Topology aware Cartesian grid mapping with MPI

Issue #120 Reading

Rolf Rabenseifner
rabenseifner@hlrs.de

Christoph Niethammer
niethammer@hlrs.de

High Performance Computing Center (HLRS), University of Stuttgart, Germany

Virtual MPI Forum Meeting June 10, 2020

Exceptional virtual MPI Forum Meeting June 29 – July 1, 2020 (originally planned for Munich)

For further information, see https://github.com/mpi-forum/mpi-issues/issues/120

Goals of Cartesian MPI_Cart/Dims_create

- Given: comm_old (e.g., MPI_COMM_WORLD), ndims (e.g., 3 dimensions)
- Provide
  - a **factorization** of #processes (of comm_old) into the dimensions **dims**$[i]_{i=1..ndims}$
  - a Cartesian communicator **comm_cart**
  - a **optimized reordering** of the ranks in comm_old into the ranks of comm_cart to minimize the Cartesian communication time, e.g., of
    - MPI_Neighbor_alltoall
    - Equivalent communication pattern implemented with
      - MPI_Sendrecv
      - Nonblocking MPI point-to-point communication

The limits of MPI_Dims_create + MPI_Cart_create

- Not application topology aware
  - MPI_Dims_create can **only** map **evenly balanced** Cartesian topologies
    - Factorization of 48,000 processes into 20 x 40 x 60 processes
      (e.g. for a mesh with 200 x 400 x 600 mesh points)
      → no chance with current interface
- Only partially hardware topology aware
  - MPI_Dims_create has no communicator argument → not hardware aware
    - An application mesh with 3000x3000 mesh points
      on 25 nodes x 24 cores (=600 MPI processes)
      - Answer from MPI_Dims_create:
        - » 25 x 24 MPI processes
        - » Mapped by most libraries to 25 x 1 nodes with 120x3000 mesh points per node
          ➔ too much node-to-node communication

Major problems:

- No weights, no info
- Two separated interfaces for two common tasks:
  - Factorization of #processes
  - Mapping of the processes to the hardware

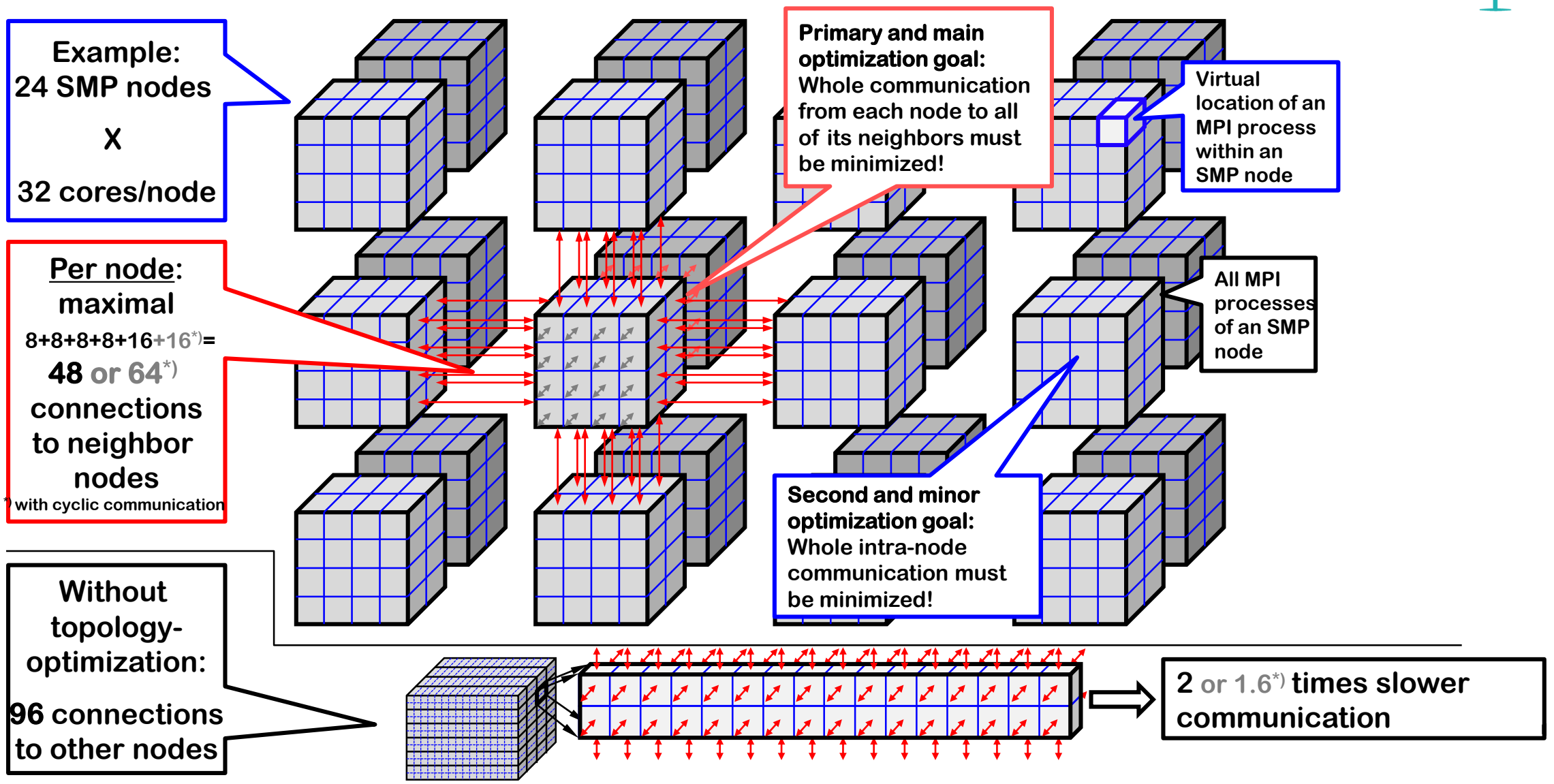
Goals of Cartesian MPI_Cart/Dims_create

- Remark: On a hierarchical hardware,
  - **optimized factorization and reordering** typically means **minimal node-to-node** communication,
  - which typically means that the communicating surfaces of the data on each node is as quadratic as possible (or the subdomain as cubic as possible)

- In the current API, i.e.,
  - due to the missing weights
  - and the non-hardware aware MPI_Dims_create,

  does **not** allow such an optimized factorization and reordering in many cases.

Hierarchical Cartesian Domain Decomposition



The new Interface

- **MPI_Dims_create_weighted** (

```
/*IN*/       int          nnodes,
/*IN*/       int          ndims,
/*IN*/       int          dim_weights[ndims],
/*IN*/       int          periods[ndims], /* for future use in combination with info */
/*IN*/       MPI_Info     info, /* for future use, currently MPI_INFO_NULL */
/*INOUT*/    int          dims[ndims]);
```

input for application-topology-awareness

  - Arguments have same meaning as in MPI_Dims_create
  - Goal (in absence of an info argument):
    - dims[i]•dim_weights[i] should be as close as possible,
    - i.e., the $\sum_{i=0..(ndims-1)}$ dims[i]•dim_weights[i] as small as possible (advice to implementors)

A new courtesy function: Weighted factorization

The new Interface, continued

input for hardware-awareness

input for application-topology-awareness

- **MPI_Cart_create_weighted** (

```
/*IN*/       MPI_Comm   comm_old,
/*IN*/       int        ndims,
/*IN*/       int        dim_weights[ndims], /*or MPI_UNWEIGHTED*/
/*IN*/       int        periods[ndims],
/*IN*/       MPI_Info   info,       /* for future use, currently MPI_INFO_NULL */
/*INOUT*/    int        dims[ndims],
/*OUT*/      MPI_Comm   *comm_cart );
```

  - Arguments have same meaning as in MPI_Dims_create & MPI_Cart_create
  - See next slide for meaning of dim_weights[ndims]
  - Goal: chooses
    - an ndims-dimensional factorization of #processes of comm_old (→ dims)
    - and an appropriate reordering of the ranks (→ comm_cart),

    such that the execution time of a communication step along the virtual process grid (e.g., with MPI_NEIGHBOR_ALLTOALL or equivalent calls to MPI_SENDRECV as in the example in Section 7.6.2) is as small as possible.

The new application & hardware topology aware interface

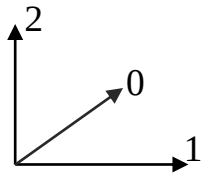
How to specify the dim_weights?

- Given: comm_old (e.g., MPI_COMM_WORLD), ndims (e.g., 3 dimensions)
- This means, **the domain decomposition has not yet taken place**!

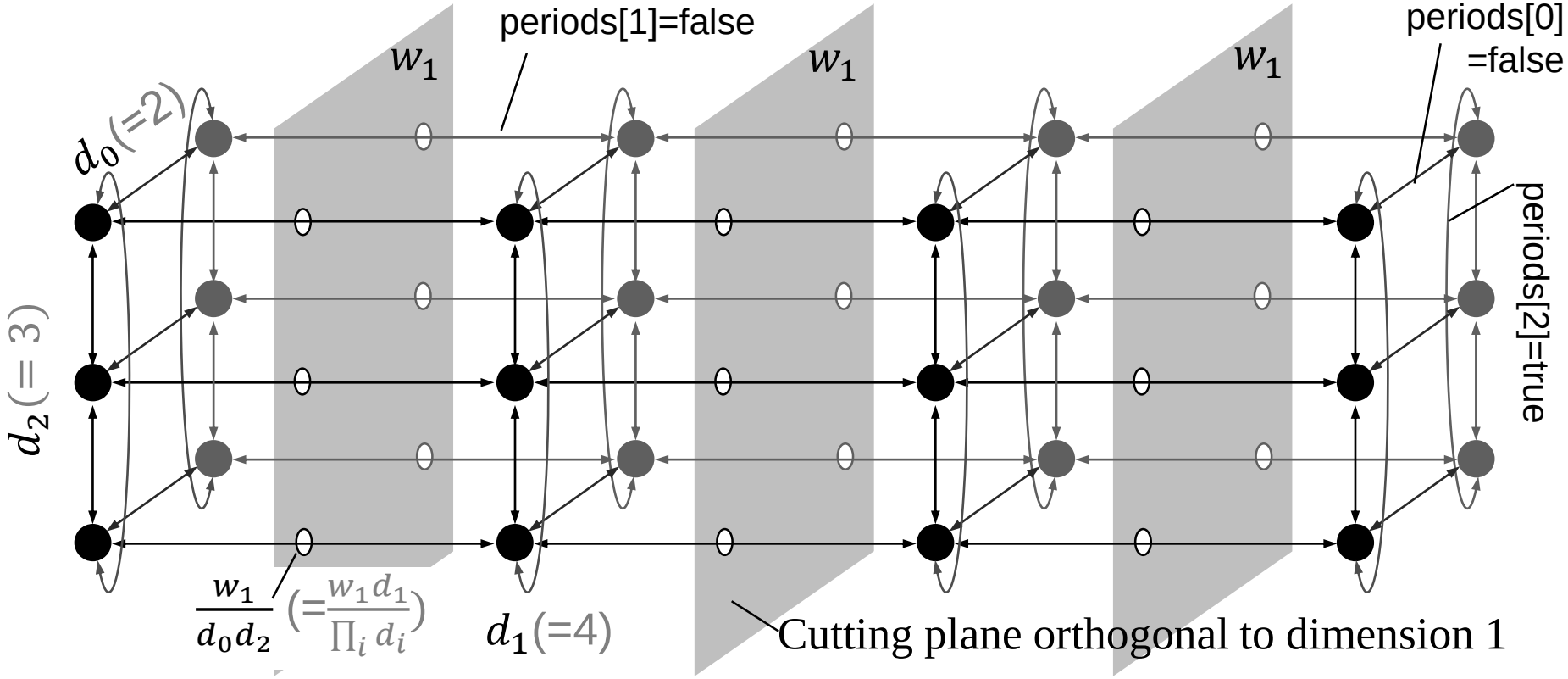
- Goals for dim_weights and the API at all:
  - Easy to understand
  - Easy to calculate
  - Relevant for typical Cartesian communication patterns (MPI_Neighbor_alltoall or alternatives)
  - Rules fit to usual design criteria of MPI
    - E.g., reusing MPI_UNWEIGHTED → integer array
    - Can be enhanced by vendors for their platforms
      → additional info argument for further specification
    - To provide also the less optimal two stage interface (in addition to the combined routine)

The dim_weights[i], example with 3 dimensions

Abbreviations:
d_i = dims[i]
w_i = dim_weights[i]
with
i = 0..(ndims-1)

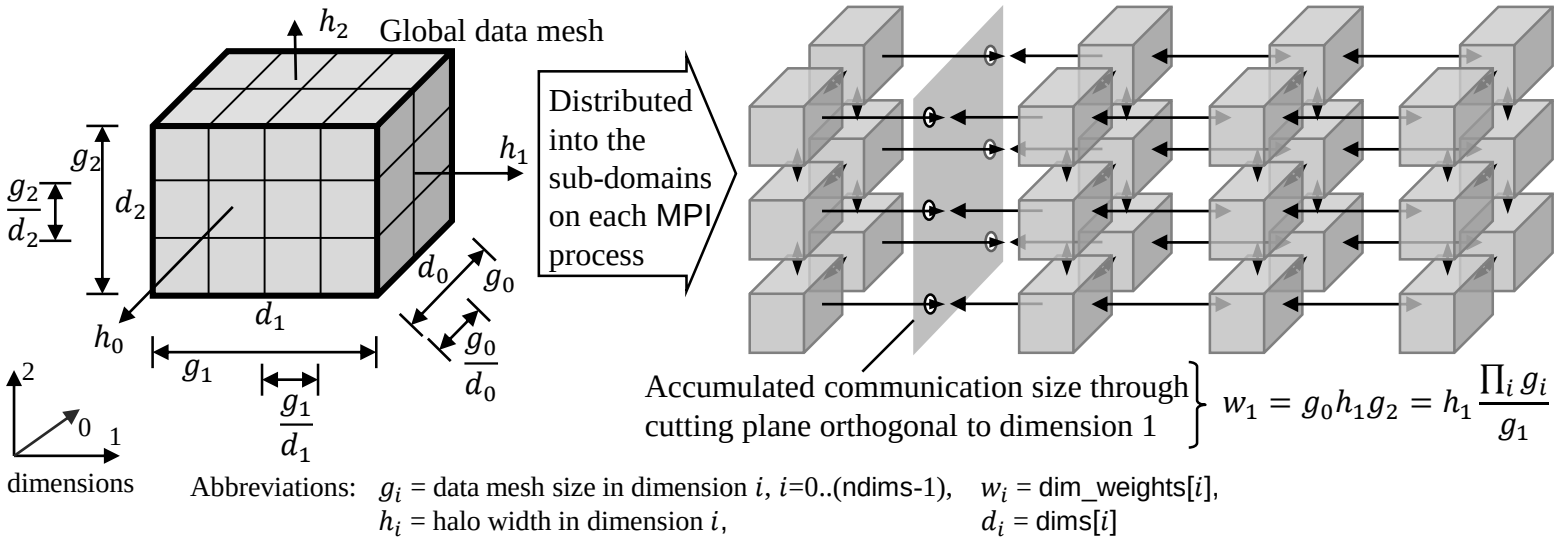


Three dimensions,
i.e., ndims=3



The arguments dim_weights[i] i =0::(ndims-1), abbreviated with w_i, should be specified as the accumulated message size (in bytes) communicated in one communication step through each cutting plane orthogonal to dimension d_i and in each of the two directions.

The dim_weights[i], example with 3 dimensions, continued



Abbreviations: g_i = data mesh size in dimension i, i=0..(ndims-1), w_i = dim_weights[i],
h_i = halo width in dimension i, d_i = dims[i]

Example for the calculation of the accumulated communication size $w_{i,i=0..2}$ in each dimension.

- g_i – The data mesh sizes $g_{i,i=0..2}$ express the three dimensions of the total application data mesh.
- h_i – The value h_i represents the halo width in a given direction when the 2-dimensional side of a subdomain is communicated to the neighbor process in that direction.

Output from MPI_Cart/Dims_create_weighted: The dimensions $d_{i,i=0..2}$

Important:

- The definition of the dim_weights (= w_i in this figure) is independent of the total number of processes and its factorization into the dimensions (= d_i in this figure)
- Result was

$$w_i = h_i \frac{\prod_j g_j}{g_i}$$

Simple answers to our problems / examples

- Existing API is not application topology aware
  - Factorization of 48,000 processes into 20 x 40 x 60 processes → no chance with current API (e.g. for a mesh with 200 x 400 x 600 mesh points)
  - Use MPI_Cart_create_weighted with the dim_weights=(N/200, N/400, N/600) with N=200•400•600
- Existing API is only partially hardware topology aware
  - An application mesh with 3000x3000 mesh points (i.e., example with MPI_UNWEIGHTED) on 25 nodes x 24 cores (=600 MPI processes)
    - Current API must factorize into 25 x 24 MPI processes
      - » 25 x 1 nodes → 120x3000 mesh points → too much node to node communication
    - Optimized answer from MPI_Carte_create_weighted may be:
      - » 30 x 20 MPI processes
      - » Mapped to 5 x 5 nodes with 600x600 mesh points per node → minimal node-to-node communication

Reading of the changes during the Portland Meeting

- Issue: https://github.com/mpi-forum/mpi-issues/issues/120
- PR: https://github.com/mpi-forum/mpi-standard/pull/98
- Annotated PDF:
  https://github.com/mpi-forum/mpi-issues/files/4758790/mpi-report-issue120-topol-2020-02-21-annotated.pdf
  (State: End of Portland meeting Feb. 21, 2020 plus small (typo) corrections = PR98 from June 10, 2020)

Thank you for your interest / any questions?