# Topology aware Cartesian grid mapping with MPI

Christoph Niethammer Rolf Rabenseifner

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

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

**HLRS, Stuttgart, Sep 25, 2018**

Slides for the poster at EuroMPI 2018

For further information, see 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

Slides 1+2

- Extreme differences in latency and accumulated bandwidth between **inter**-node and **intra**-node communication

Slides 3-5

- The reordering by MPI_Cart_create:
  - Many implementations do nothing
  - A perfect reordering may require complex domain decomposition algorithms (e.g. Metis) [3]

Slide 6

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

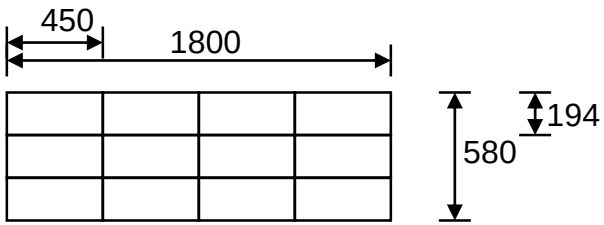
Slides 7-17

[1, 3] see References on last slide

Slide 19

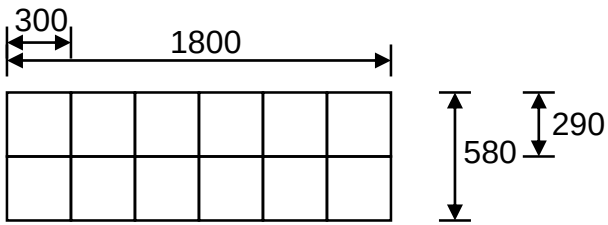
# Examples

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



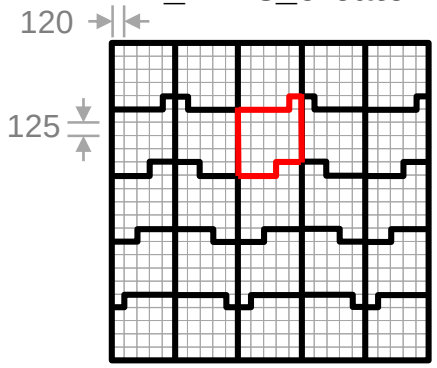

Boundary of a subdomain = 2(450+194) = **1288** ☹

    - **grid aware → 6x2 processes**




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

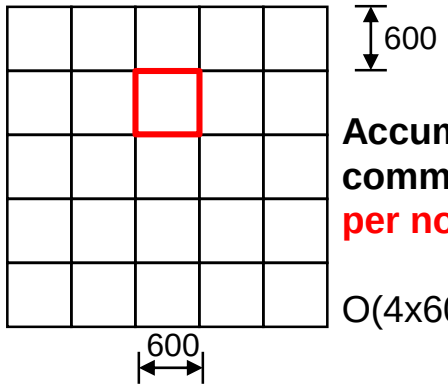
- Hardware topology awareness
  - 2-D example with 25 nodes x 24 cores and gridsize 3000x3000
    - **MPI_Dims_create → 25 x 24**




**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**

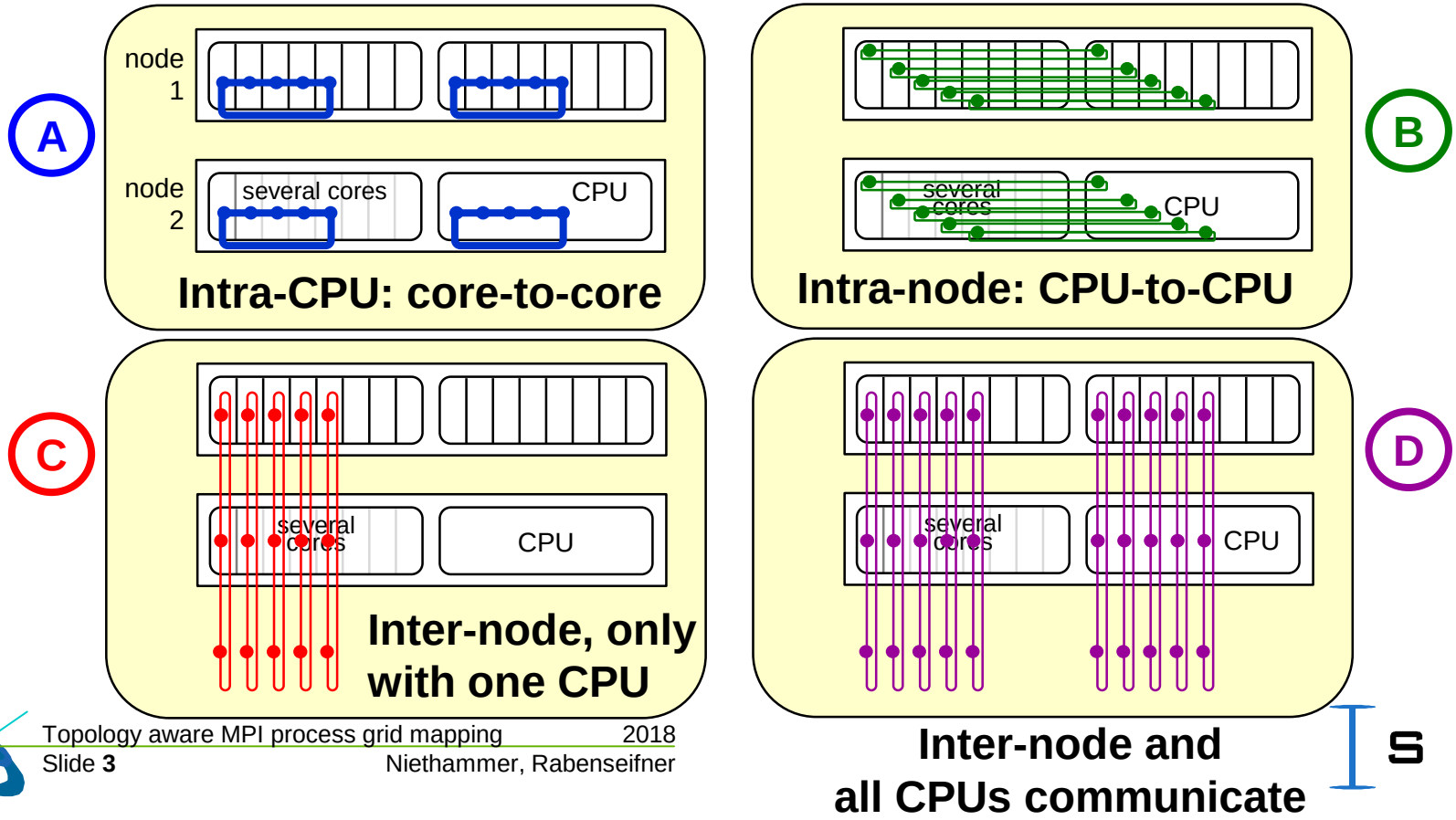
O(4x600) = O(**2400**) ☺

# Ring Benchmarks for Inter- and Intra-node Communication

Benchmark halo_irecv_send_multiplelinks_toggle.c

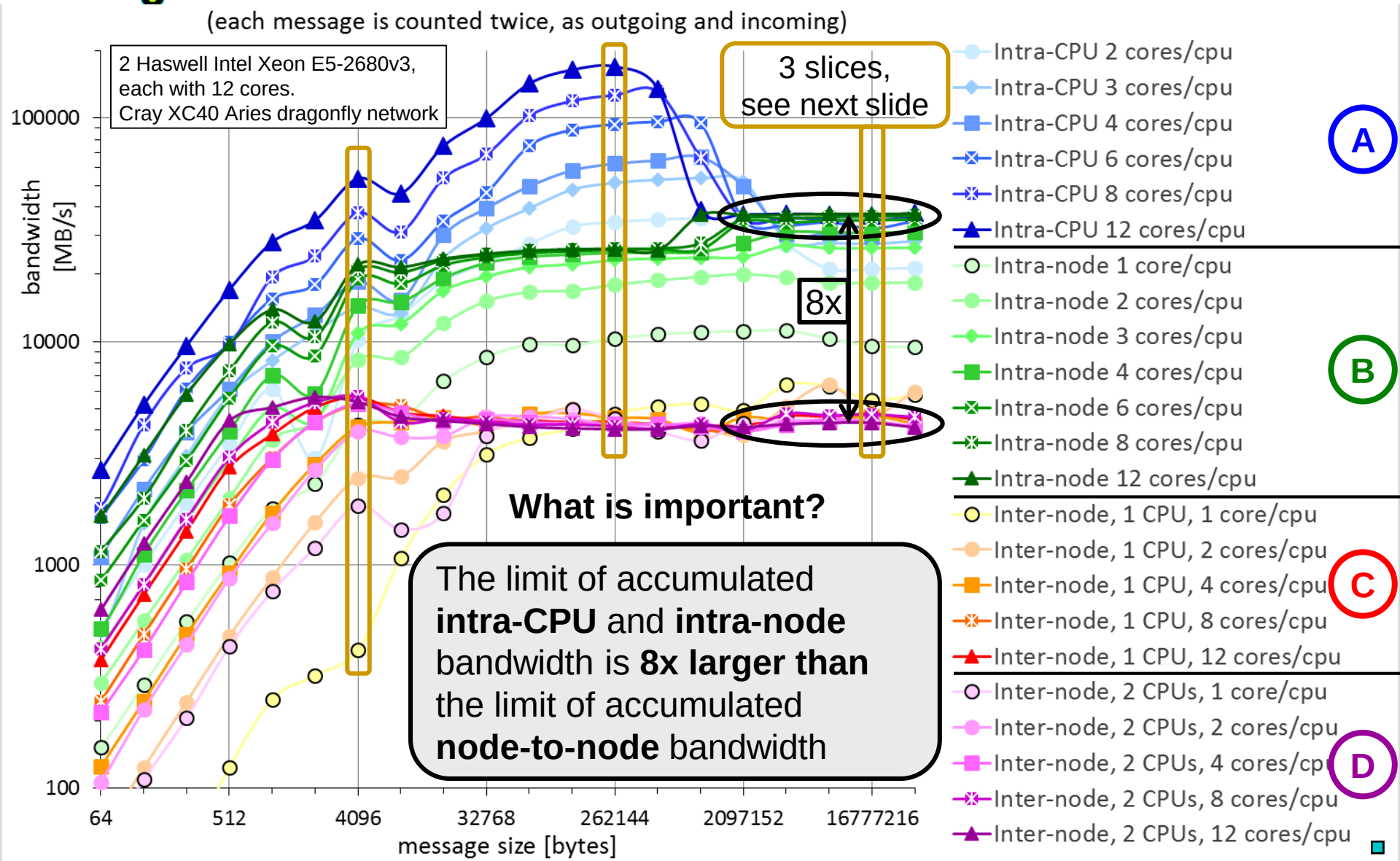
See HLRS online courses
**http://www.hlrs.de/training/par-prog-ws/**
→ Practical → MPI.tar.gz
→ subdirectory MPI/course/C/1sided/

- 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

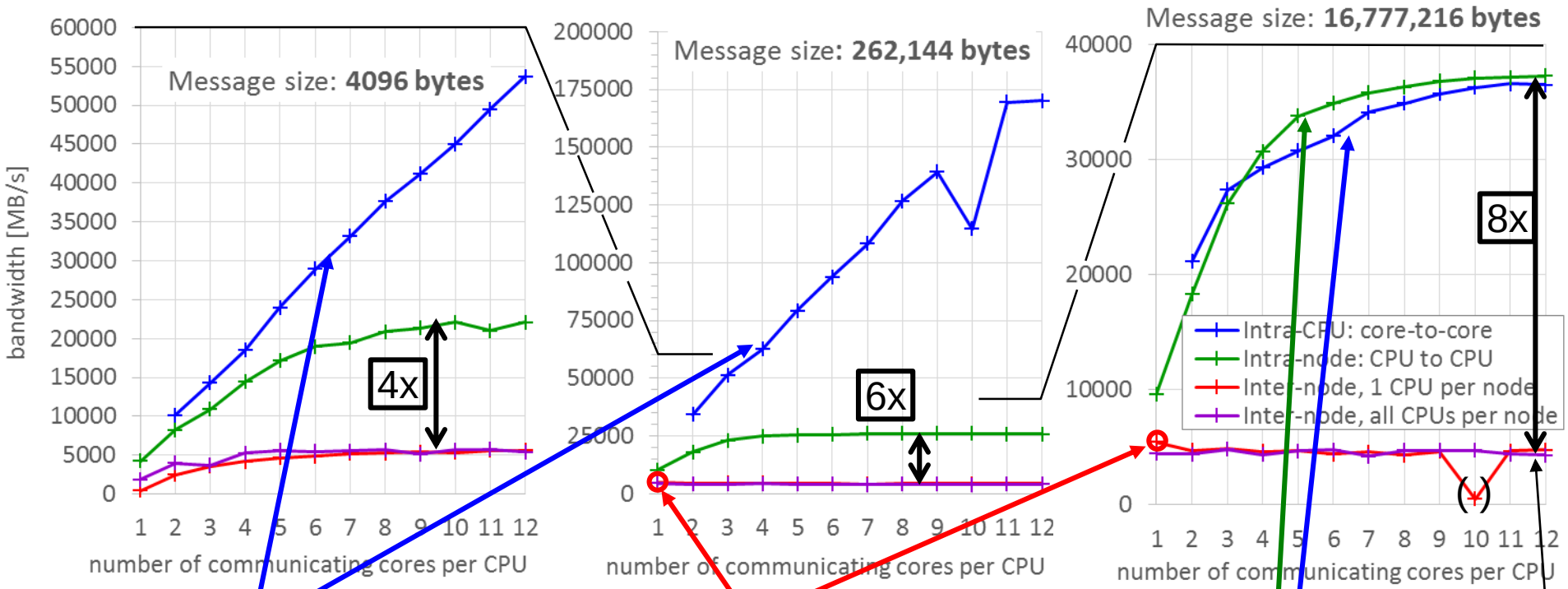



**What is important?**

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

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

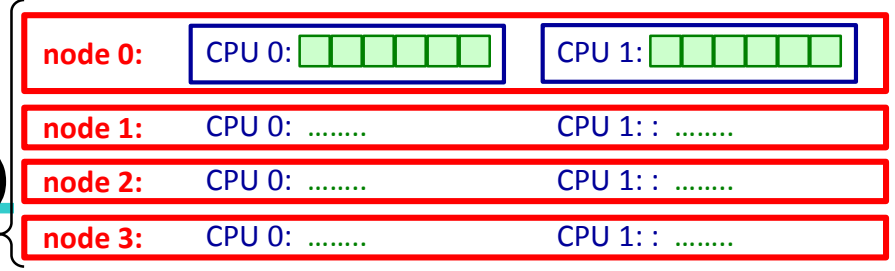
Node-to-node:
One duplex link by **one core** already fully saturates the network

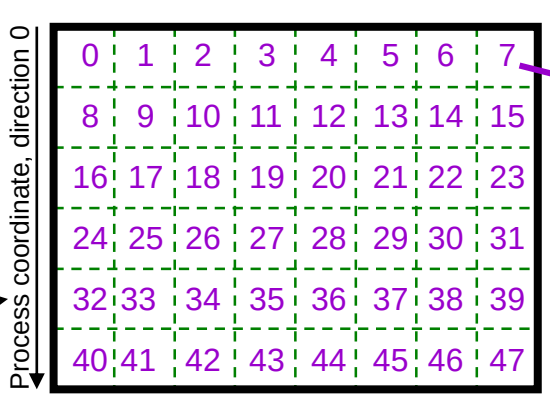
Core-to-core & CPU-to-CPU:
**Long messages**:
Same asymptotic limit through **memory bandwidth**

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

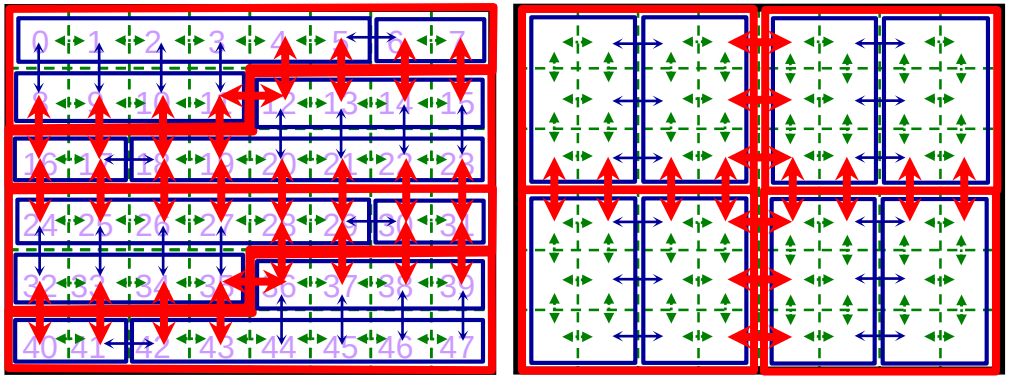
# Re-numbering on a cluster of SMPs (cores / CPUs / nodes)

- 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**
- How to locate the MPI processes on the hardware?
  - **Using sequential ranks in MPI_COMM_WORLD**
  - **Optimized placement**
  - **→ Proposed algorithm in slides 7-15**




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



**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

# Proposed Mapping Algorithm

- To keep the algorithm small (and fast):
  - Use multi-level Cartesian subdomains
- Based on the benchmark results:
  - The first and major optimization goal is minimizing the **inter**-node communication volume
  - Using the algorithm from [4] for the multi-dimensional factorization of the number of nodes, but with a modified optimization goal that is application topology aware
- Using the same principles for each further hardware level to minimize
  - intra-node (i.e., CPU-to-CPU) communication
  - intra-CPU (i.e., core-to-core) communication

# The Optimization Algorithm – First level

**Given:** $\boldsymbol{d}$-dimensional Cartesian grid with a total of $T = \prod_{i=1}^{d} \boldsymbol{t_i}$ elements

Level 1 (= outer level = node level) on $\boldsymbol{N}$ nodes:

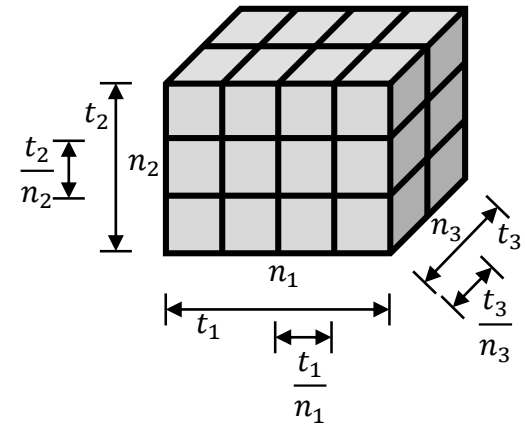
The total grid divided into subdomains, one on each node

- Factorization of $N$ into factors $(n_i)_{i=1,d}$ with $N = \prod_{i=1}^{d} n_i$
- Communication cost in both directions of each dimension (example for $d = 3$):

$$2\frac{t_2 t_3}{n_2 n_3}, \quad 2\frac{t_1 t_3}{n_1 n_3}, \quad 2\frac{t_1 t_2}{n_1 n_2}$$

- Minimizing the communication costs $c$

$$c^{(level=1)} = 2\sum_{i=1}^{d} \prod_{\substack{j=1 \\ j\neq i}}^{d} \frac{t_j}{n_j} = 2\frac{T}{N}\sum_{i=1}^{d} \frac{n_i}{t_i}$$




**Summary of Level 1:** One must search factors $(n_i)_{i=1,d}$

- that factorize $N$ with $N = \prod_{i=1}^{d} n_i$
- and minimize the term $\sum_{i=1}^{d} \frac{n_i}{t_i}$

# Second Level Optimization

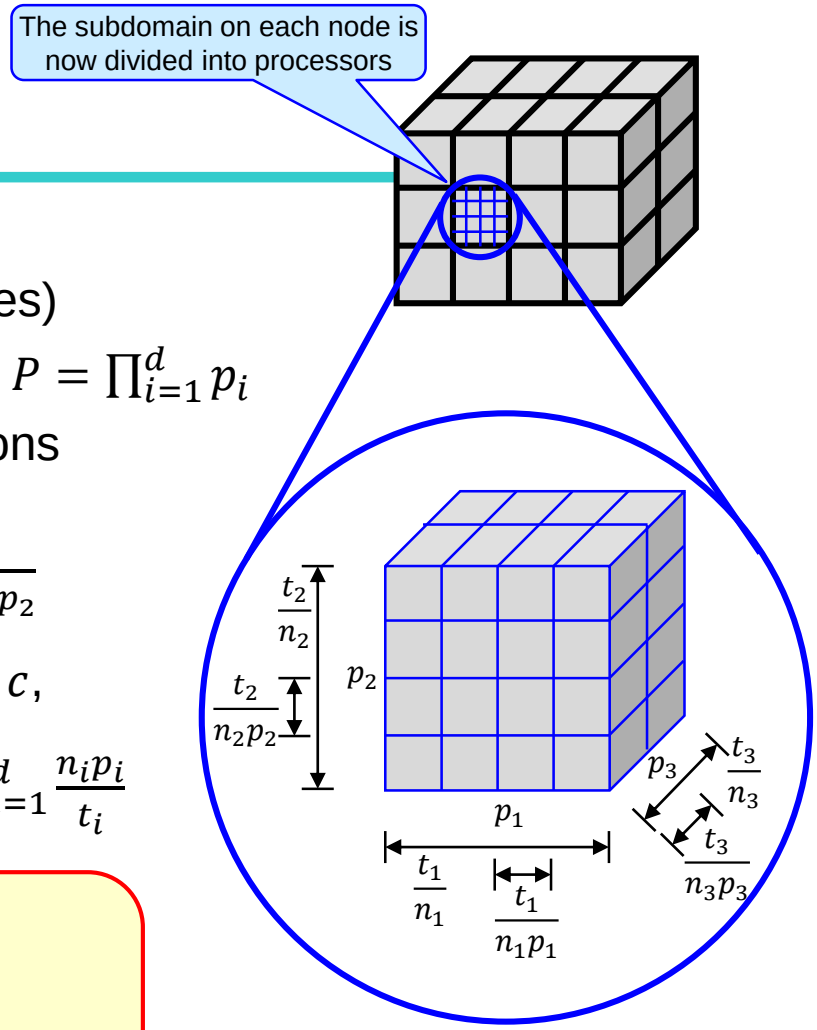
Level 2:

- Each node has $\boldsymbol{P}$ processors (or cores)
- Factorization of $P$ into factors $p_i$ with $P = \prod_{i=1}^{d} p_i$
- Communication costs in both directions of each dimension:

$$2\frac{t_2 t_3}{n_2 p_2 n_3 p_3}\,,\quad 2\frac{t_1 t_3}{n_1 p_1 n_3 p_3}\,,\quad 2\frac{t_1 t_2}{n_1 p_1 n_2 p_2}$$

- Minimizing the communication costs $c$,

$$c^{(level=2)} = 2\sum_{i=1}^{d}\prod_{\substack{j=1\\ j\neq i}}^{d}\frac{t_j}{n_j p_j} = 2\frac{T}{NP}\sum_{i=1}^{d}\frac{n_i p_i}{t_i}$$

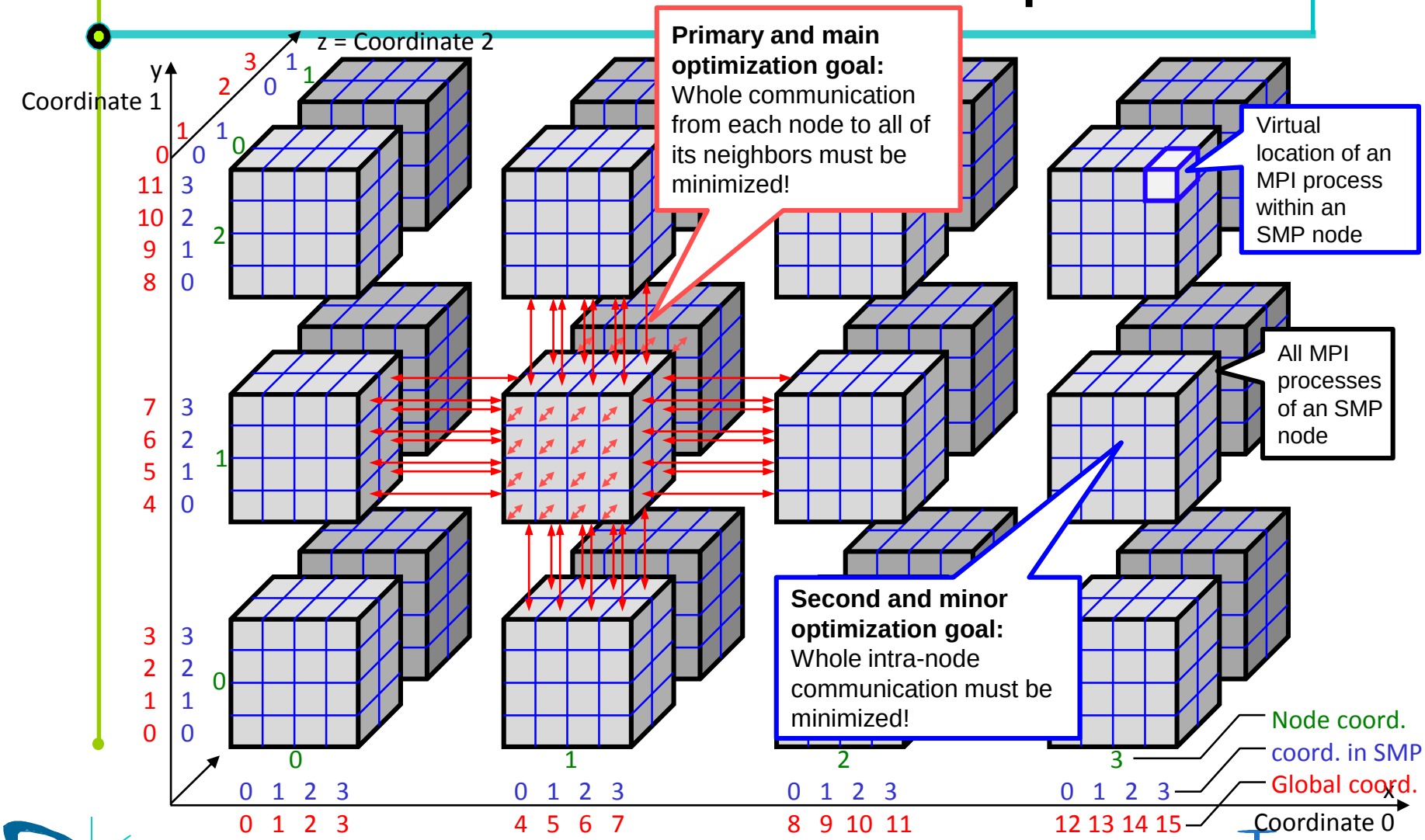



**Summary of Level 2:**

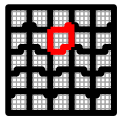
One must search factors $(p_i)_{i=1,d}$

- that factorize $P$ with $P = \prod_{i=1}^{d} p_i$
- and minimize the term $\sum_{i=1}^{d}\frac{n_i p_i}{t_i}$

# Hierarchical Cartesian Domain Decomposition



# Example

- **Given:** $\boldsymbol{d}$=3 dimensions, $\boldsymbol{N}$=625 nodes, $\boldsymbol{P}$=24 cores, and all $\boldsymbol{t_i}$ are identical
- → in all formulas, the $1/t_i$ can be ignored
- Level 1:
  - Search $(n_i)_{i=1,3}$ that $n_1 n_2 n_3 = 625$ and $\sum_{i=1}^{3} n_i$ minimal
  - Result: $(n_i)_{i=1,3} = (25,5,5)$ with $\sum_{i=1}^{3} n_i = 35$
- Level 2:
  - Search $(p_i)_{i=1,3}$ that $p_1 p_2 p_3 = 24$ and $\sum_{i=1}^{3} n_i p_i$ minimal
  - Result: $(p_i)_{i=1,3} = (1,6,4)$ with $\sum_{i=1}^{3} n_i p_i = 25 + 30 + 20 = 75$
- Optimized result:
  - with $(nodes\ x\ cores)$ in each dimension: (25$x$1) X (5$x$6) X (5$x$4)
  - Total core numbers as used for MPI_Cart_create: 25 X 30 X 20
- Comparison with existing MPI_Dims_create:
  - MPI_Dims_create would result in 25 x 25 x 24
  - → complex mapping to the hardware is needed, see also slide 2  ,
    or no reordering is done → (25$x$1) X (25$x$1) X (1$x$24)
    with $\sum_{i=1}^{3} n_i = 51$ ☹ (instead of 35 ☺), → **46%** more **inter-node** communication!

# Generalized multi-level optimization

**Given:** $\boldsymbol{d}$-dimensional Cartesian grid with a total of $T = \prod_{i=1}^{d} \boldsymbol{t_i}$ elements

Number of hardware levels $\boldsymbol{L}$

and for each level the number of processors $\boldsymbol{N^{(l)}}$, $l = 1, L$

Communication costs on each level $l$:

$$c^{(l)} = 2\frac{T}{\prod_{k=1}^{l} N^{(k)}} \sum_{i=1}^{d} \frac{\prod_{k=1}^{l} n_i^{(k)}}{t_i} \qquad \text{for a factorization } N^{(l)} = \prod_{i=1}^{d} n_i^{(l)}$$

Search:

- On each level $l = 1..L$, one must search a factorization $\left(n_i^{(l)}\right)_{i=1,d}$ of $N^{(l)}$
- with $N^{(l)} = \prod_{i=1}^{d} n_i^{(l)}$
- and minimal sum $\sum_{i=1}^{d} \frac{\prod_{k=1}^{l} n_i^{(k)}}{t_i}$ ,

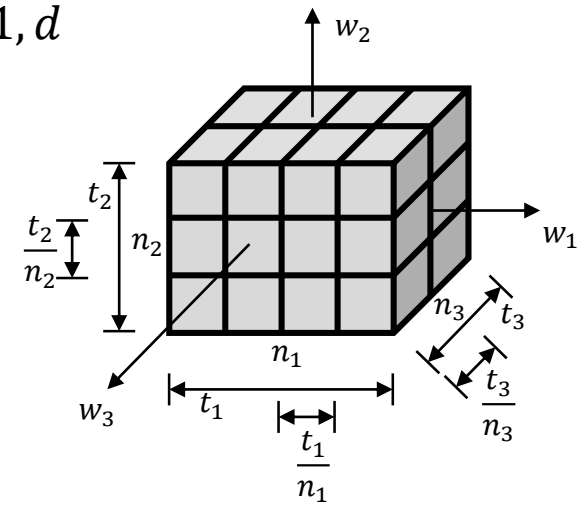
  i.e., with minimal sum $\sum_{i=1}^{d} a_i^{(l)} n_i^{(l)}$ with $a_i^{(l)} = \frac{\prod_{k=1}^{l-1} n_i^{(k)}}{t_i}$

# Generalization with weighted communication in each direction, e.g. by different halo width

- If the communication cost in each direction $i = 1, d$ is multiplied with a halo width $\boldsymbol{w_i}$, e.g., $2\frac{t_2 t_3}{n_2 n_3} w_1$
- On each $l = 1..L$, the sum to be minimized is

$$\sum_{i=1}^{d} \frac{\prod_{k=1}^{l} n_i^{(k)}}{(t_i/w_i)}$$

- i.e., $\sum_{i=1}^{d} a_i^{(l)} n_i^{(l)}$ with $a_i^{(l)} = \frac{\prod_{k=1}^{l-1} n_i^{(k)}}{(t_i/w_i)}$



# Using weighted MPI_Dims_create for application topology awareness

New MPI_Dims_weighted_create

- Task: In general, on each hardware topology level $l$ with $l = 1, \boldsymbol{L}$ and for given $\boldsymbol{N}^{(\boldsymbol{l})}$, find factorizations $\left(n_i^{(l)}\right)_{i=1,d}$ with $N^{(l)} = \prod_{i=1}^{d} n_i^{(l)}$ **and** the following sum is minimal: $\sum_{i=1}^{d} a_i^{(l)} n_i^{(l)}$ with $a_i^{(l)} = \frac{\prod_{k=1}^{l-1} n_i^{(k)}}{(t_i/w_i)}$
- Algorithm on each level $l = 1..L$
  - Sorting indexes $\boldsymbol{i} = \mathbf{1}..\boldsymbol{d}$ so that the $\left(a_{i'}\right)_{i'=\mathbf{1}..\boldsymbol{d}}$ to be non-decreasing
  - Calculate all divisors of $\boldsymbol{N}^{(\boldsymbol{l})}$
  - Loop over all non-increasing factorizations and find optimum according to
    - **1st criterion: a factorization is better if $\sum_{i=1}^{d} a_i^{(l)} n_i^{(l)}$ is smaller**
    - **2nd criterion: if the $\sum_{i=1}^{d} a_i^{(l)} n_i^{(l)}$ is the same then a factorization is better if $\Delta = n_{\mathbf{1}}^{(l)} - n_{\boldsymbol{d}}^{(l)}$ is smaller**
    - **3rd criterion: if $\sum_{i=1}^{d} n_i$ and $\Delta = n_{\mathbf{1}}^{(l)} - n_{\boldsymbol{d}}^{(l)}$ are the same then a factorization is better if $n_{\mathbf{1}}^{(l)}$ is smaller**
  - Revert the index mapping $\left(n_{i'}^{(l)}\right)_{i'=\mathbf{1}..\boldsymbol{d}} \rightarrow \left(n_i^{(l)}\right)_{\boldsymbol{i}=\mathbf{1}..\boldsymbol{d}}$

i.e. with $n_i \geq n_{i+1}$

Additional tricks:
- Calculate divisors only upto sqrt(N), calculate the rest by reciprocal values.
- Loop over divisors from highest to smallest, recursively over $i = 1, d$,
- Start value for $n_{i+1}$ is next real divisor equal or smaller then $n_i$

Re-mapping of all process ranks according to $\left(n_i^{(l)}\right)_{\boldsymbol{i}=\mathbf{1}..\boldsymbol{d},\boldsymbol{l}=\mathbf{1},\boldsymbol{L}}$ and creation of the new Cartesian communicator according to algorithm in [2]

# Proposed Interfaces

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

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 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.,
{ MPI_COMM_TYPE_SHARED,
MPI_COMM_TYPE_NUMA }

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],
***/*OUT*/*** *int dims[ndims]* );

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

Same as above



H L R S

# Conclusion and Outlook

Conclusions

- We developed a new algorithm to minimize the total communication time in a cluster of ccNUMA nodes with multi-core CPUs.
- It is needed, due to the significant bandwidth differences between inter- and intra-node communication.
- It can be implemented based on the algorithm in [4],
  but with a modified optimization goal,
  and repeated for each hardware level.

Outlook

- We plan to provide a portable implementation, and
- compare it with existing solutions with MPI_Dims_create + MPI_Cart_create.
- We plan to propose an appropriate interface for the next MPI standard, because MPI libraries may internally have faster access to the hardware topology information for a given communicator.

# 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.

[2] Bill Gropp. 2018. Using Node Information to Implement MPI Cartesian Topologies. In *Proceedings of the 25nd European MPI Users' Group Meeting (EuroMPI '18)*, September 23–26, 2018, Barcelona, Spain. ACM, New York, NY, USA, 9 pages.

[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.

[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.

# Appendix

- Additional material that is not part of the poster

# Renumbering in a 2-dim example with 3 levels (nodes / CPUs / cores)

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

dim1 = inner_d1 * mid_d1 * outer_d1 = 3*2*3 = 18

direction 1 →

oc1 = outer coordinate in 0..(outer_d1-1)
mc1= middle coordinate in 0..(mid_d1-1)
ic1 = inner coordinate in 0..(inner_d1-1)
c1 = process coordinate

oc0 = outer coordinate
mc0= middle coordinate
ic0 = inner coordinate
c0 = process coordinate

dim0 = inner_d0 * mid_d0 * outer_d0 = 2*2*3 = 12

| oc0 | mc0 | ic0 | c0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | oc1 | 0 | | | 1 | | | 2 | | | 0 | | | 1 | | | 2 | | |
| | | | mc1 | 0 | | | 1 | | | 0 | | | 1 | | | 0 | | | 1 | | |
| | | | ic1 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| 0 | 0 | 0 | 0 | 0 | 1 | 2 | 6 | 7 | 8 | 24 | 25 | 26 | 30 | 31 | 32 | 48 | | | 54 | | |
| | | | | →0 | →1 | →2 | →3 | →4 | →5 | →6 | →7 | →8 | →9 | →10 | →11 | →12 | →13 | →14 | →15 | →16 | →17 |
| | | 1 | 1 | 3 | 4 | 5 | 9 | 10 | 11 | 27 | 28 | 29 | 33 | 34 | 35 | | | | | | |
| | | | | →18 | →19 | →20 | →21 | →22 | →23 | →24 | →25 | →26 | →27 | →28 | →29 | →30 | →31 | →32 | →33 | →34 | →35 |
| | 1 | 0 | 2 | 12 | 13 | 14 | 18 | 19 | 20 | 36 | 37 | 38 | 42 | 43 | 44 | 60 | | | 66 | | |
| | | | | →36 | →37 | →38 | → | → | → | → | → | → | → | → | → | → | → | → | → | → | → |
| | | 1 | 3 | 15 | 16 | 17 | 21 | 22 | 23 | 39 | 40 | 41 | 45 | 46 | 47 | | | | | | |
| | | | | →54 | → | → | → | → | → | → | → | → | → | → | → | → | → | → | → | → | → |
| 1 | 0 | 0 | 4 | 72 | 73 | 74 | 78 | | | 96 | | | 102 | | | 120 | | | 126 | | |
| | | | | →72 | | | | | | | | | | | | | | | | | |
| | | 1 | 5 | 75 | 76 | 77 | | | | | | | | | | | | | | | |
| | | | | →90 | | | | | | | | | | | | | | | | | |
| | 1 | 0 | 6 | 84 | | | 90 | | | 108 | | | 114 | | | 132 | | | 138 | | |
| | | | | →108 | | | | | | | | | | | | | | | | | |
| | | 1 | 7 | 87 | | | | | | | | | | | | | | | | | |
| | | | | →126 | | | | | | | | | | | | | | | | | |
| 2 | 0 | 0 | 8 | 144 | | | 150 | | | 168 | | | 174 | | | 192 | | | 198 | | |
| | | | | →144 | | | | | | | | | | | | | | | | | |
| | | 1 | 9 | 147 | | | | | | | | | | | | | | | | | |
| | | | | →162 | | | | | | | | | | | | | | | | | |
| | 1 | 0 | 10 | 156 | | | 162 | | | 180 | | | 186 | | | 204 | | | 210 | | |
| | | | | →180 | | | | | | | | | | | | | | | | | |
| | | 1 | 11 | 159 | | | | | | | | | | | | | | | | | 215 |
| | | | | →198 | | | | | | | | | | | | | | | | | →215 |

**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 through the cluster

**New ranks in optimized communicator.**
→ New global communicator with
**minimal node-to-node &**
**optimal intra-node communication**

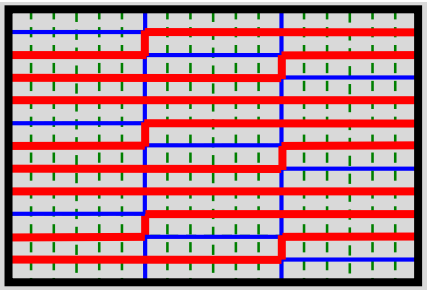
2-D example with 12*18 processes on hierarchical hardware

direction 0 ↓

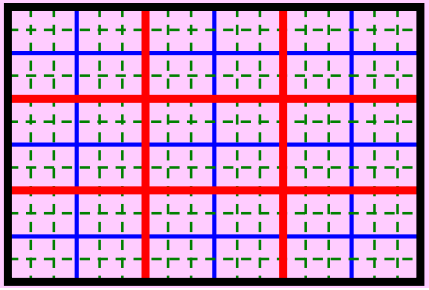
**Outer** = several ccNUMA nodes
**Middle** = CPUs of a ccNUMA node
**Inner** = cores of a CPU

**Number of communication links:**

**Without re-num-bering:**
**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.**

# Renumbering in a 2-dim example with 3 levels (nodes / CPUs / cores) – the code

This algorithm requires sequential ranking in MPI_COMM_WOLRD

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

```
/*Input: */ inner_d0=…;  mid_d0=…;   outer_d0=…;
            inner_d1=…;  mid_d1=…;   outer_d1=…;
dim0=inner_d0*mid_d0*outer_d0;          dim1=inner_d1*mid_d1*outer_d1;
idim=inner_d0*inner_d1; mdim=mid_d0*mid_d1;  odim=outer_d0*outer_d1;
whole_size=dim0*dim1   /* 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++)
     { old_rank = ic1 + inner_d1*ic0 + (mc1 + mid_d1  *mc0)*idim
                                     + (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;
       new_rank = c1 + dim1*c0;
       ranks[new_rank] = old_rank;
     }
/* Establishing new_comm with the new ranking in a array "ranks": */
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;
MPI_Cart_create(new_comm, 2, dims, periods, 0 /*=false*/, &comm_cart);
```

/* final output */

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

# MPI_Dims_create optimizing $\sum_{i=1}^{d} n_i$

- Based on Jesper Träff's algorithm *tuwdims.c* developed for [4]
  - Jesper's optimization criterion:
    - Minimizing $\Delta = n_1 - n_d$
- New optimization criterion (three criteria)
  - 1st criterion: a factorization is better if $\sum_{i=1}^{d} n_i$ **is smaller**
  - 2nd criterion: if the $\sum_{i=1}^{d} n_i$ is the same then a factorization is better if $\Delta = n_1 - n_d$ is smaller
  - 3rd criterion: if $\sum_{i=1}^{d} n_i$ and $\Delta = n_1 - n_d$ are the same then a factorization is better if $n_1$ is smaller
- Both algorithms have nearly same execution time: ~O(10µs/call)
- For n=2..10,000,000 and ndims=2..10
  - 4630x different factorization: with **same $\Sigma$ and** $\Delta$ and smaller $dims[0]$
    - e.g., ndims=3: N=**360 = (old) 10**x**6**x**6 = (new) 9**x**8**x**5** (both $\Sigma$=22 and $\Delta$=4)
  - 6066x different factorization: with **better $\Sigma$** and **mostly worse** $\Delta$
    - e.g., ndims=3: N=**35200 = (old) 40**x**40**x**22** ($\Sigma$=102, $\Delta$=18) **= (new) 44**x**32**x**25** ($\Sigma$=**101**, $\Delta$=19)
    - N=**37044 = (old) 42**x**42**x**21** ($\Sigma$=105, $\Delta$=21) **= (new) 49**x**28**x**27** ($\Sigma$=**104**, $\Delta$=22)

Next below 10,000 are:
- ndims=3:
  22x14x12 = 21x16x11
  21x16x15 = 20x18x14
  26x16x15 = 24x20x13
- ndims=4:
  10xFx6x6 = 9x8xFx5
  with F=6,7,8,9

All other larger than 50,000

[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.