_{i=0..d-1}^{l=0..L-1}$, i.e., the multi-level multi-dimensional factorization of the number of MPI processes N.

Each process calculates its rank on each hardware level. On each level l each process can calculate its coordinates $(c_i^{(l)})_{i=0..d-1}$ based on its rank and $(n_i^{(l)})_{i=0..d-1}$. Then, on each dimension i, each process can calculate its final coordinate c_i based on its multi-level coordinates $(c_i^{(l)})^{l=0..L-1}$, and the number of processes in the given dimension for each level $(n_i^{(l)})^{l=0..L-1}$.

Based on these final coordinates and the implied dimensions $d_i = \prod_{l=0}^{L-1} n_i^{(l)}$, each process can calculate its new_rank according to the algorithm of MPI_Cart_rank. The reordering can be implemented through MPI_Comm_split with color=0 and key=new_rank, as described in [1] in Section 7.5.8 as an advice to implementors for MPI_Cart_map.

If the application chooses to predefine some of the dimensions with a fixed value, then still a weighted factorization (with only one level) can be used together with the algorithm of William Gropp [8, 9].

If the hardware cannot be described with symmetric levels, then again MPI_Dims_create_weighted can be used with only one level, and the existing MPI_Cart_create can be applied. The latter can be optimized for example with the methods described in [17] or by implementing MPI_Cart_create through MPI_Dist_graph_create, and using an appropriate rank reordering as presented in [13].

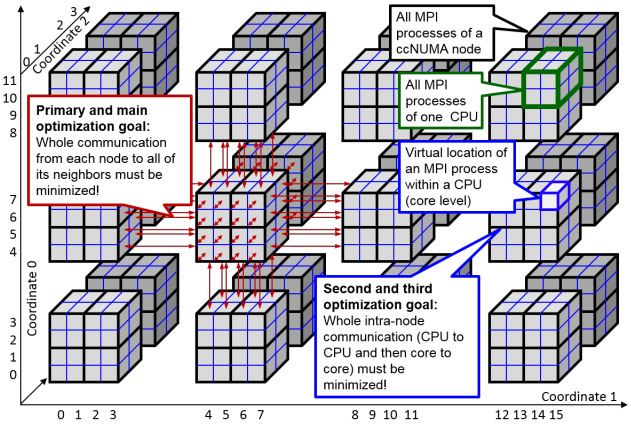


Figure 6: Example with 24 nodes (level 0), 4 CPUs per node (level 1) and 8 cores per CPU (level 2). The figure shows the 3-dimensional distribution calculated by MPI_Cart_create_weighted **based on weight values (1/12, 1/16, 1/8).**

6 EXAMPLE: MULTI-LEVEL VERSUS SINGLE-LEVEL

In this example the number of processes on each hardware level are $N^{(l)}$ = (24, 4, 8), for example expressing 24 nodes with each 4 CPUs per node and 8 cores per CPU. The application weights are $w_i = (\frac{1}{12}, \frac{1}{16}, \frac{1}{8})$, for example expressing a total data mesh of 1200 x 1600 x 800 elements. The following table and Figure 6 show the results of the proposed MPI_Cart_create_weighted, i.e., the results $n_i^{(l)}$ of minimizing (4) from level 0 until level 2 with the weights defined in (6).

		i=0	i=1	i=2				
		w_i=						
		$\frac{1.}{12}$	$\frac{1.}{16}$	$\frac{1.}{8}$				$\sum_i w_i \cdot$
level l		$n_i^{(l)}$ =			$w_i \prod_{k=0}^{l} n_i^{(k)}$			$\prod_{k=0}^{l} n_i^{(k)}$
(e.g. ...)	$N^{(l)}$							
0 (nodes)	24	3	4	2	0.25	0.25	0.25	→ **0.75**
1 (CPUs)	4	2	2	1	0.50	0.50	0.25	→ 1.25
2 (cores)	8	2	2	2	1.00	1.00	0.50	→ 2.50
	N =	d_i =			$w_i d_i$ =			$\sum_i w_i d_i$ =
product	768	12	16	4	1.00	1.00	0.50	→ 2.50

The most important optimization level is the minimization of the inter-node communication. The minimal possible sum on level 0 is 0.75.

Note that without this multi-level approach, i.e., using only MPI_Dims_create_weighted, the result would be as described in the following table:

		i=0	i=1	i=2				
		w_i=						
		$\frac{1.}{12}$	$\frac{1.}{16}$	$\frac{1.}{8}$				
level l	N =	d_i =			$w_i d_i$ =			$\sum_i w_i d_i$ =
level 0	768	8	12	8	0.67	0.75	1.00	→ 2.42
Possible embedded factorizations on node-level:								
Case A	24	4	6	1	0.333	0.375	0.125	→ 0.833
Case B	24	2	6	2	0.167	0.375	0.25	→ 0.792
Case C	24	4	3	2	0.333	0.188	0.25	→ **0.771**

The multi-level factorization with its total process grid dimensions (d_i) = 12x16x4 processes allows directly for an optimal node-to-node communication using a virtual node grid with 3x4x2 nodes, with the minimal sum of 0.75 as shown in the upper table. The single-level factorization with 8x12x8 processes cannot be directly mapped onto 3x4x2 nodes. This implies that optimization algorithms as described in [8] cannot result in a virtual node grid with 3x4x2 nodes. Possible embedded node-level factorizations are 4x6x1 (Case A), 2x6x2 (Case B), and 4x3x2 (Case C). The minimal sum of 0.771 is achieved in case C, which would cause about 2.8% more communication overhead than a factorization with 3x4x2 nodes with the sum 0.750. Nevertheless, the total sum $\sum_i w_i d_i$ seems with 2.42 more optimal than the total sum of the multi-level optimization, which is only 2.50 (higher is worse!).

Note that the input values in this example were not chosen to show a significant performance gain between the single-level and the multi-level factorization, and that further improvements are possible. For example, if $n_i^{(l)}$ is 1, then one can take into account that there is no communication in this direction i on this hardware level l, and therefore when calculating the sum in (4) on level l, the corresponding term may be removed. Furthermore, with non-cyclic communication in direction i and for the case $n_i^{(l)} = 2$, there may be communication only in one direction (either left or right) on this level l. Therefore, the corresponding term in the sum may be accordingly reduced.

7 BENCHMARKING

A synthetic benchmark was implemented in order to test the presented approach for the computation of the subdomain sizes and the reordering of the processes in the communicator. The benchmark takes the proportions of a global 3-dimensional data mesh as input. For distributing the data mesh to the processes, a virtual process topology is then created using the following three different methods:

- MPI_Dims_create + MPI_Cart_create,
- MPI_Cart_weighted_create with equal weights,
- MPI_Cart_weighted_create with data mesh-based weights.

All three methods provide different factorizations of the given total amount of processes into the three dimensions of the virtual Cartesian process grid and also different mappings of the processes onto the cores of the given hardware cluster of SMP nodes. One can freely choose the ratio g_0:g_1:g_2 of the data mesh sizes. The benchmark then measures the halo communication time for 10 different grid sizes G, enlarging them by a factor 8 at each step, which implies a factor 4 for the communicated halos. For each dimension, the benchmark communicates in both directions in parallel in a ring using 2 times MPI_Irecv followed by 2 times MPI_Send and an MPI_Waitall(2,...). After each communication step, the role of send and receive buffer is swapped to prevent cache effects related to data buffer reuse, which does not reflect the behavior of real applications. The total of 6 messages into the 3 dimensions to the coord+1 and coord-1 neighbors is then repeated 50 times. The transfer time per 6 messages and the total halo size are reported. The resulting average bandwidth is also calculated. Note that the new interface allows to reduce the size of communicated data (see also Table 2) by taking the ratio of the mesh dimensions g_i into account, and also enables a higher bandwidth due to the optimized rank mapping (see also the grey columns in Table 3). Both factors add to the total reduction of the communication time.

7.1 Benchmark Results

The benchmark was run on two different systems: Hazel Hen at HLRS, a Cray XC 40 system with Cray MPI, and IvyMUC at LRZ, an Infiniband cluster with Intel MPI. Table 3 shows the results of the benchmark. In both cases 8 dual socket nodes with 12 core Intel Xeon CPUs were used. The application data mesh ratios g_0:g_1:g_2 in the shown experiment were chosen to be 1:2:4.

The amount of communicated halo data depends on the chosen factorization, as shown in Table 2. Therefore, for comparison, one should only use the communication time.

As it can be seen, the version using MPI_Cart_create_weighted with mesh-based weights achieves the best performance for all halo sizes.

The percentages reflect the time saved compared to the original MPI_Cart_create execution time, which serves as reference time. The benchmark has some shortcomings in eliminating OS and network jitter: Therefore, the results for small message sizes are not exact, and we concentrate only on accumulated halo sizes larger than 1 kByte.

The multi-level factorization and the implied factorization of the total amount of processes into the dimensions (d_i) of the Cartesian virtual process grid are shown in Table 1. Note that for MPI_Dims_create + MPI_Cart_create the multi-level factorization is implied through the sequential ranking in MPI_COMM_WORLD and the non-existing rank reordering in the used MPI libraries on both systems.

In Table 3, one can see that for halo sizes larger than 1 kB, i.e., starting from the 3rd row, using the multi-level factorization of

level	MPI_Dims_create + Cart_create	MPI_Cart_create_weighted	
		equal weights	mesh-based
0=node	8 x 1 x 1	2 x 2 x 2	1 x 2 x 4
1=CPU	1 x 2 x 1	2 x 1 x 1	2 x 1 x 1
2=core	1 x 3 x 4	2 x 3 x 2	2 x 3 x 2
implied (d_i)	8 x 6 x 4	8 x 6 x 4	4 x 6 x 8

Table 1: Implied factorization (d_i) resulting from multi-level factorizations in the three cases.

MPI_Cart_create_weighted with equal weights already results in savings of about 40% to 60% on Hazel Hen and 20%-50% on IvyMUC. These savings are due to the significant reduction of node-to-node communications through the multi-level reordering, and to the fact that CPU-to-CPU and core-to-core communication is significantly faster than node-to-node communication. Note that these measurements use the same halo sizes as the reference benchmark.

In order to appreciate the influence of the two optimizations performed by MPI_Cart_weighted_create, Table 2 lists the halo sizes for the original MPI_Dims_create-based virtual topology together with the MPI_Cart_weighted_create-based decomposition using mesh-based weights. The reduction of the halo size area is around 25%, which is based on the fact that the dimensions (d_i) = 4x6x8 fit significantly better to the ratio of our mesh sizes, with ratios g_0:g_1:g_2 = 1:2:4.

The last 3 columns in Table 3 represent the results of the modified factorization and process rank reordering by taking into account the mesh-based weights and the implied 25% reduction of the halo sizes. One can see that the final savings of the halo communication time on Hazel Hen are about 60% to 75% and on IvyMUC 40% to 70%.

Mesh dimensions g_i (number of floats)	halo size in Byte		
	original	mesh-based	reduction
8 x 12 x 24	160	128	20.0 %
12 x 24 x 48	640	432	32.5 %
24 x 48 x 96	2216	1728	22.0 %
48 x 96 x 184	8864	6688	24.5 %
92 x 186 x 376	34968	26008	25.6 %
184 x 372 x 744	137208	103168	24.8 %
372 x 738 x 1480	548088	411192	25.0 %
740 x 1476 x 2960	2187192	1639840	25.0 %
1476 x 2958 x 5912	8740888	6551480	25.0 %
2956 x 5910 x 11816	34936960	26194104	25.0 %

Table 2: Halo sizes for the virtual topologies created with equal and mesh-based weights for different (global) mesh dimensions. For calculating the subdomain mesh dimensions and the implied halo sizes, these global values must be divided by the number of processes in each direction. The halo sizes are calculated within one MPI process and accumulated over all directions. The halo width is chosen as one.

8 CONCLUSION

This work, investigates the problems of the current MPI interface when it comes to optimal process mapping for applications which are implemented through computations on a Cartesian process grid using a regular stencil and are parallelized via domain decomposition.

An approach to achieve optimal domain decomposition is presented and evaluated with a synthetic benchmark. Our approach takes into account the computational data mesh as well as the hardware topology to optimize inter-process communication. The optimization criteria is the minimization of slow bandwidth communication and is applied hierarchically to the different hardware layers. Based on our generalized approach, we intend to propose the addition of a new API to the MPI standard, which would help to create suitable communicators for such scenarios.

ACKNOWLEDGMENTS

This work has been supported by the EXPERTISE project that has received funding from the European UnionâĂŹs Horizon 2020 research and innovation programme under Grant Agreement No. 721865. Further, the authors would like to thank the members of the MPI Forum and especially Guillaume Mercier and the hardware topology group for their reviewing of the MPI_Cart_create_weighted proposal and providing a lot of feedback for enhancements. The authors would also like to thank Jepser Träff for his brute force algorithm used for the factorization, our collaborators in the MPI+X tutorial, Claudia Blaas-Schenner, Georg Hager and Irene Reichl for their helpful suggestions, all the EuroMPI reviewers for their valuable feedback and corrections, and HLRS and LRZ for the HPC resources used for the benchmarks.

REFERENCES

[1] 2015. MPI: A Message-Passing Interface Standard Version 3.1.
[2] 2017. *Cray Performance Measurement and Analysis Tools User Guide 7.0.0.*
[3] Pavan Balaji et al. 2009-2012. Topology awareness in MPI_Dims_create. https://github.com/mpi-forum/mpi-forum-historic/issues/195. Accessed 2018-07-19.
[4] Abhinav Bhatele, Gagan Raj Gupta, Laxmikant V. Kalé, and I-Hsin Chung. 2010. Automated mapping of regular communication graphs on mesh interconnects. In *2010 International Conference on High Performance Computing, HiPC 2010, Dona Paula, Goa, India, December 19-22, 2010*. 1–10. https://doi.org/10.1109/HIPC.2010.5713190
[5] Abhinav Bhatele and Laxmikant V. Kalé. 2008. Application-specific topology-aware mapping for three dimensional topologies. In *22nd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, Miami, Florida USA, April 14-18, 2008*. 1–8. https://doi.org/10.1109/IPDPS.2008.4536348
[6] Cy P. Chan, John D. Bachan, Joseph P. Kenny, Jeremiah J. Wilke, Vincent E. Beckner, Ann S. Almgren, and John B. Bell. 2016. Topology-Aware Performance Optimization and Modeling of Adaptive Mesh Refinement Codes for Exascale. In *First International Workshop on Communication Optimizations in HPC, COMHPC@SC 2016, Salt Lake City, UT, USA, November 18, 2016*. 17–28. https://doi.org/10.1109/COMHPC.2016.008
[7] Cédric Chevalier and François Pellegrini. 2008. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel computing* 34, 6-8 (2008), 318–331.
[8] Bill Gropp. 2018. Using Node Information to Implement MPI Cartesian Topologies. In *Proceedings of the 25th European MPI Users' Group Meeting (EuroMPI '18)*. ACM, New York, NY, USA.
[9] William D Gropp. 2019. Using node and socket information to implement MPI Cartesian topologies. *Parallel Comput.* 85 (1 7 2019), 98–108. https://doi.org/10.1016/j.parco.2019.01.001
[10] T. Hoefler and M. Snir. 2011. Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In *Proceedings of the 2011 ACM International Conference on Supercomputing (ICS'11)*. ACM, 75–85.
[11] T. Hoefler and M. Snir. 2011. Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In *Proceedings of the 2011 ACM International Conference*

Mesh dimensions (number of floats)	MPI_Cart_create		MPI_Cart_weighted_create					
			equal weights			mesh-based weights		
	time in μs	bw in MB/s	time in μs		bw in MB/s	time in μs		bw in MB/s
8 x 12 x 24	8.240	19.418	23.880	(-189.8 %)	6.700	17.877	(-117.0 %)	7.160
12 x 24 x 48	11.001	58.178	14.424	(-31.1 %)	44.369	7.544	(31.4 %)	57.267
24 x 48 x 96	17.118	129.451	10.023	(41.4 %)	221.089	12.503	(27.0 %)	138.210
48 x 96 x 184	41.280	214.730	24.161	(41.5 %)	366.867	15.459	(62.6 %)	432.627
92 x 186 x 376	107.241	326.070	52.681	(50.9 %)	663.769	38.357	(64.2 %)	678.055
184 x 372 x 744	481.539	284.937	182.381	(62.1 %)	752.317	127.559	(73.5 %)	808.788
372 x 738 x 1480	1931.901	283.704	779.319	(59.7 %)	703.291	480.480	(75.1 %)	855.794
740 x 1476 x 2960	7822.924	279.588	3467.302	(55.7 %)	630.805	1861.439	(76.2 %)	880.953
1476 x 2958 x 5912	32419.939	269.615	14402.356	(55.6 %)	606.907	8416.939	(74.0 %)	778.368
2956 x 5910 x 11816	168891.082	206.861	64691.877	(61.7 %)	540.052	38406.062	(77.3 %)	682.030

(a) Hazel Hen

Mesh dimensions (number of floats)	MPI_Cart_create		MPI_Cart_weighted_create					
			equal weights			mesh-based weights		
	time in μs	bw in MB/s	time in μs		bw in MB/s	time in μs		bw in MB/s
8 x 12 x 24	12.341	12.965	13.499	(-9.4 %)	11.853	11.024	(10.7 %)	11.611
12 x 24 x 48	11.101	57.654	10.300	(7.2 %)	51.264	7.257	(34.6 %)	59.525
24 x 48 x 96	14.482	153.022	11.497	(20.6 %)	176.749	8.240	(43.1 %)	209.715
48 x 96 x 184	23.236	381.473	15.759	(32.2 %)	515.753	10.843	(53.3 %)	616.788
92 x 186 x 376	49.562	705.534	32.277	(34.9 %)	990.175	19.383	(60.9 %)	1341.765
184 x 372 x 744	187.821	730.524	86.279	(54.1 %)	1473.083	53.821	(71.3 %)	1916.886
372 x 738 x 1480	798.960	686.002	449.576	(43.7 %)	1120.361	273.118	(65.8 %)	1505.547
740 x 1476 x 2960	3196.821	684.177	2047.000	(36.0 %)	978.474	1361.361	(57.4 %)	1204.560
1476 x 2958 x 5912	13538.141	645.649	8618.941	(36.3 %)	928.979	5793.381	(57.2 %)	1130.856
2956 x 5910 x 11816	54434.857	641.812	35900.860	(34.0 %)	892.158	22294.002	(59.0 %)	1174.940

(b) IvyMUC

Table 3: Comparison of the communication times between the original MPI_Cart_create **and the virtual topologies created with** MPI_Cart_weighted_create **using equal and mesh-based weights. Values in parenthesis show the percentage of saved execution time compared to the original** MPI_Cart_create **procedure.**

on Supercomputing (ICS'11). ACM, 75–85.

[12] George Karypis and Vipin Kumar. 2009. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. http://www.cs.umn.edu/~metis.

[13] Guillaume Mercier and Emmanuel Jeannot. 2011. Improving MPI Applications Performance on Multicore Clusters with Rank Reordering. In *Recent Advances in the Message Passing Interface - 18th European MPI Users' Group Meeting, EuroMPI 2011, Santorini, Greece, September 18-21, 2011. Proceedings (Lecture Notes in Computer Science)*, Yiannis Cotronis, Anthony Danalis, Dimitrios S. Nikolopoulos, and Jack J. Dongarra (Eds.), Vol. 6960. Springer, 39–49. https://doi.org/10.1007/978-3-642-24449-0

[14] Seyed Hessam Mirsadeghi and Ahmad Afsahi. 2016. Topology-Aware Rank Reordering for MPI Collectives. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2016, Chicago, IL, USA, May 23-27, 2016*. 1759–1768. https://doi.org/10.1109/IPDPSW.2016.139

[15] Christoph Niethammer and Rolf Rabenseifner. 2018. Topology aware Cartesian grid mapping with MPI. In *Poster at the 25th European MPI Users' Group Meeting (EuroMPI '18), Barcelona, Spain.*

[16] François Pellegrini and Jean Roman. 1996. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*. Springer, 493–498.

[17] Mohammad Javad Rashti, Jonathan Green, Pavan Balaji, Ahmad Afsahi, and William Gropp. 2011. Multi-core and Network Aware MPI Topology Functions. In *Recent Advances in the Message Passing Interface - 18th European MPI Users' Group Meeting, EuroMPI 2011, Santorini, Greece, September 18-21, 2011. Proceedings (Lecture Notes in Computer Science)*, Yiannis Cotronis, Anthony Danalis, Dimitrios S. Nikolopoulos, and Jack J. Dongarra (Eds.), Vol. 6960. Springer, 50–60. https://doi.org/10.1007/978-3-642-24449-0

[18] Hari Subramoni, Krishna Chaitanya Kandalla, Jérôme Vienne, Sayantan Sur, Bill Barth, Karen A. Tomko, Robert T. McLay, Karl W. Schulz, and Dhabaleswar K. Panda. 2011. Design and Evaluation of Network Topology-/Speed- Aware Broadcast Algorithms for InfiniBand Clusters. In *2011 IEEE International Conference on Cluster Computing (CLUSTER), Austin, TX, USA, September 26-30, 2011*. 317–325. https://doi.org/10.1109/CLUSTER.2011.43

[19] Hari Subramoni, Sreeram Potluri, Krishna Chaitanya Kandalla, Bill Barth, Jérôme Vienne, Jeff Keasler, Karen A. Tomko, Karl W. Schulz, Adam Moody, and Dhabaleswar K. Panda. 2012. Design of a scalable InfiniBand topology service to enable network-topology-aware placement of processes. In *SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA - November 11 - 15, 2012*. 70. https://doi.org/10.1109/SC.2012.47

[20] Jesper Larsson Träff and Felix Donatus Lübbe. 2015. Specification Guideline Violations by MPI_Dims_Create. In *Proceedings of the 22nd European MPI Users' Group Meeting (EuroMPI '15)*. ACM, New York, NY, USA, Article 19, 2 pages.