Topology aware Cartesian grid mapping with MPI

Issue 120 Reading

Christoph Niethammer Rolf Rabenseifner

niethammer@hlrs.de rabenseifner@hlrs.de

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

HLRS, Stuttgart, March 02, 2019

MPI Forum March 4-7, 2019, Chattanooga

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

The Problems of MPI_Dims_create + MPI_Cart_create

- The factorization of a given amount of MPI processes must be
  - Application topology aware [1]
  - Hardware topology aware
- Current definition of MPI_Dims_create is not prepared for this
- Extreme differences in latency and accumulated bandwidth between **inter**-node and **intra**-node communication
- The reordering by MPI_Cart_create:
  - Many implementations do nothing [2]
  - A perfect reordering may require complex domain decomposition algorithms (e.g. Metis) [3]

Slides 2-3

Slides 4-6

Slide 7-10

We propose a new interface together with a fast algorithm, which is application and hardware topology aware

Slides 11-22

Implementation remarks

Slide 23-24

Benchmark + results

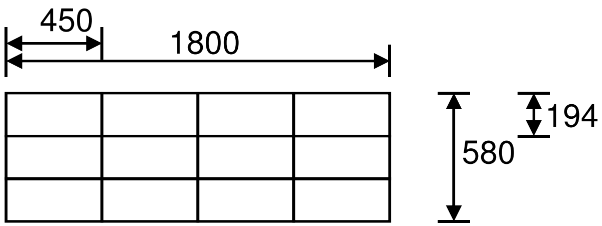
Slide 25-34

[1, 2, 3] see References on last slide

Slide 35

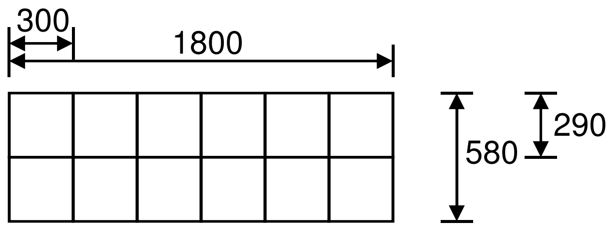
Examples

- Application topology awareness
  - 2-D example with 12 MPI processes and gridsize 1800x580
    - **MPI_Dims_create → 4x3**



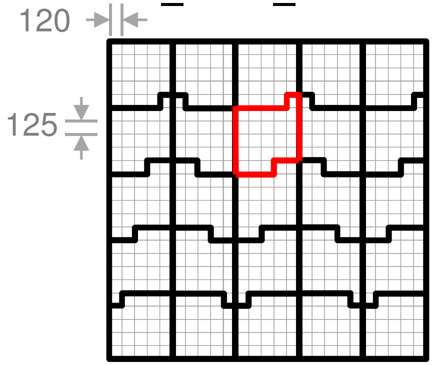
Boundary of a subdomain = 2(450+194) = **1288** ☹

    - **grid aware → 6x2 processes**



Boundary of a subdomain = 2(300+290) = **1180** ☺

- Hardware topology awareness
  - 2-D example with 25 nodes x 24 cores and gridsize 3000x3000
    - **MPI_Dims_create → 25 x 24**



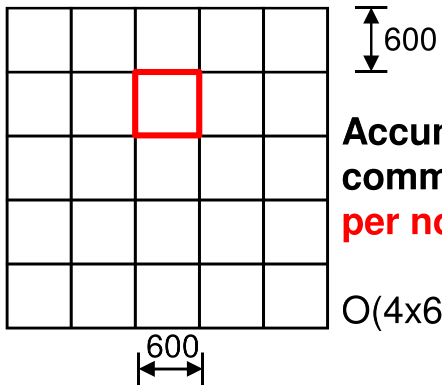
Accumulated communication per node

O(10x120+12x125)

= O(**2700**) ☹

    - **Hardware aware**

→ (5 nodes x 6 cores) X (5 nodes x 4 cores)



Accumulated communication per node

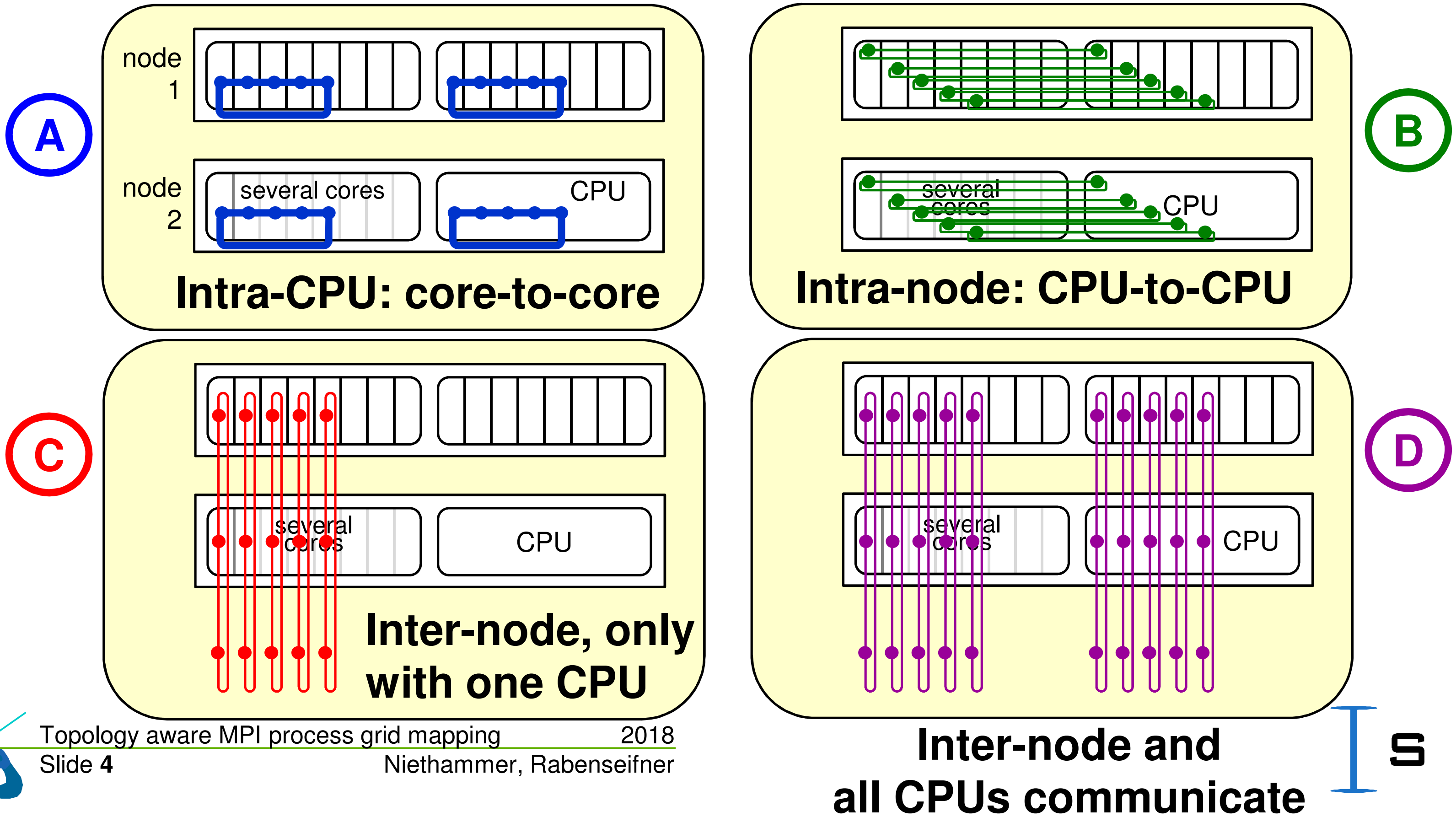
O(4x600) = O(**2400**) ☺

Ring Benchmarks for Inter- and Intra-node Communication

Benchmark halo_irecv_send_multiplelinks_toggle.c

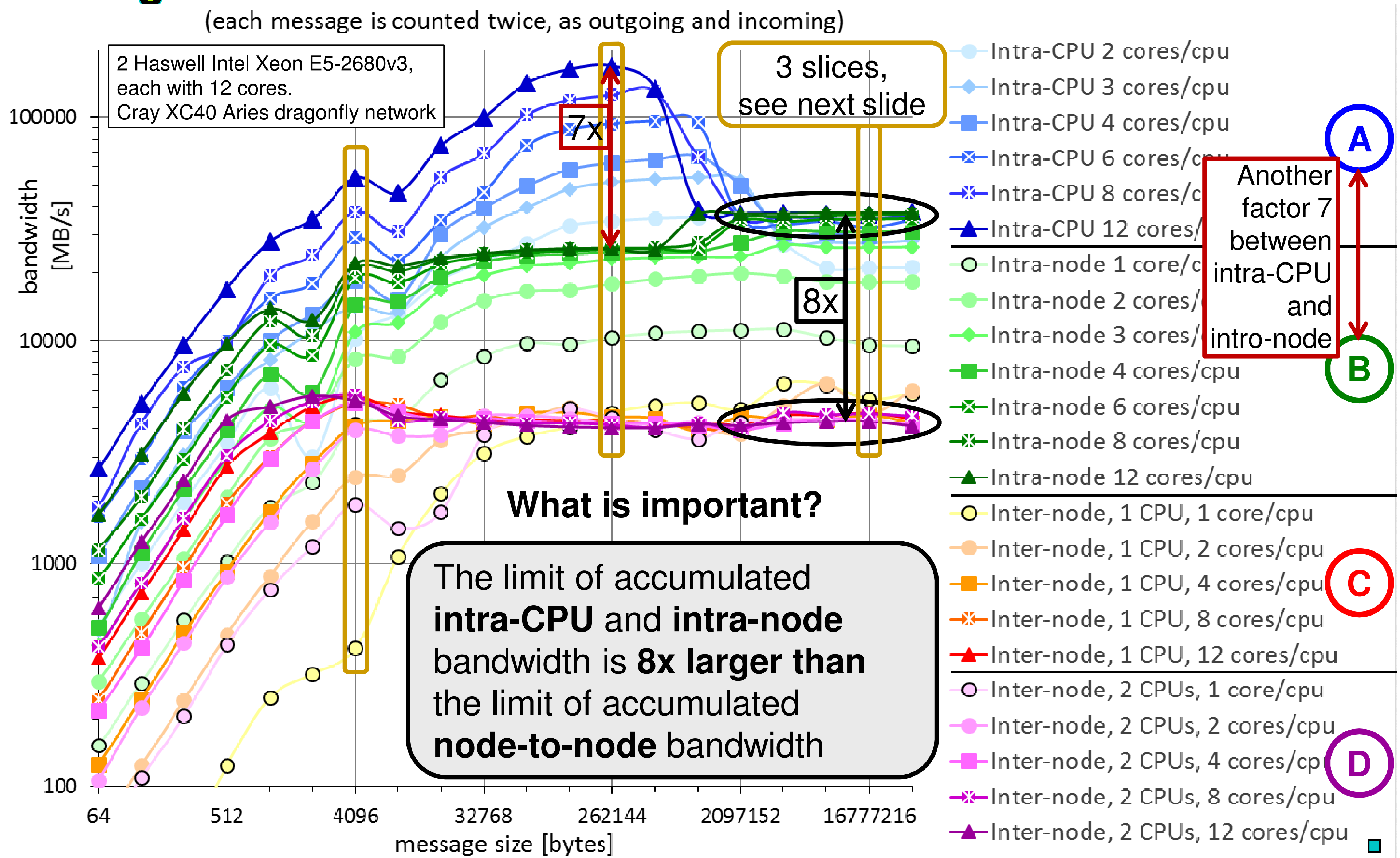
See HLRS online courses
http://www.hlrs.de/training/par-prog-ws/
→ Practical → MPI.tar.gz
→ subdirectory MPI/course/C/1sided/

- Varying message size,
- number of ***communication cores per CPU***, and
- four communication schemes (example with 5 ***communicating cores per CPU***)



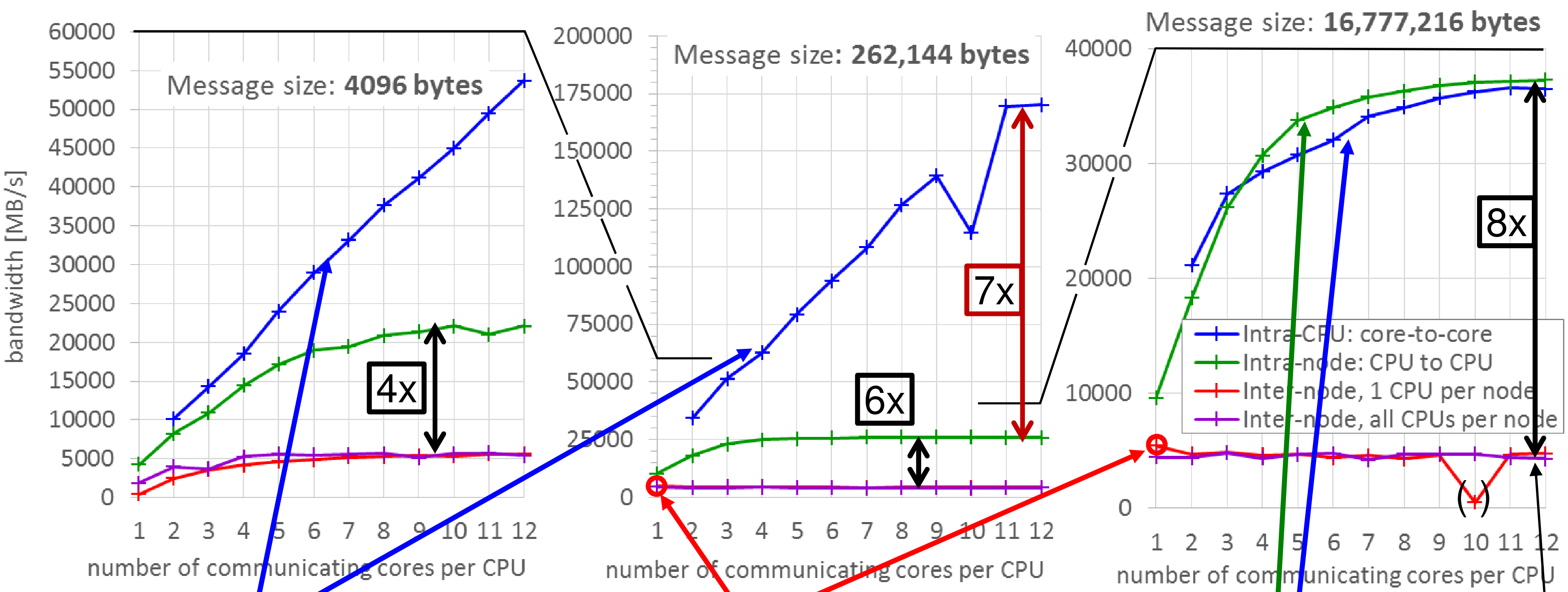
See HLRS online courses **http://www.hlrs.de/training/par-prog-ws/** → Practical → MPI.tar.gz → subdirectory MPI/course/C/1sided/

Duplex accumulated ring bandwidth per node



Measurement with halo_irecv_send_multiplelinks_toggle.c on 4 nodes of Cray XC40 hazelhen.hww.de, June 15, 2018, HLRS, by Rolf Rabenseifner (protocol 10)

Duplex accumulated ring bandwidth per node – scaling vs. asymptotic behavior



Core-to-core:
Linear scaling for small to medium size messages due to caches

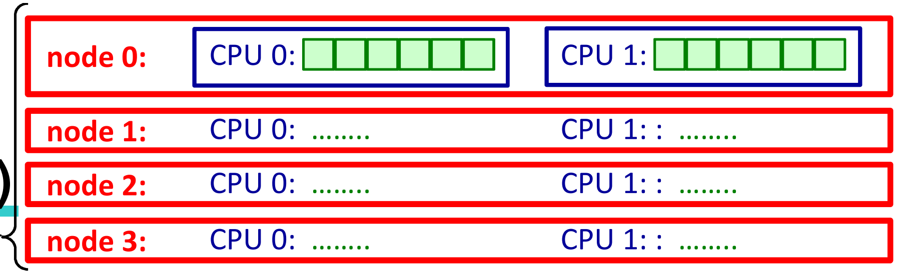
Node-to-node:
One duplex link by **one core** already fully saturates the network

Core-to-core & CPU-to-CPU:
Long messages:
Same asymptotic limit through **memory bandwidth**

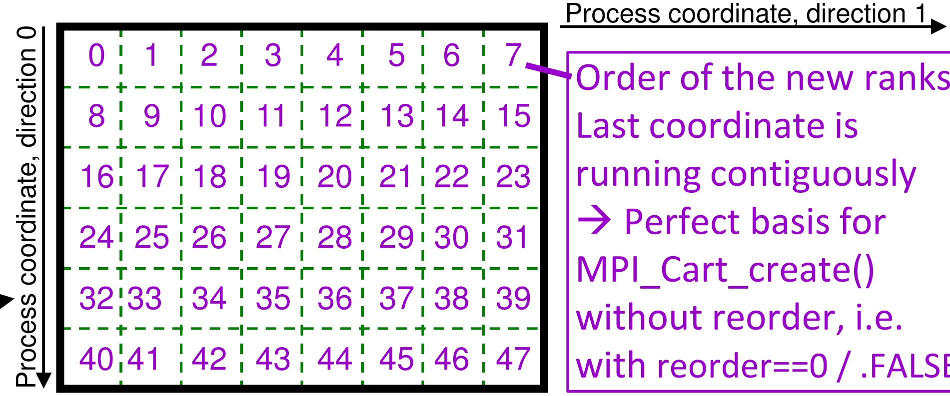
Result: The limit of accumulated **intra-CPU** and **intra-node** bandwidth is **8x larger than** the limit of accumulated **node-to-node** bandwidth

Re-numbering on a cluster of SMPs (cores / CPUs / nodes)

- Example with 48 cores on:
  - **4 ccNUMA nodes**
  - **each node with 2 CPUs**,
  - **each CPU with 6 cores**



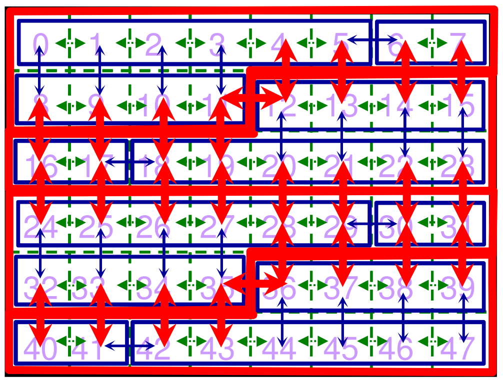
- 2-dim application with 6000 x 8080 gridpoints
  - **Minimal communication with 2-dim domain composition with 1000 x 1010 gridpoints/core (shape as quadratic as possible**
    → minimal circumference
    → minimal halo communication)
  - **virtual 2-dim process grid: 6 x 8**

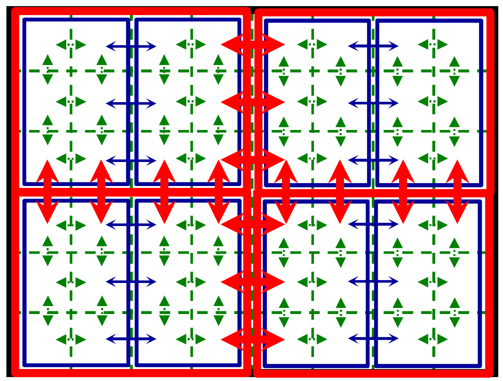


Order of the new ranks: Last coordinate is running contiguously
→ Perfect basis for MPI_Cart_create() without reorder, i.e. with reorder==0 / .FALSE.

- How to locate the MPI processes on the hardware?
  - **Using sequential ranks in MPI_COMM_WORLD**
  - **Optimized placement**

  → Proposed algorithm in slides 7-15



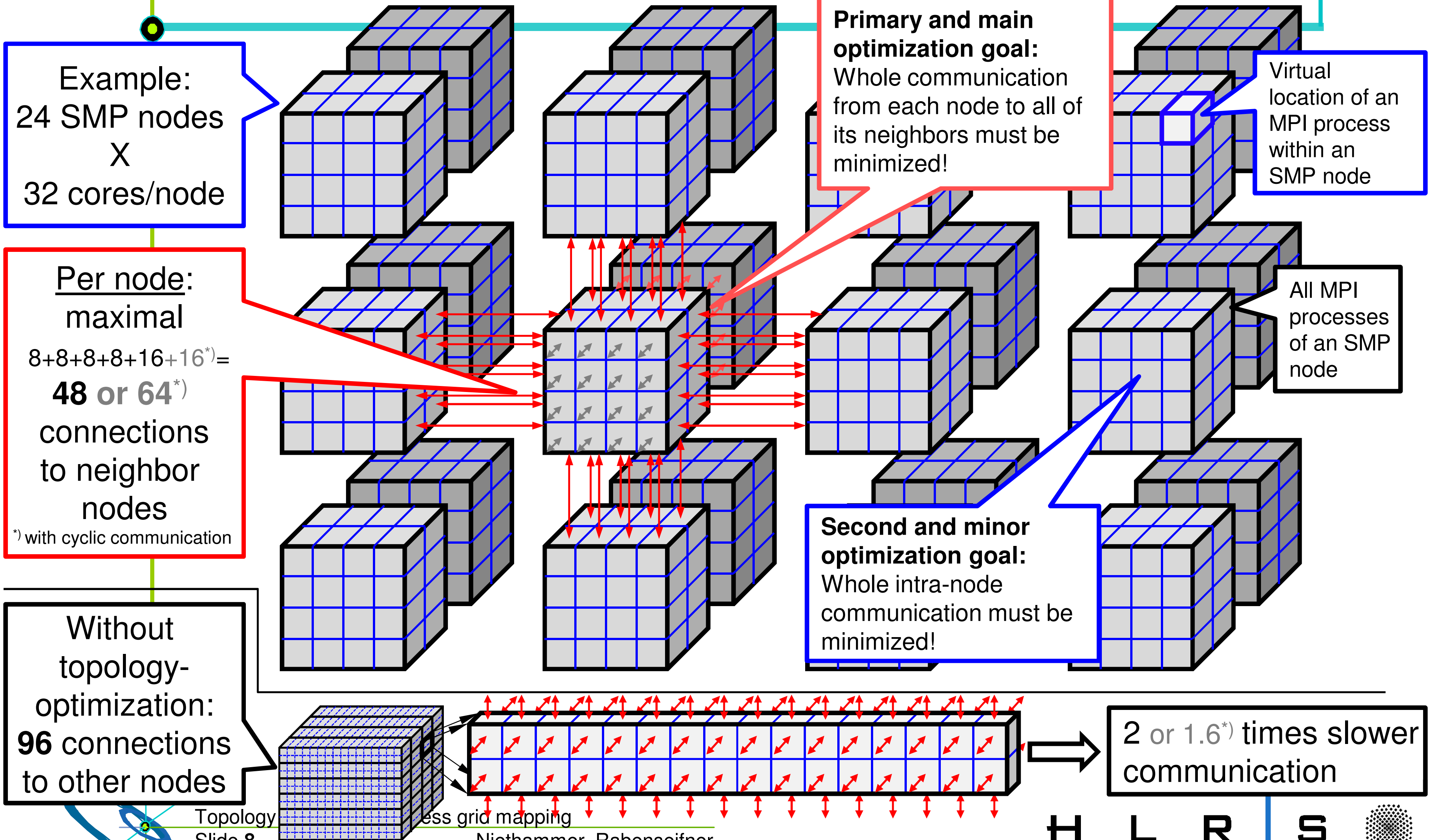


Non-optimal communications:
26 node-to-node (outer)
20 CPU-to-CPU (middle)
36 core-to-core (inner)

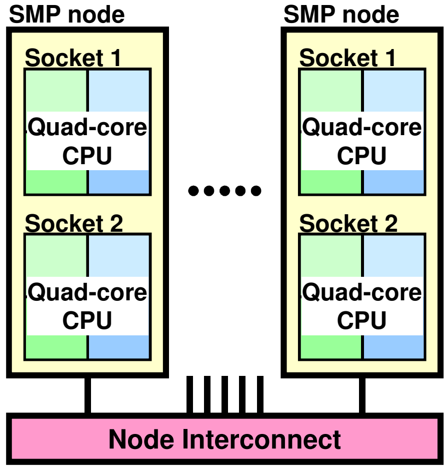
Optimized placement:
Only 14 node-to-node
Only 12 CPU-to-CPU
56 core-to-core

H L R I S

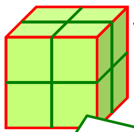
Hierarchical Cartesian Domain Decomposition



Levels of communication & data access



- Three levels:
  - Between the SMP nodes
  - Between the sockets inside of a ccNUMA SMP node
  - Between the cores of a socket
- On all levels, the communication should be minimized:
  - With 3-dimensional sub-domains:
    - **They should be as cubic as possible = minimal surface = minimal communication**



Outer surface corresponds to the data communicated to the neighbor nodes in all 6 directions → **Major optimization goal**

Inner surfaces correspond to the data communicated or accessed between the cores inside of a node → Least important

- **"as cubic as possible" may be qualified** due to different communication bandwidth in each direction caused by sending (fast) non-strided or (slow) strided data

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 dimensions on each hardware level

Base factors, not absolute values

By default, communication time strongly depends on cuboid's direction

On Cray XC40 Hazel hen at HLRS Stuttgart, Jan 2019

Optimized communication times are nearly independent of cuboid's form

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:

- **MPI_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 );
```

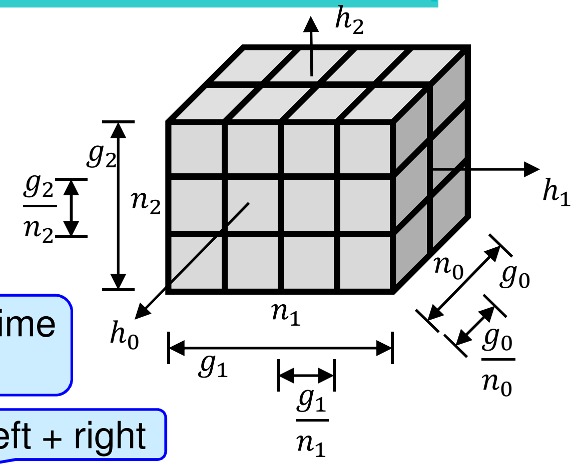
  - Arguments have same meaning as in MPI_Dims_create & MPI_Cart_create
  - See next slide for meaning of dim_weights[ndims]

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 $i = 0, d\text{-}1$ 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

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

Primarily for the node decomposition and secondarily on core level

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

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

MPIX routines, courtesy of Christoph Niethammer, HLRS

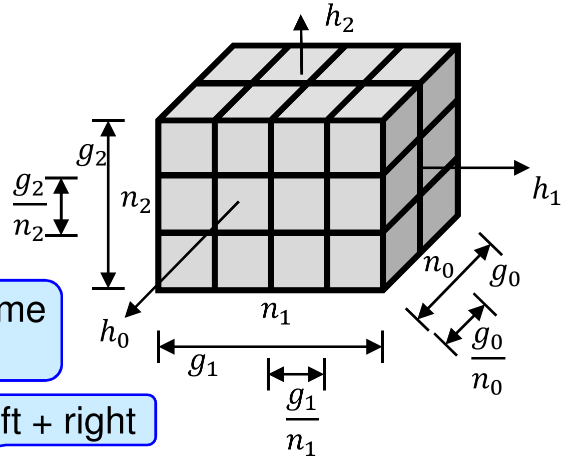
The weights w_i

Note that

- The proposal for the MPI standard does not discuss the implementation.
- It uses the same figure, but n_i expresses number of processes, i.e. dims[i], and not number of nodes in one direction, as on previous slide.
- Can be implemented with **MPI_Cart_ml_create_from_types()** → next slide

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

MPIX routines, courtesy of Christoph Niethammer, HLRS

Further Interfaces (1)

If the application wants to choose the hardware levels,
then the most simple interface is to choose a split type for each level
(except for the last one):

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., nsplit_types=2, split_types=
{ MPI_COMM_TYPE_SHARED,
OMPI_COMM_TYPE_NUMA
within OpenMPI, or for further
splitting: MPIX_COMM_TYPE_
HW_TOPOLOGY}

```
MPI_Cart_ml_create_from_types(MPI_Comm comm_old,
             int nsplit_types,      int split_types[nsplit_types],
             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**

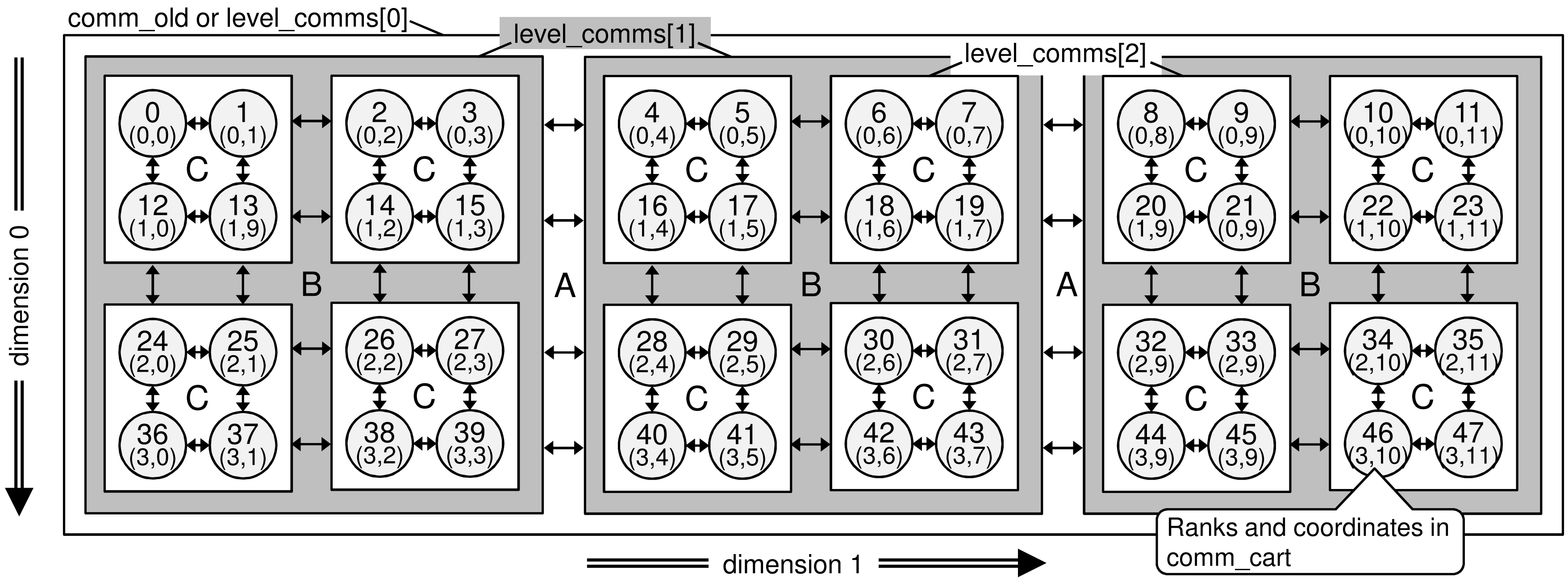
The Cartesian communicator reflects this result: **30 x 25 x 20**

Note that MPI_Dims_create() would produce
25 x 25 x 24
which would never fit to the needed node-level distribution
5 x 25 x 5

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

2-D Cartesian Example (proposed for the MPI standard)

with
- 48 processes and ndims=2,
- dim_weights=$({}^{1.0}/_{4},\ {}^{1.0}/_{12})$, and
- nsplit_types=2 (e.g. splitting into ccNUMA nodes and those into CPUs)
- or appropriate level_comms[0] .. [2]



The optimization chooses an appropriate re-ordering of the ranks of comm_cart that
1st communication A (e.g., node to node) is minimized,
2nd B (e.g., CPU to CPU) is minimized, and 3rd C (e.g., core to core) is minimized

Further Interfaces (2)

If the application wants to choose the hardware levels,
but appropriate split types (as for MPI_COMM_SPLIT_TYPE() do not exist,
then the splitting can be done by the application
and an array with the hierarchical set of communicators is the input:

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., nlevels=3, and
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 previous slide.

```
MPI_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 );
```

Same as above

Same as above

Further Interfaces (3) – the basis

MPI_Cart_ml_create_from_comms(… level_comms, …)
can be implemented based on **MPI_Dims_ml_create(),**
which can be implemented through **MPI_Dims_weighted_create**()
provided that the **level_comms** have same size on each level

MPI_Dims_weighted_create (int nnodes, int ndims, double dim_weights[ndims],
int periods[ndims], MPI_Info info, /***INOUT***/ *int dims[ndims]*);

MPI_Dims_ml_create (int nnodes, int ndims, double dim_weights[ndims],
int nlevels, int sizes[nlevels], int periods[ndims], MPI_Info info,
/***OUT***/ *int dims_ml[ndims][nlevels]*);

Multi-level info

Further Interfaces (3) – the basis (a)

MPI_Dims_weighted_create (int nnodes, int ndims, double dim_weights[ndims], int periods[ndims], MPI_Info info, /***INOUT***/ *int dims[ndims]*);

The factorization of nnodes into the array dims is chosen in a way to minimize the communication according to the given weights. Note that this means that this function looks for a factorization with minimal sum $\sum_i$ *dims[i]* · *dims_weight[i]*.

Rationale. As shown in Figure 7.3, the total communication cost per node (left and right, in all dimensions) would be $C = 2\sum_i w_i / \prod_{j,j\neq i} d_j$ and with nnodes=$N = \prod_j d_j$, the cost is $C = \frac{2}{N}\sum_i w_i d_i$ (End of rationale.)

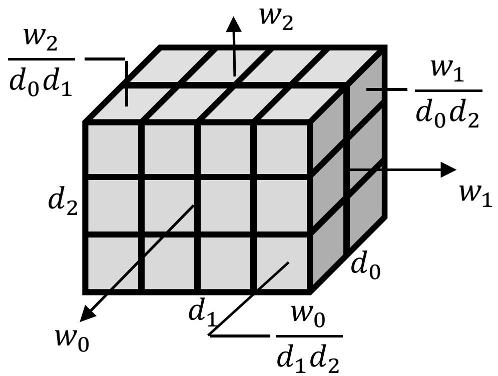


Figure 7.3: Communication model:
The dim_weights[i]=w_i express the total communication cost in the direction of dimension i. The communication cost of one process in direction i is therefore $w_i / \prod_{j,j\neq i} d_j$ with dims[i]=d_i.

Further Interfaces (3) – the basis (b)

MPI_Dims_ml_create (int nnodes, int ndims, double dim_weights[ndims],
int nlevels, int sizes[nlevels], int periods[ndims], MPI_Info info,
/***OUT***/ *int *dims_ml[ndims][nlevels]*);

It uses internally MPI_Dims_weighted_create() for each level → next slide

int *dims_ml

- represents in C an C++ a contigous two dimensional array *dims_ml[ndims][nlevels]*
- and *dims_ml(ndims,nlevels)* in Fortran

Further Interfaces Summary of (1-3)

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 further
splitting: MPIX_COMM_TYPE_
HW_TOPOLOGY}

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*);

MPI_Cart_ml_create_from_types(MPI_Comm comm_old,
int nsplit_types, int split_types[nsplit_types],
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*)

MPI_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

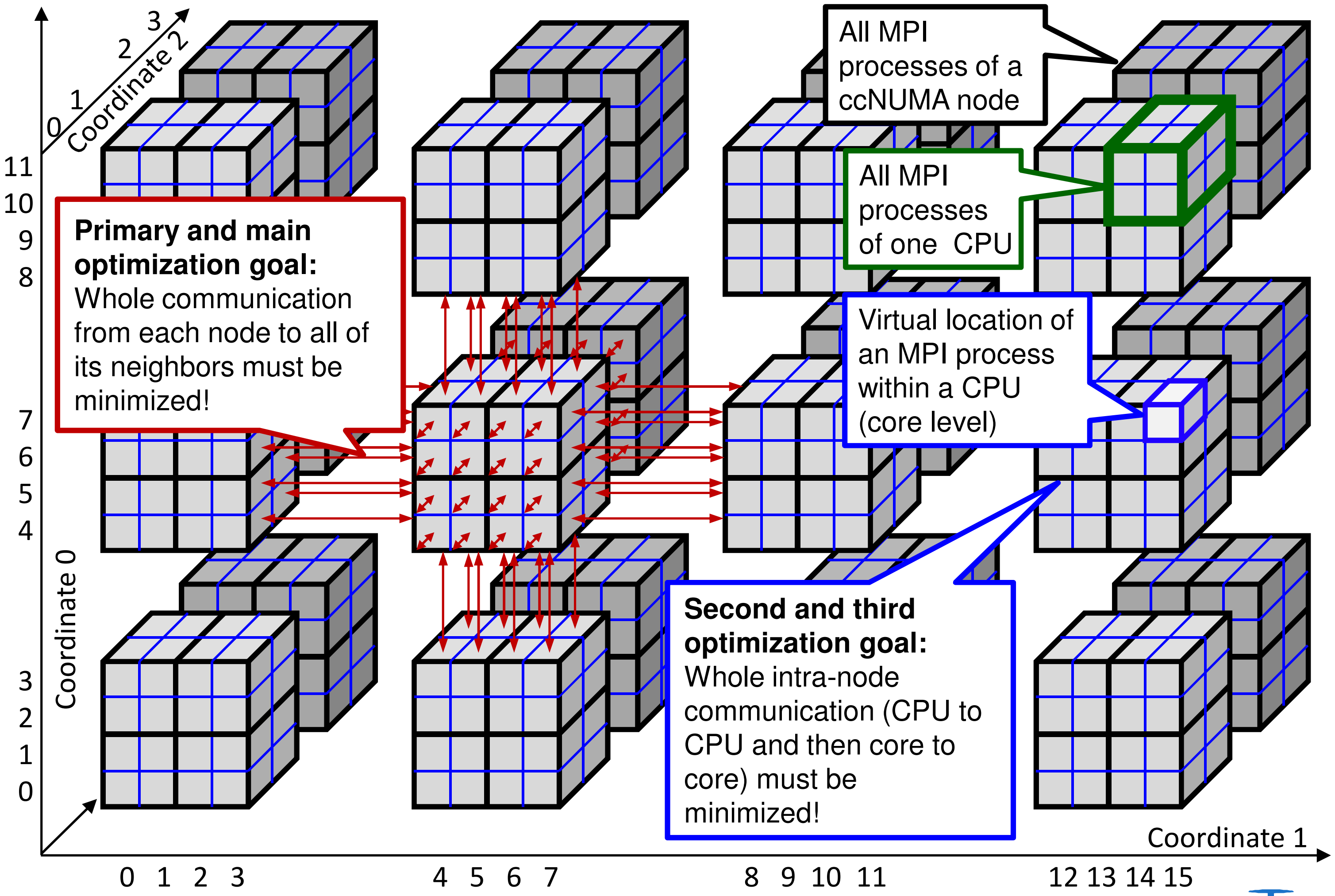
MPI_Dims_weighted_create (int nnodes, int ndims, double dim_weights[ndims],
int periods[ndims], MPI_Info info, /***INOUT***/ *int dims[ndims]*);

MPI_Dims_ml_create (int nnodes, int ndims, double dim_weights[ndims],
int nlevels, int sizes[nlevels], int periods[ndims], MPI_Info info,
/***OUT***/ *int dims_ml[ndims][nlevels]*);

Multi-level info

Hierarchical Cartesian Domain Decomposition

(figure used in the proposal for the MPI standard)



Further Interfaces (4)

From previous slides

MPI_Dims_ml_create (int nnodes, int ndims, double dim_weights[ndims],
int nlevels, int sizes[nlevels], int periods[ndims], MPI_Info info,
/***OUT***/ *int dims_ml[ndims][nlevels]*);

MPI_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**

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*);

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.

Implementation 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 <u>**sequential** ranking </u>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 dimensions on each hardware level

Base factors, not absolute values

By default, communication time strongly depends on cuboid's direction

On Cray XC40 Hazel hen at HLRS Stuttgart, Jan 2019

Optimized communication times are nearly independent of cubid's form

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)

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)

```
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
```

divide by dims[i]

First value for our table

* 2 directions * 4 byte

multiply for 2-d halo array size

```
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
```

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

Second value for our table

…cess grid mappi…
Niethamm…

H L R I S

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_ml_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

Block length + stride for each dimension

Base gridsizes

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

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

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

		cart_method==1		**cart_method==2**		**cart_method==3**	
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.	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 _

References

[1] Pavan Balaji et al. 2009-2012. Topology awareness in MPI Dims create. https://github.com/mpi-forum/mpi-forumhistoric/issues/195 Accessed 2018-07-19.

Another approach using the existing MPI_Cart_create() interface:

[2] William 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:

[3] 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.

Used as theoretical basis for implementing MPI_Dims_weighted_create():

[4] 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.