Topology aware Cartesian grid mapping with MPI

Software Documentation

Christoph Niethammer Rolf Rabenseifner

niethammer@hlrs.de **rabenseifner@hlrs.de**

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

HLRS, Stuttgart, February 12, 2019

EuroMPI 2018

For further information, see https://fs.hlrs.de/projects/par/mpi/EuroMPI2018-Cartesian/

Back to the problems

1. All MPI libraries provide the necessary interfaces ☺ ☺ ☺, but **without** re-numbering in nearly all MPI-libraries ☹ ☹ ☹
   - **You may substitute MPI_Cart_create() by the software solution of Bill Gropp** (see Bill Gropp, EuroMPI 2018)

2. **The existing MPI-3.1 interfaces are not optimal**
   - **for cluster of ccNUMA node hardware,**
     - **We substitute MPI_Dims_create() + MPI_Cart_create() by MPIX_Cart_weighted_create(… MPIX_WEIGHTS_EQUAL …)**
   - **nor for application specific grid sizes or direction-dependent bandwidth**
     - **by MPIX_Cart_weighted_create(… weights ….)**

3. **Caution: The application must be prepared for rank re-numbering**
   - **All communication through the newly created Cartesian communicator with re-numbered ranks!**
   - **One must not load data based on MPI_COMM_WOLRD ranks!**

The new interfaces

Substitute for / enhancement to existing MPI-1

- MPI_Dims_create (size_of_comm_old, ndims, *dims[ndims]*);
- MPI_Cart_create (comm_old, ndims, dims[ndims], periods, reorder, **comm_cart*);

New:

- **MPIX_Cart_weighted_create** (

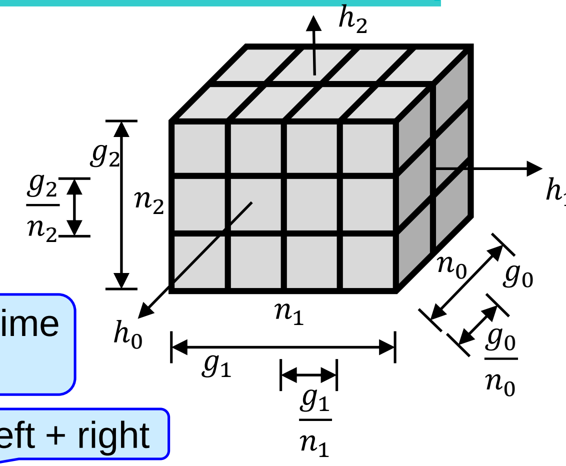
```
/*IN*/      MPI_Comm   comm_old,
/*IN*/      int        ndims,
/*IN*/      double     dim_weights[ndims], /*or MPIX_WEIGHTS_EQUAL*/
/*IN*/      int        periods[ndims],
/*IN*/      MPI_Info   info,      /* for future use, currently MPI_INFO_NULL */
/*INOUT*/   int        dims[ndims],
/*OUT*/  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

The weights w_i

User level

- **Given:**
  - d-dimensional Cartesian grid with a total grid size of $G = \prod_{i=0}^{d-1} g_i$ elements
  - The communication cost in each direction $i = 0, d$-1 is multiplied
    - with a halo width h_i,
    - and a communication cost factor c_i,



communication time per grid point

2 = left + right

- → total communication cost is $2g_1g_2h_0c_0 + 2g_0g_2h_1c_1 + 2g_0g_1h_2c_2$
- The weight w_i is defined as total cost for the communication in one direction:
  - $w_i = 2\frac{G}{g_i}h_ic_i$

common factors, like $2G$ or absolute values of c_i, are not relevant

MPI library level

- With a domain decomposition (i.e., factorization) to $N = \prod_{i=0}^{d-1} n_i$ nodes, the total communication costs per node is

$$2\frac{g_1g_2}{n_1n_2}h_0c_0 + 2\frac{g_0g_2}{n_0n_2}h_1c_1 + 2\frac{g_0g_1}{n_0n_1}h_2c_2 = \sum_{i=0}^{d-1}\frac{n_i}{N}w_i$$

Primarily for the node decomposition and secondarily on core level

➔ The topology functions have to find a factorization with minimal $\sum_{i=0}^{d-1} n_i w_i$

→ common factors $2G$ and $\frac{1}{N}$ are not relevant →

MPIX routines, courtesy of Christoph Niethammer, HLRS

Further Interfaces

Substitute for / enhancement to existing MPI-1
MPI_Dims_create (size_of_comm_old, ndims, *dims*);
MPI_Cart_create (comm_old, ndims, dims, periods, reorder, **comm_cart*);

e.g., with 25*25*24 = **15000 processes** on **625 ccNUMA nodes** with **2 CPUs/node** and **12 cores/CPU**

e.g., { MPI_COMM_TYPE_SHARED, OMPI_COMM_TYPE_NUMA within OpenMPI, or for other MPIs splitting into half nodes: MPIX_COMM_TYPE_HALFNODE }

MPIX_Cart_ml_create_from_types(MPI_Comm comm_old,
int ntype_levels, int type_levels[ntype_levels],
int ndims, double dim_weights[ndims],
int periods[ndims], MPI_Info info,
/*OUT*/ *int dims[ndims], MPI_Comm *comm_cart*);

e.g., 3 dimensions with a data grid with 1000 x 1100 x 950 elements ➔ dim_weights[] = { 1.0/1000, 1.0/1100, 1.0/950 }

Rank mapping is based on:
- Node level: **625** = 5 x 25 x 5
- CPU level: **2** = 2 x 1 x 1
- Core level: **12** = 3 x 1 x 4

Result (product): **30 x 25 x 20**

The Cartesian communicator reflects this result: **30 x 25 x 20**

Next steps:
MPI_Comm_rank (comm_cart, &my_rank);
MPI_Cart_coords (comm_cart, my_rank, ndims, *coords*)

MPIX_Cart_ml_create_from_comms(int nlevels,
MPI_Comm level_comms[nlevels],
int ndims, double dim_weights[ndims], int periods[ndims], MPI_Info info,
/*OUT*/ *int dims[ndims], MPI_Comm *comm_cart*);

e.g., level_comms[0] is comm_old, level_comms[1 and 2] are the result recursively called MPI_Comm_split_type with the type_levels from above.

Same as above

Same as above

MPIX_Dims_weighted_create (int nnodes, int ndims, double dim_weights[ndims],
/*OUT*/ *int dims[ndims]*);

MPIX_Dims_ml_create (int nnodes, int ndims, double dim_weights[ndims],
int nlevels, int sizes[nlevels], **/*OUT*/** *int dims_ml[ndims][nlevels]*);

Multi-level info

Further Interfaces

Substitute for / enhancement to existing MPI-1
MPI_Dims_create (size_of_comm_old, ndims, *dims*);
MPI_Cart_create (comm_old, ndims, dims, periods, reorder, **comm_cart*);

From previous slide

MPIX_Dims_ml_create (int size_of_comm_old, int ndims, double dim_weights[ndims], int nlevels, int sizes[nlevels], **/*OUT*/** *int dims_ml[ndims][nlevels]*);

MPIX_Cart_ml_create (MPI_Comm comm_old, int ndims, int *periods, int nlevels, int dims_ml[ndims][nlevels], MPI_Info info, **/*OUT*/** *int *dims, MPI_Comm *comm_cart*);

This interface requires that comm_old is **ranked sequentially in the hardware**

We proposed the algorithm in

- Christoph Niethammer and Rolf Rabenseifner. 2018. Topology aware Cartesian grid mapping with MPI. EuroMPI 2018. https://eurompi2018.bsc.es/ → Program → Poster Session → Abstract+Poster
- https://fs.hlrs.de/projects/par/mpi/EuroMPI2018-Cartesian/ → All info + slides + software
- http://www.hlrs.de/training/par-prog-ws/ → Practical → MPI.tar.gz → MPI/course/C/eurompi18/

Here, you get the new optimized interface + implementation + documentation

MPIX_Dims_weighted_create() is based on the ideas in:

- 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.

Remarks

- The portable MPIX routines internally use MPI_Comm_split_type(…, **MPI_COMM_TYPE_SHARED**, …) to split comm_old into ccNUMA nodes,
- plus (may be) additional splitting into NUMA domains.
- With using hyperthreads, it ***may be helpful*** to apply **sequential** ranking to the hyperthreads,
  - i.e., in MPI_COMM_WORLD, ranks 0+1 should be
    - **the first two hyperthreads**
    - of the first core
    - of the first CPU
    - of the first ccNUMA node
- Especially with weights $\boldsymbol{w_i}$ based on $\frac{G}{g_i}$, it is important
  - that the data of the grid points is **not** read in based on (**old**) ranks in MPI_COMM_WORLD,
  - because the domain decomposition must be done based on **comm_cart** and its dimensions and (**new**) ranks

Internal implementation plan

- **MPIX_Cart_weighted_create(...)**
  - chooses available and useful types for splitting, e.g., {MPI_COMM_TYPE_SHARED, OMPI_COMM_TYPE_NUMA or MPIX_COMM_TYPE_HALFNODE}
  - → **MPIX_Cart_ml_create_from_types(...)**
- **MPIX_Cart_ml_create_from_types(...)**
  - loop over MPIX_Comm_split_type
  - → **MPIX_Cart_ml_create_from_comms(...)**
- **MPIX_Cart_ml_create_from_comms(...)**
  - must calculate level_sizes[nlevels] and wether they are equally sized within the same level
  - if (equally-sized) then
    - → **MPIX_Dims_ml_create(...)**
    - Appropriate renumbering based on dims_ml and the level_comms
    - Calculation of dims[] & creation of comm_cart → **MPI_Cart_create(…)** without reorder
  - else, e.g., algorithm of Thorsten Hoefler
- **MPIX_Cart_ml_create(...)** → Usable only for sequentially ranked comm_old
  - Appropriate renumbering based on dims_ml and the sequential comm_old
  - Calculation of dims[] & creation of comm_cart → **MPI_Cart_create(…)** without reorder
- **MPIX_Dims_ml_create(...)**
  - → **MPIX_Dims_weighted_create(...)** on each level
- **MPIX_Dims_weighted_create(...)**
  - This is the important new base routine with a new fast brute force algorithm

Benchmark: halo_irecv_send_toggle_3dim_grid_solution.c

- Input per measurement, e.g.on 8 nodes x 2 CPUs x 12 cores: **Example**
  - cart_method: 2
    - **0=end, 1=Dims_create+Cart_create,**
    - **2=Cart_weighted_create(MPIX_WEIGHTS_EQUAL),**
    - **3=dito(weights), 4=dito manually, 5=Cart_ml_create(dims_ml)**
  - start grid sizes integer start values 1 2 4
  - Using MPI_Type_vector, **for each dimension a pair of blocklength&stride** 0 0 0 0 0 0
    - 0 0 = contiguous
  - weights (double values) **(only with cart_method==4)** 1.00 0.50 0.25
  - number of hardware levels **(only with cart_method==5)** 3
    dims_ml: for each of the 3 Cartesian dimensions a list of 3 dimensions from outer to inner hardware level, e.g., 8 nodes x 2 CPUs x 12 cores are split into 1x2x4 nodes x 2x1x1 CPUs x 2x3x2 cores
    - **dims_ml[d=0] =** 1 2 2
    - **dims_ml[d=1] =** 2 1 3
    - **dims_ml[d=2] =** 4 1 2
- Input can be concatenated to one line per experiment:
  - 1 1 2 4 0 0 0 0 0 0
  - 2 1 2 4 0 0 0 0 0 0
  - 3 1 2 4 0 0 0 0 0 0
  - 4 1 2 4 0 0 0 0 0 0 4. 2. 1.
  - 5 1 2 4 0 0 0 0 0 0 3 1 2 2 2 1 3 4 1 2
  - 3 2 2 2 256 1024 4 32 0 0
  - 0

examples for strided data in direction 0 & 1

Start a 16-node batch-job with your own input file: **Report your acceleration factors to the course group**

Additional Remarks

- Caution with stdout and stdin when switching I/O from process world_rank==0 to cart_rank==0:
  - **Before** establishing the new comm_cart, all I/O on stdout/stdin is done by world_rank==0 (in **MPI_COMM_WORLD)**
  - **After** establishing the new comm_cart, all I/O on stdout/stdin is done by cart_rank==0 (in comm_cart)
  - In between, we recommended (although it is not guaranteed that an *output on comm_cart* may overtake an *output on MPI_COMM_WORLD*):
    - **MPI_Barrier(MPI_COMM_WORLD);**
    - **sleep(1);** // costs nearly nothing, e.g., 30 Mio € TCO/year / (365 days/year * 24 hours/day * 3600 sec/hour) * 1 sec = 1€
    - **MPI_Barrier(comm_cart);**
- The following slide shows the win through the re-ranking by the new routines:
  - Less % is better – the communication time reduction factors are:
    - **1.1-1.2**
    - **1.75**
    - **2.75**
    - **4.5-5.0**

Halosize/process ~= 26 MB		MPI_Dims_create + MPI_Cart_create		MPIX_Cart_weighted_create(MPIX_WEIGHTS_EQUAL)		MPIX_Cart_weighted_create(…weights…)	
Base grid sizes	Nodes x CPUs x cores	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communication time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]
2 x 2 x 2	8x2x12	84.545 = baseline	8 = 8 x 1 x 1 6 = 1 x 2 x 3 4 = 1 x 1 x 4	52.666 = 62%	8 = 2 x 2 x 2 6 = 2 x 1 x 3 4 = 2 x 1 x 2	48.556 = 57%	8 = 2 x 2 x 2 6 = 2 x 1 x 3 4 = 2 x 1 x 2
	64x2x12	194.856 = baseline	16=16 x 1 x 1 12= 4 x 1 x 3 8= 1 x 2 x 4	73.756 = 38%	16= 4 x 2 x 2 12= 4 x 1 x 3 8= 4 x 1 x 2	72.051 = 37%	16= 4 x 2 x 2 12= 4 x 1 x 3 8= 4 x 1 x 2
	512x2x12	247.631 = baseline	32=32 x 1 x 1 24=16 x 1x1.5 16= 1 x 2 x 8	85.530 = 35%	32= 8 x 2 x 2 24= 8 x 1 x 3 16= 8 x 1 x 2	85.491 = 35%	32= 8 x 2 x 2 24= 8 x 1 x 3 16= 8 x 1 x 2
1 x 2 x 4	8x2x12	172.850 = baseline	8 = 8 x 1 x 1 6 = 1 x 2 x 3 4 = 1 x 1 x 4	63.796 = 37%	8 = 2 x 2 x 2 6 = 2 x 1 x 3 4 = 2 x 1 x 2	37.953 = 22%	4 = 1 x 2 x 2 6 = 2 x 1 x 3 8 = 4 x 1 x 2
	64x2x12	360.364 = baseline	16=16 x 1 x 1 12= 4 x 1 x 3 8= 1 x 2 x 4	91.524 = 25%	16= 4 x 2 x 2 12= 4 x 1 x 3 8 = 4 x 1 x 2	74.199 = 21%	8 = 2 x 2 x 2 12= 4 x 1 x 3 16= 8 x 1 x 2
	512x2x12	457.858 = baseline	32=32 x 1 x 1 24=16 x 1x1.5 16= 1 x 2 x 8	125.468 = 27%	32= 8 x 2 x 2 24= 8 x 1 x 3 16= 8 x 1 x 2	93.615 = 20%	16= 4 x 2 x 2 24= 8 x 1 x 3 32=16 x 1 x 2
4 x 2 x 1	8x2x12	40.050 = baseline	8 = 8 x 1 x 1 6 = 1 x 2 x 3 4 = 1 x 1 x 4	59.421 =148%	8 = 2 x 2 x 2 6 = 2 x 1 x 3 4 = 2 x 1 x 2	36.778 =92%	16 = 4 x 2 x 2 6 = 2 x 1 x 3 2 = 1 x 1 x 2
	64x2x12	78.503 = baseline	16=16 x 1 x 1 12= 4 x 1 x 3 8= 1 x 2 x 4	100.203 =128%	16= 4 x 2 x 2 12= 4 x 1 x 3 8 = 4 x 1 x 2	69.802 =89%	32= 8 x 2 x 2 12= 4 x 1 x 3 4 = 2 x 1 x 2
	512x2x12	103.002 = baseline	32=32 x 1 x 1 24=16 x 1x1.5 16= 1 x 2 x 8	93.189 = 90%	32= 8 x 2 x 2 24= 8 x 1 x 3 16= 8 x 1 x 2	85.044 =83%	64=16 x 2 x 2 24= 8 x 1 x 3 8= 4 x 1 x 2

Depend on chosen dims

Used process dimensions

Internally applied dimen-sions on each hardware level

Base factors, not absolute values

By default, communica-tion time strongly depends on cuboid's direction

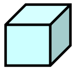
Optimized communi-cation times are nearly indepen-dent of cubid's form

On Cray XC40 Hazel hen at HLRS Stuttgart, Jan 2019

As Exercise: To do (1)

- **cp ~/MPI/course/C/Ch9/MPIX_*/* .**
  - You get the benchmark skeleton halo_irecv_send_toggle_3dim_grid_skel.c
  - And all MPIX_*.c files and the header MPIX_interface_proposal.h
- **mpicc –o halo_skel.exe halo_irecv_send_toggle_3dim_grid_skel.c MPIX_*.c**
- First test with non-optimized cart_method==1**, i.e., MPI_Dims_create + MPICart_create**
  - Choose your batch job: **halo_skel_[LRZ|VSC|HLRS].sh** which contains
    - **Number of nodes and cores/node**
    - **and, e.g.,**
      mpirun –np 192 ./halo_skel.exe < input-skel.txt

```
1   2 2 2   0 0 0 0 0 0
1   1 2 4   0 0 0 0 0 0
0
```

  - Start your batchjob
  - Try to understand the output:
    - **It contains two experiments: a grid with cubic**  **and one with non-cubic**  **ratio**
    - **The number of MPI processes, e.g. 192, is factorized**
      → domain decomposition into, e.g., 8 x 6 x 4 processes
    - **The measurements are done for 10 global gridsizes**
    - **The domain decomposition implies the local gridsizes**
    - **The local gridsizes imply the size of the halos in each direction**
    - **→ the sum of the time for the communication into the 3 dimensions x 2 directions (left+right)**

See **next slide**

Exercise: To do (2)

```
cart_method = 1
start grid sizes integer start values for 3 dimensions = 2 2 2
blocklength & sgtride pairs for each of the 3 dimensions = 1 0   1 0   1 0
Creating the Cartesian communicator and further input arguments:
cart_method == 1: MPI_Dims_create + MPI_Cart_create
[MPI_Barrier and switching to output via stdout through rank==0 in comm_cart]
ndims=3 dims= 8 6 4
  message size     transfertime   duplex bandwidth per process and neighbor (grid&halo in #floats)
                                                 gridsizes total=   per process=   halosizes=
     128 bytes      34.537 usec    3.706 MB/s     16   12   12       2   2    3          16=     6 +     6 +      4
     432 bytes      39.840 usec   10.843 MB/s     24   24   24       3   4    6          54=    24 +    18 +     12
    1728 bytes      41.122 usec   42.021 MB/s     48   48   48       6   8   12         216=    96 +    72 +     48
    6688 bytes      23.961 usec                   96   96   92      12  16   23         836=   368 +   276 +    192
   25576 bytes      93.703 usec                  184  186  184      23  31   46        3197=  1426 +  1058 +    713
  104408 bytes     271.721 usec                  376  372  372      47  62   93       13051=  5766 +  4371 +   2914
  411192 bytes    1033.001 usec       .056 MB/s  744  738  740      93 123  185       51399= 22755+17205 + 11439
 1636392 bytes    4398.680 usec    372.019 MB/s 1480 1476 1476     185 246  369      204549= 90774+68265 + 45510
 6561336 bytes   18173.518 usec    361.038 MB/s 2960 2958 2956     370 493  739      820167=364327+273430+182410
26194104 bytes   76132.216 usec    344.061 MB/s 5912 5910 5908     739 985 1477     3274263=1454845+1091503+727915
```

```
cart_method = 1
start grid sizes integer start values for 3 dimensions = 1 2 4
blocklength & sgtride pairs for each of the 3 dimensions = 0 0   0 0   0 0
cart_method == 1: MPI_Dims_create + MPI_Cart_create
ndims=3 dims= 8 6 4
  message size       transfertime   duplex bandwidth per process and neighbor (grid&halo in #floats)
     160 bytes        14.720 usec   10.870 MB/s     8   12    24      1   2    6          20=     12 +     6 +      2
  …
34936960            156869.278 usec  222.714 MB/s 2960 5910 11816   370 985 2954     4367120=2909690+1092980+364450
```

192 processes: Input for `MPI_Dims_create()`

These base values (per process) are multiplied with $\sqrt[3]{\#processes}$ and then with 1, 2, 4, 8, … 512, e.g., $2 \cdot \sqrt[3]{192} \cdot 512 =$ **5912** (rounded to a multiple the dimension)

divide by dims[i]

First value for our table

* 2 directions * 4 byte

multiply for 2-d halo array size

Same values, because MPI_Dims_create() factorizes the #processes independent from the user's gridsizes.

Second value for our table

Exercise: To do (3)

- **Fill in the table**

Given from MPI_Dims_create()

Nodes CPUs cores

d=0: **8** = 8 x 1 x 1
d=1: **6** = 1 x 2 x 3
d=2: **4** = 1 x 1 x 4

Total 192 = **8 x 2 x12**

Given by batchjob and hardware

Please, calculated by hand:
Fill in maximal factors.
Factorize first the cores and start with d=2.
Then the CPUs & then the nodes.
(All based on sequential ranking of MPI_COMM_WORLD)

Defined in batch job + hardware knowledge

Execution time of **largest** grid and halo size of both measurements

Halosize/process ~= 26 MB		MPI_Dims_create + MPI_Cart_create		MPIX_Cart_weighted_create(MPIX_WEIGHTS_EQUAL)		MPIX_Cart_weighted_create(…weights…)	
Base grid sizes	Nodes x CPUs x cores	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communication time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]
2 x 2 x 2	__x__x__	______ = baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _				
1 x 2 x 4	Same as above	= baseline	Same as above				

Exercise: To do (4)

- **cp halo_irecv_send_toggle_3dim_grid_skel.c halo_optim.c**
- **Edit halo_optim.c**
  - On lines 160, 165, 171, and 184, substitute the **/* TODO: … */** by correct code

```
153  if (cart_method == 1) {
154    if (my_world_rank==0) printf("cart_method == 1: MPI_Dims_create + MPI_Cart_create\n");
155    MPI_Dims_create(size, ndims, dims);
156    MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, 0, &comm_cart);
157  } else if (cart_method == 2) {
158    if (my_world_rank==0) printf("cart_method == 2: MPIX_Cart_weighted_create( MPIX_WEIGHTS_EQUAL )\n");

160    /* TODO: Appropriate call to MPIX_Cart_weighted_create(...) with MPIX_WEIGHTS_EQUAL
161             instead of calling MPI_Dims_create() and MPI_Cart_create() as in method 1 */

163  } else if (cart_method == 3) {

165    /* TODO: Appropriate calculation of weights[ ] based on gridsize_avg_per_proc_startval[ ] */

167    if (my_world_rank==0) { printf("cart_method == 3: MPIX_Cart_weighted_create( weights := ___________TODO___________)\n");
168      printf("weights= "); for (d=0; d<ndims; d++) printf(" %lf",weights[d]); printf("\n");
169    }

171    /* TODO: Appropriate call to MPIX_Cart_weighted_create(...) with weights
172             instead of MPIX_WEIGHTS_EQUAL as in method 2 */

174  } else if (cart_method == 4) {
175    for (d=0; d<ndims; d++) weights[d] = 4.0 / gridsize_avg_per_proc_startval[d];
176    if (my_world_rank==0) { printf("cart_method == 4: MPIX_Cart_weighted_create( manual weights )\n");
177      printf("weights (double values) for %d dimensions (e.g., ", ndims);
178      for (d=0; d<ndims; d++) printf(" %lf",weights[d]); printf(") ?\n");
179      for (d=0; d<ndims; d++) scanf("%lf",&weights[d]);
180      printf("weights= "); for (d=0; d<ndims; d++) printf(" %lf",weights[d]); printf("\n");
181    }
182    MPI_Bcast(weights, ndims, MPI_DOUBLE, 0, MPI_COMM_WORLD);

184    /* TODO: Appropriate call to MPIX_Cart_weighted_create(...)
185             same as in method 3, but without the calculation of the weights */

187  } else { …
```

Exercise: To do (5)

- **mpicc –o halo_optim.exe halo_optim.c MPIX_*.c**
- Check: **diff halo_optim.c halo_irecv_send_toggle_3dim_grid_solution.c**
- Now, use all three cart_method==1, 2, 3
  - – Choose your batch job:
    - **halo_optim_[LRZ|VSC|HLRS].sh** which contains:
    - **mpirun –np 192 ./halo_optim.exe < input-optim.txt**
  - – Start your batchjob → output file **output_optim.txt**
  - – Fill in the table

Note, that the optimization changes the dims-array → modified halo sizes!

Although halos may be larger, the optimized communication time should be shorter!

Base gridsizes — Block length + stride for each dimension — **Cart_method**

```
1  2 2 2  0 0  0 0  0 0
2  2 2 2  0 0  0 0  0 0
1  1 2 4  0 0  0 0  0 0
2  1 2 4  0 0  0 0  0 0
3  1 2 4  0 0  0 0  0 0
0
```

0 = end

Halosize/process ~= 26 MB		**cart_method==1** MPI_Dims_create + MPI_Cart_create		**cart_method==2** MPIX_Cart_weighted_ create(MPIX_WEIGHTS_EQUAL)		**cart_method==3** MPIX_Cart_weighted_ create(…weights…)	
Base grid sizes	Nodes x CPUs x cores	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communication time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communicat.	dims_ml[d=0] [d=1] [d=2]
2 x 2 x 2	__x__x__	______ = baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _	______ = ____% of baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _	Same as with MPIX_WEIGHTS_EQUAL	
1 x 2 x 4	__x__x__	______ = baseline	Same as above	______ = ____% of baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _	______ = ____% of baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _

Reported by MPI_Cart_weighted_ create()

Exercise: Results – HLRS, Stuttgart, hazelhen

Halosize/process ~= 26 MB		MPI_Dims_create + MPI_Cart_create		MPIX_Cart_weighted_ create(MPIX_WEIGHTS_EQUAL)		MPIX_Cart_weighted_ create(…weights…)	
Base grid sizes	Nodes x CPUs x cores	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communication time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]
2 x 2 x 2	8 x 2 x 12	78.748 = baseline	8 = 8 x 1 x 1 6 = 1 x 2 x 3 4 = 1 x 1 x 4	50.971 = 65% of baseline	8 = 2 x 2 x 2 6 = 2 x 1 x 3 4 = 2 x 1 x 2	Same as with MPIX_WEIGHTS_EQUAL	
1 x 2 x 4	8 x 2 x 12	168.891 = baseline	Same as above	64.691 = 38% of baseline	8 = 2 x 2 x 2 6 = 2 x 1 x 3 4 = 2 x 1 x 2	38.406 = 23% of baseline	4 = 1 x 2 x 2 6 = 2 x 1 x 3 8 = 4 x 1 x 2

Exercise: Results – LRZ, Garching, ivyMUC

Halosize/process ~= 26 MB		MPI_Dims_create + MPI_Cart_create		MPIX_Cart_weighted_ create(MPIX_WEIGHTS_EQUAL)		MPIX_Cart_weighted_ create(…weights…)	
Base grid sizes	Nodes x CPUs x cores	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communication time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]
2 x 2 x 2	12 x 2 x 8	34.814 = baseline	8 = 6 x 1.3 x 1 6 = 1 x 1.5 x 2 4 = 1 x 1 x 4	26.675 = 77% of baseline	6 = 3 x 1 x 2 8 = 2 x 2 x 2 4 = 2 x 1 x 2	Same as with MPIX_WEIGHTS_EQUAL	
1 x 2 x 4	12 x 2 x 8	54.344 = baseline	Same as above	35.665 = 66% of baseline	6 = 3 x 1 x 2 8 = 2 x 2 x 2 4 = 2 x 1 x 2	22.933 = 42% of baseline	4 = 1 x 2 x 2 6 = 3 x 1 x 2 8 = 4 x 1 x 2

Exercise: Results – VSC, Vienna, ___(not yet done)___

Halosize/process ~= 26 MB		MPI_Dims_create + MPI_Cart_create		MPIX_Cart_weighted_create(MPIX_WEIGHTS_EQUAL)		MPIX_Cart_weighted_create(…weights…)	
Base grid sizes	Nodes x CPUs x cores	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communication time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]
2 x 2 x 2	12 x 2 x 8	________ = baseline	8 = 6 x 1.3 x 1 6 = 1 x 1.5 x 2 4 = 1 x 1 x 4	________ = _____% of baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _	Same as with MPIX_WEIGHTS_EQUAL	
1 x 2 x 4	12 x 2 x 8	________ = baseline	Same as above	________ = _____% of baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _	________ = _____% of baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _

Exercise: Your result: ____________________________

Halosize/process ~= 26 MB		MPI_Dims_create + MPI_Cart_create		MPIX_Cart_weighted_create(MPIX_WEIGHTS_EQUAL)		MPIX_Cart_weighted_create(…weights…)	
Base grid sizes	Nodes x CPUs x cores	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communication time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]	Communicat. time [ms]	dims_ml[d=0] dims_ml[d=1] dims_ml[d=2]
2 x 2 x 2	__x__x__	________ = baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _	________ = _____% of baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _	Same as with MPIX_WEIGHTS_EQUAL	
1 x 2 x 4	__x__x__	________ = baseline	Same as above	________ = _____% of baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _	________ = _____% of baseline	__ = _ x _ x _ __ = _ x _ x _ __ = _ x _ x _

Further references

Another approach using the existing MPI_Cart_create() interface:

- W. D. Gropp, Using Node [and Socket] Information to Implement MPI Cartesian Topologies, Parallel Computing, 2019, and in: Proceedings of the 25th European MPI User' Group Meeting, EuroMPI'18, ACM, New York, NY, USA, 2018, pp. 18:1-18:9. doi:10.1145/3236367.3236377.
  Slides: http://wgropp.cs.illinois.edu/bib/talks/tdata/2018/nodecart-final.pdf

And for unstructured grids:

- 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.

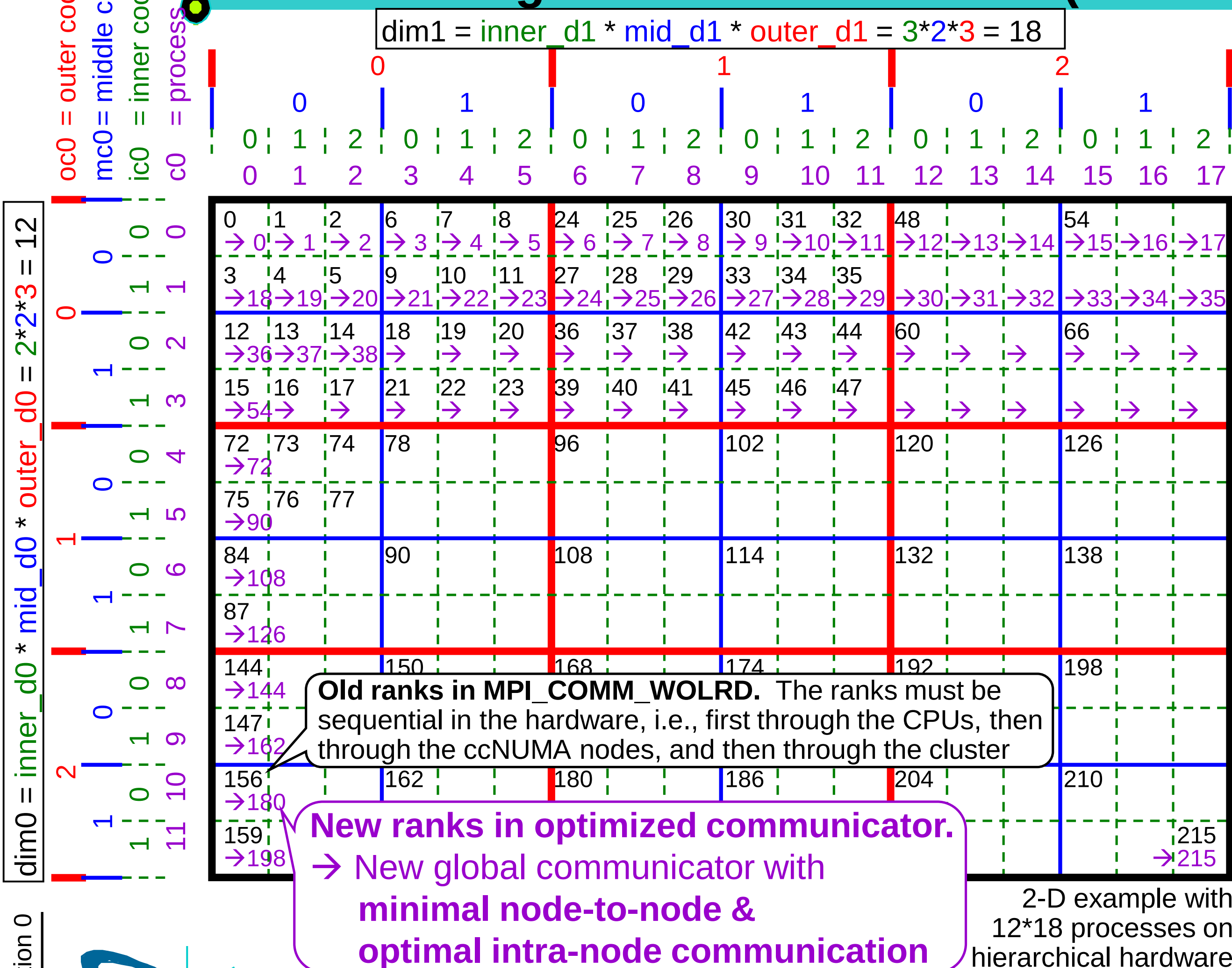
Appendix

- Method used in **MPIX_Cart_ml_create(...)**
  (only for sequentially ranked comm_old)

Implementation hints (2D & 3D) on following (skipped) slide

2-D example with 12*18 processes on hierarchical hardware with 9 nodes x 4 CPUs x 6 cores

Cartesian Grids – 2 dim – Renumbering on a cluster of SMPs (cores / CPUs / nodes)

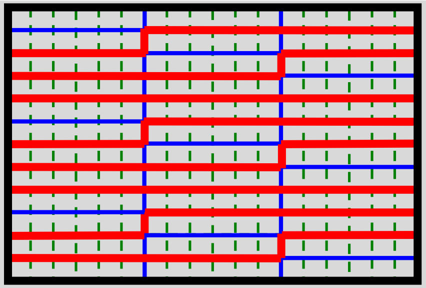


2-D example with 12*18 processes on hierarchical hardware

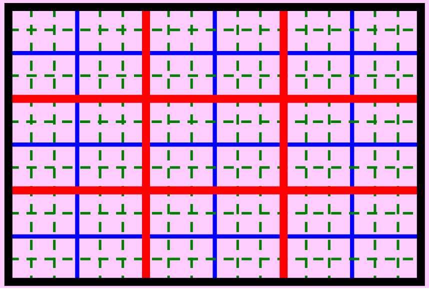
Outer = several ccNUMA nodes
Middle = CPUs of a ccNUMA node
Inner = cores of a CPU

Number of communication links:

Without re-numbering:
150 outer
72 mid
180 inner



With new ranks:
60 outer
90 mid
252 inner



Order of the **new ranks**: last coordinate is running contiguously
→ Perfect basis for MPI_Cart_create() with reorder==0 / .FALSE.

HLRS

Small 2-dim example — Renumbering on a cluster of SMPs (cores / CPUs / nodes)

2-D example with 6*4 processes
on hierarchical hardware
With 3 nodes x 2 CPUs x 4 cores

dim1 = inner_d1 * mid_d1 * outer_d1 = 2*1*3 = 6

direction 1 →

- oc1 = outer coordinate in 0..(outer_d1-1), i.e., numbering the nodes
- mc1 = middle coordinate in 0..(mid_d1-1), i.e., numbering the CPUs
- ic1 = inner coordinate in 0..(inner_d1-1), i.e., numbering the cores
- c1 = process coordinate

oc0 = outer coordinate, mc0 = middle coordinate, ic0 = inner coordinate, c0 = process coord.

dim0 = 2*2*1 = 4

direction 0 ↓

oc0	mc0	ic0	c0 \ oc1 / mc1 / ic1 / c1	0 / 0 / 0 / 0	0 / 0 / 1 / 1	1 / 0 / 0 / 2	1 / 0 / 1 / 3	2 / 0 / 0 / 4	2 / 0 / 1 / 5	
0	0	0	0	0 → 0	1 → 1	8 → 2	9 → 3	16 → 4	17 → 5	∑=15
0	0	1	1	2 → 6	3 → 7	10 → 8	11 → 9	18 →10	19 →11	51
0	1	0	2	4 →12	5 →13	12 →14	13 →15	20 →16	21 →17	87
0	1	1	3	6 →18	7 →19	14 →20	15 →21	22 →22	23 →23	123
				∑=36	40	44	48	52	56	

Old ranks in MPI_COMM_WOLRD. The ranks must be sequential in the hardware, i.e., first through the CPUs, then through the ccNUMA nodes, and then trough the cluster

New ranks in optimized communicator.

Sums of the ranks in each direction

- Six nested loops over oc0, mc0, ic0, oc1, mc1, ic1
- idim = inner_d0 * inner_d1
  mdim = mid_d0*mid_d1
- **old_rank =**
  ic1 + inner_d1*ic0
  + (mc1 + mid_d1*mc0)*idim
  + (oc1 + outer_d1*oc0)*mdim*idim
- c0=ic0+inner_d0*(mc0+mid_d0*oc0)
  c1=ic1+inner_d1*(mc1+mid_d1*oc1)
  new_rank = c1 + dim1*c0

- **ranks(new_rank) = old_rank**
  →re-numbering: MPI_Group_incl(ranks)
  →MPI_Comm_create()
- Details in 2D & 3D → **next slides**

Number of communication links with sequential ranks and → new ranks:
14 → **8** outer (btw. nodes), 8 → **6** mid (btw. CPUs), 16 → **24** inner (btw. cores)

Cartesian Grids – 3 dim –
Re-numbering on a cluster of SMPs (cores / CPUs / nodes)

This algorithm requires sequential ranking in MPI_COMM_WOLRD

Product = number of cores/CPU number of CPUs/node number of nodes

Useful and correct factorization that the sub-grid in each core is as cubic as possible

```
/*Input:*/   inner_d0=…;   mid_d0=…;   outer_d0=…;
             inner_d1=…;   mid_d1=…;   outer_d1=…;
             inner_d2=…;   mid_d2=…;   outer_d2=…;
dim0=inner_d0*mid_d0*outer_d0;
dim1=inner_d1*mid_d1*outer_d1;   dim2=inner_d2*mid_d2*outer_d2;
idim=inner_d0*inner_d1*inner_d2;
mdim=mid_d0*mid_d1*mid_d2;        odim=outer_d0*outer_d1*outer_d2;
whole_size=dim0*dim1*dim2   /* or  =idim*mdim*odim */;
ranks= malloc(whole_size*sizeof(int));
for (oc0=0; oc0<outer_d0; oc0++) /*any sequence of the nested loops works*/
 for (mc0=0; mc0<mid_d0;   mc0++)
  for (ic0=0; ic0<inner_d0; ic0++)
   for (oc1=0; oc1<outer_d1; oc1++)
    for (mc1=0; mc1<mid_d1;   mc1++)
     for (ic1=0; ic1<inner_d1; ic1++)
      for (oc2=0; oc2<outer_d2; oc2++)
       for (mc2=0; mc2<mid_d2;   mc2++)
        for (ic2=0; ic2<inner_d2; ic2++)
        { old_rank =    (ic2 + inner_d2*(ic1 + inner_d1*ic0))
                      + (mc2 +   mid_d2*(mc1 +   mid_d1*mc0))*idim
                      + (oc2 + outer_d2*(oc1 + outer_d1*oc0))*idim*mdim;
          c0 = ic0 + inner_d0*mc0 + inner_d0*mid_d0*oc0;
          c1 = ic1 + inner_d1*mc1 + inner_d1*mid_d1*oc1;
          c2 = ic2 + inner_d2*mc2 + inner_d2*mid_d2*oc2;
          new_rank = c2 + dim2*(c1 + dim1*c0);
          ranks[new_rank] = old_rank;
        }
MPI_Comm_group(MPI_COMM_WORLD, &world_group);
MPI_Group_incl(world_group, world_size, ranks, &new_group);   free(ranks);
MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);
dims[0] = dim0; dims[1] = dim1; dims[2] = dim2;
MPI_Cart_create(new_comm, 3, dims, periods, 0 /*=false*/, &comm_cart);
```

/* final output */

All renumbering is implemented with MPI_Comm_split in the provided software, see MPI-3.1, Sec. 7.5.8, page 313, lines 7-13. In MPIX_Cart_ml_create.c, an additional test-version uses this algorithm here.

For an alternative with MPI_Comm_split, see MPI-3.1, Sec. 7.5.8, page 313, lines 7-13.