
Parallel programming / computation

Sultan ALPAR

s.alpar@iitu.edu.kz

IITU

Lecture 8

Virtual Topologies

Neighborhood communication

Optimization through reordering

= perfect scalable !?

Course Chap. 9-(2):

Sparse Collective Operations on Process Topologies

New in MPI-3.0

- MPI process topologies (Cartesian and (distributed) graph) usable for communication
 - `MPI_(I)NEIGHBOR_ALLGATHER(V)`
 - `MPI_(I)NEIGHBOR_ALLTOALL(V,W)`
- If the topology is the full graph, then neighbor routine is identical to full collective communication routine
 - Exception: `s/rdispls` in `MPI_NEIGHBOR_ALLTOALLW` are `MPI_Aint`

= perfect scalable !?

Course Chap. 9-(2):

Sparse Collective Operations on Process Topologies

New in MPI-3.0

- MPI process topologies (Cartesian and (distributed) graph) usable for communication
 - `MPI_(I)NEIGHBOR_ALLGATHER(V)`
 - `MPI_(I)NEIGHBOR_ALLTOALL(V,W)`
- If the topology is the full graph, then neighbor routine is identical to full collective communication routine
 - Exception: `s/rdispls` in `MPI_NEIGHBOR_ALLTOALLW` are `MPI_Aint`
- Allows for optimized communication scheduling and scalable resource binding

= perfect scalable !?

Course Chap. 9-(2):

Sparse Collective Operations on Process Topologies

New in MPI-3.0

- MPI process topologies (Cartesian and (distributed) graph) usable for communication
 - `MPI_(I)NEIGHBOR_ALLGATHER(V)`
 - `MPI_(I)NEIGHBOR_ALLTOALL(V,W)`
- If the topology is the full graph, then neighbor routine is identical to full collective communication routine
 - Exception: `s/rdispls` in `MPI_NEIGHBOR_ALLTOALLW` are `MPI_Aint`
- Allows for optimized communication scheduling and scalable resource binding
- Cartesian topology:
 - Sequence of buffer segments is communicated with:
 - **direction=0 source, direction=0 dest, direction=1 source, direction=1 dest, ...**
 - Defined only for `disp=1` (`direction`, `source`, `dest` and `disp` are defined as in `MPI_CART_SHIFT`)

= perfect scalable !?

Course Chap. 9-(2):

Sparse Collective Operations on Process Topologies

New in MPI-3.0

- MPI process topologies (Cartesian and (distributed) graph) usable for communication
 - `MPI_(I)NEIGHBOR_ALLGATHER(V)`
 - `MPI_(I)NEIGHBOR_ALLTOALL(V,W)`
- If the topology is the full graph, then neighbor routine is identical to full collective communication routine
 - Exception: `s/rdispls` in `MPI_NEIGHBOR_ALLTOALLW` are `MPI_Aint`
- Allows for optimized communication scheduling and scalable resource binding
- Cartesian topology:
 - Sequence of buffer segments is communicated with:
 - **direction=0 source, direction=0 dest, direction=1 source, direction=1 dest, ...**
 - Defined only for `disp=1` (`direction`, `source`, `dest` and `disp` are defined as in `MPI_CART_SHIFT`)
 - If a source or dest rank is `MPI_PROC_NULL` then the buffer location is still there but the content is not touched.

= perfect scalable !?

Course Chap. 9-(2):

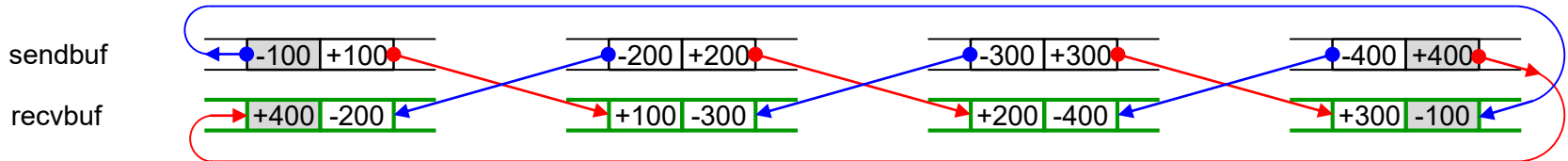
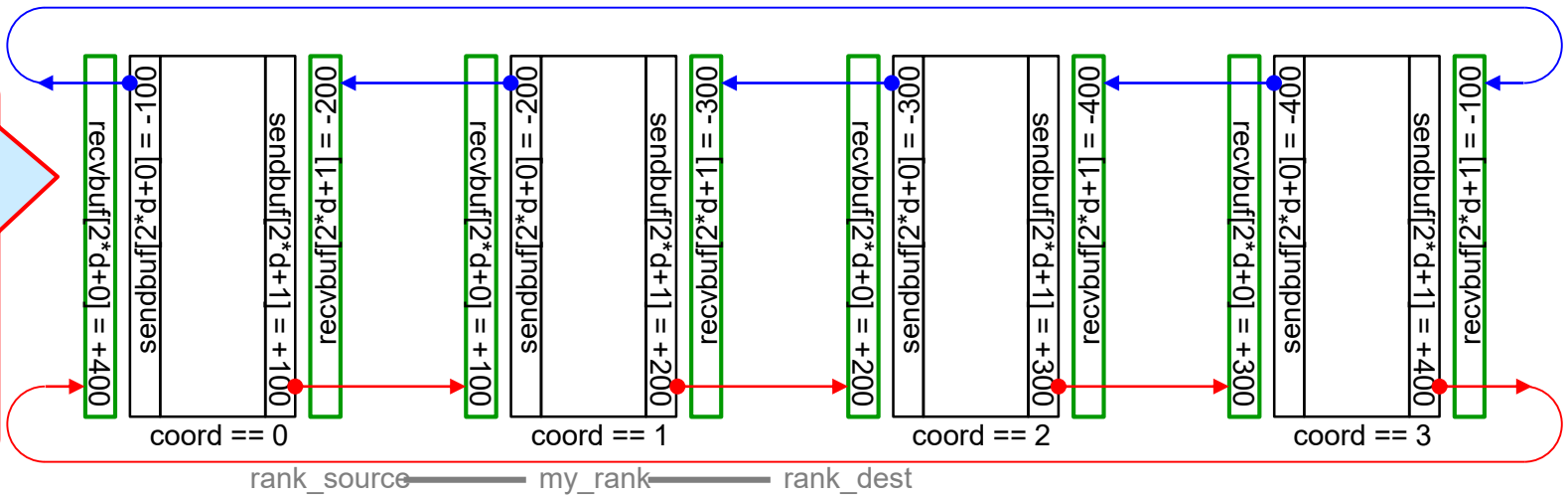
Sparse Collective Operations on Process Topologies

New in MPI-3.0

- MPI process topologies (Cartesian and (distributed) graph) usable for communication
 - `MPI_(I)NEIGHBOR_ALLGATHER(V)`
 - `MPI_(I)NEIGHBOR_ALLTOALL(V,W)`
- If the topology is the full graph, then neighbor routine is identical to full collective communication routine
 - Exception: `s/rdispls` in `MPI_NEIGHBOR_ALLTOALLW` are `MPI_Aint`
- Allows for optimized communication scheduling and scalable resource binding
- Cartesian topology:
 - Sequence of buffer segments is communicated with:
 - **direction=0 source, direction=0 dest, direction=1 source, direction=1 dest, ...**
 - Defined only for `disp=1` (`direction`, `source`, `dest` and `disp` are defined as in `MPI_CART_SHIFT`)
 - If a source or dest rank is `MPI_PROC_NULL` then the buffer location is still there but the content is not touched.
 - See exercise 5 and advanced exercise 6

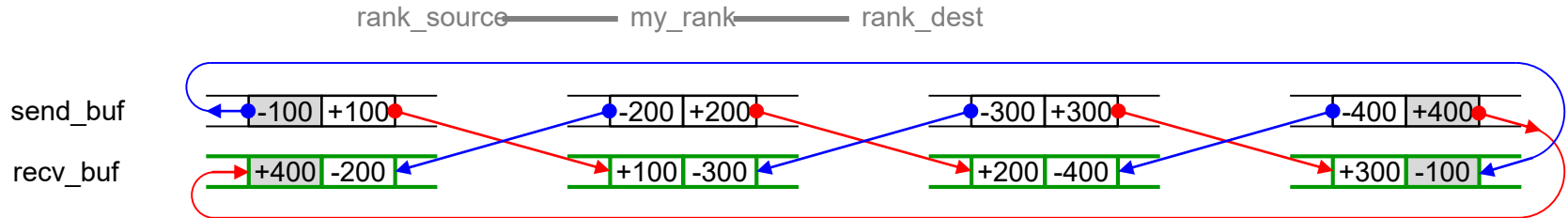
Periodic MPI_NEIGHBOR_ALLTOALL in direction d with 4 processes

This figure represents one direction d . Of course, it is valid for any direction



... grey array entries are used only if `periods[d] == non-zero` in C or `.TRUE.` in Fortran

As if ...

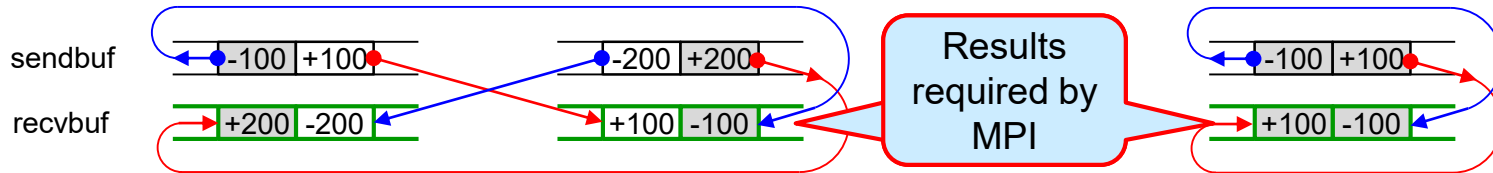
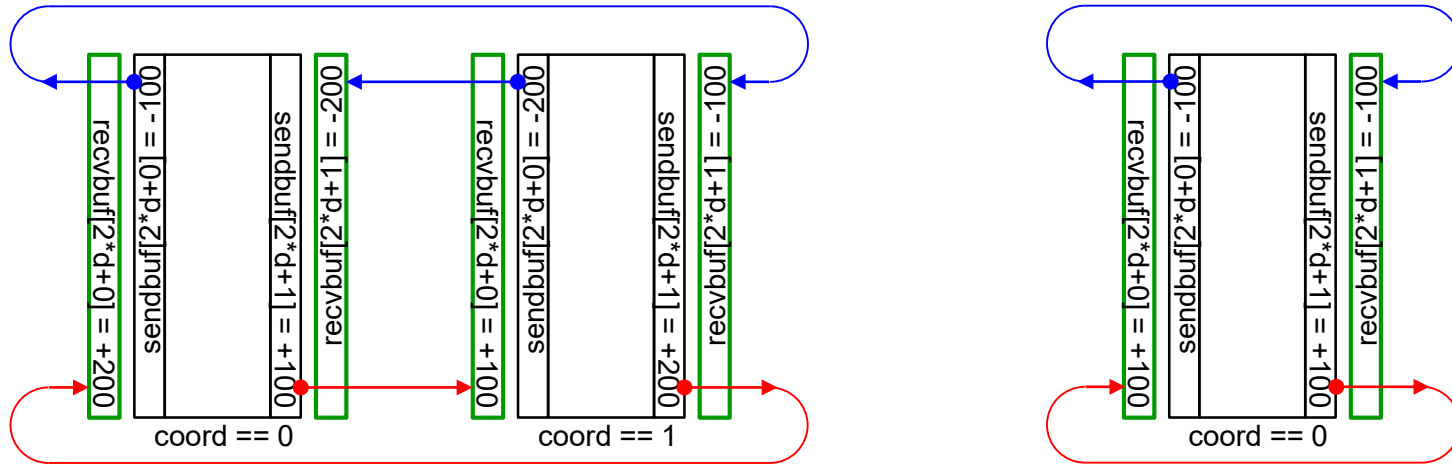


After MPI_NEIGHBOR_ALLTOALL on a Cartesian communicator returned, the content of the `recvbuf` is **as if** the following code is executed:

```
MPI_Cartdim_get(comm, &ndims);
for( /*direction*/ d = 0; d < ndims; d++) {
    MPI_Cart_shift(comm, /*direction*/ d, /*disp*/ 1, &rank_source, &rank_dest);
    MPI_Sendrecv(sendbuf[d*2+0], sendcount, sendtype, rank_source, /*sendtag*/ d*2,
                recvbuf[d*2+1], recvcount, recvtype, rank_dest, /*recvtag*/ d*2,
                comm, &status); /* 1st communication in direction of displacement -1 */
    MPI_Sendrecv(sendbuf[d*2+1], sendcount, sendtype, rank_dest, /*sendtag*/ d*2+1,
                recvbuf[d*2+0], recvcount, recvtype, rank_source, /*recvtag*/ d*2+1,
                comm, &status); /* 2nd communication in direction of displacement +1 */
}
```

The tags are chosen to guarantee that both communications (i.e., in negative and positive direction) cannot be mixed up, even if the MPI_SENDRECV is substituted by nonblocking communication and the MPI_ISEND and MPI_IRECV calls are started in any sequence.

Wrong implementations of periodic MPI_NEIGHBOR_ALLTOALL with only 2 and 1 processes



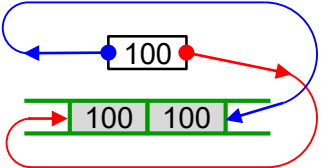
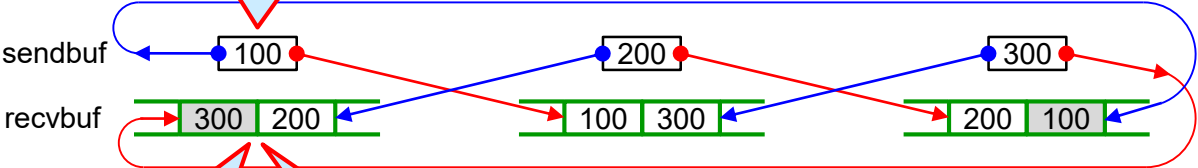
Wrong results with `openmpi/4.0.1-gnu-8.3.0` and `cray-mpich/7.7.6` with 2 and 1 processes:



Clarified in MPI-4.0

Communication pattern of MPI_NEIGHBOR_ALLGATHER

The send_buf is only one element, which is sent to the neighbor processes in all directions



... grey array entries are used only if periods[d] == non-zero in C or .TRUE. In Fortran

The recv_buf represents one direction d . Of course, this figure is valid for any direction

The green recv_buf elements are $recvbuf[2*d+0]$ and $recvbuf[2*d+1]$

Other MPI features: MPI_BOTTOM and absolute addresses

- MPI_BOTTOM in point-to-point and collective communication:
 - Buffer argument is MPI_BOTTOM
 - Then absolute addresses can be used in
 - **Communication routines with byte displacement arguments, e.g., MPI_(I)NEIGHBOR_ALLTOALLW**
 - **Derived datatypes with byte displacements**
 - Displacements must be retrieved with MPI_GET_ADDRESS()

Other MPI features: MPI_BOTTOM and absolute addresses

- MPI_BOTTOM in point-to-point and collective communication:
 - Buffer argument is MPI_BOTTOM
 - Then absolute addresses can be used in
 - **Communication routines with byte displacement arguments, e.g., MPI_(I)NEIGHBOR_ALLTOALLW**
 - **Derived datatypes with byte displacements**
 - Displacements must be retrieved with MPI_GET_ADDRESS()
 - MPI_BOTTOM is an address, i.e., **cannot be assigned to a Fortran variable!**
 - MPI-3.1/MPI-4.0, Section 2.5.4, page 15 line 45 – page 16 line 6 / page 21 lines 14-23 shows all such address constants that cannot be used in expressions or assignments **in Fortran**, e.g.,
 - **MPI_STATUS_IGNORE** (→ point-to-point comm.)
 - **MPI_IN_PLACE** (→ collective comm.)

Fortran

Other MPI features: MPI_BOTTOM and absolute addresses

- MPI_BOTTOM in point-to-point and collective communication:
 - Buffer argument is MPI_BOTTOM
 - Then absolute addresses can be used in
 - **Communication routines with byte displacement arguments, e.g., MPI_(I)NEIGHBOR_ALLTOALLW**
 - **Derived datatypes with byte displacements**
 - Displacements must be retrieved with MPI_GET_ADDRESS()
 - MPI_BOTTOM is an address, i.e., **cannot be assigned to a Fortran variable!**
 - MPI-3.1/MPI-4.0, Section 2.5.4, page 15 line 45 – page 16 line 6 / page 21 lines 14-23 shows all such address constants that cannot be used in expressions or assignments **in Fortran**, e.g.,
 - **MPI_STATUS_IGNORE** (→ point-to-point comm.)
 - **MPI_IN_PLACE** (→ collective comm.)
 - Fortran: Using MPI_BOTTOM & absolute displacement of variable X
 → **<type>, ASYNCHRONOUS :: X** and **MPI_F_SYNC_REG(X)** is needed:
 - **MPI_BOTTOM** in a blocking MPI routine → **MPI_F_SYNC_REG** before and after this routine
 - in a nonblocking routine → **MPI_F_SYNC_REG** before this routine & after final **WAIT/TEST**

Fortran

New in MPI-3.0

Exercise 5 — Neighbor Collective Communication

In MPI/tasks/...

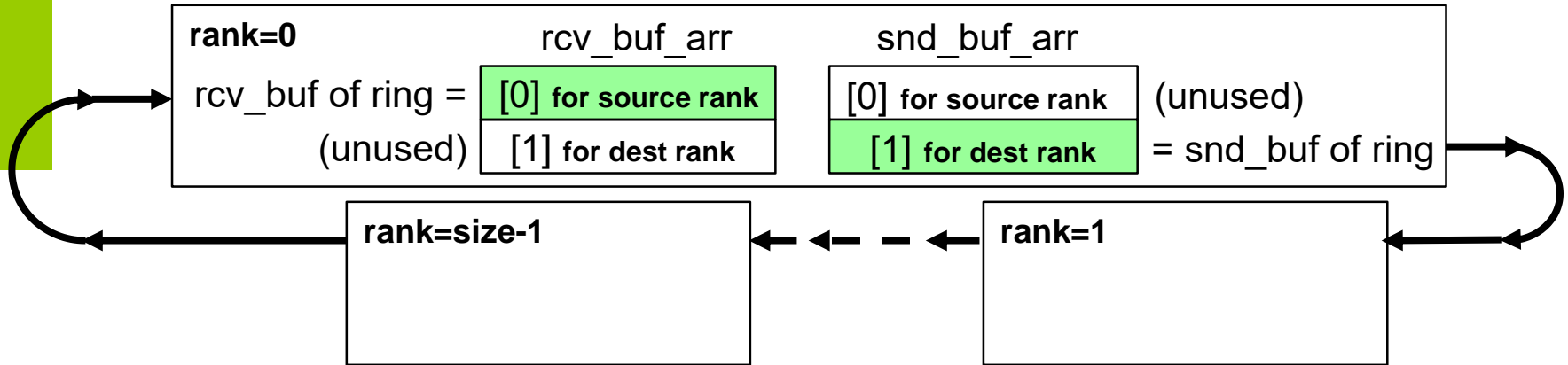
Use **C** `C/Ch9/ring_neighbor_alltoall_skel.c`
 or **Fortran** `F_30/Ch9/ring_neighbor_alltoall_skel_30.f90`
 or **Python** `PY/Ch9/ring_neighbor_alltoall_skel.py`

In this example, we ignore the communication in the other direction.
 Of course in real applications, both communications (to the left and to the right) are used.

Keep the ring communication in the virtual topology example, but substitute the point-to-point communication by neighborhood collective:

- I.e., `Isend-Recv-Wait` → one call to `MPI_Neighbor_alltoall`
- `rcv_buf` and `snd_buf` must be extended to a `rcv_buf_arr` and `snd_buf_arr` with `rcv_buf_arr[0]` as `rcv_buf` and `snd_buf_arr[1]` as `snd_buf`, i.e., according to the sequence rule for the buffer segments.
- `snd_count` and `rcv_count` are both 1 (not 2!), describing one buffer, not the array of buffers (i.e., one message)!

Exercise 5



Exercise 6 (advanced) —

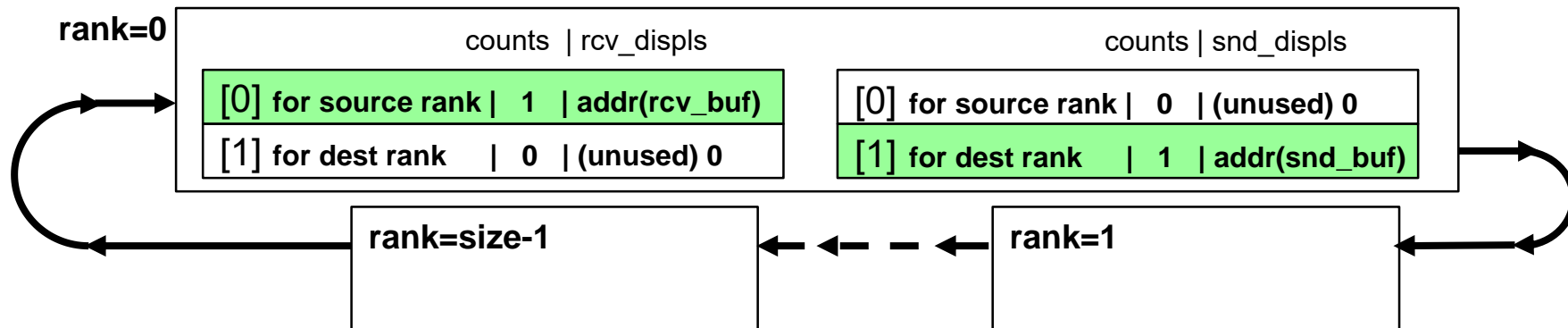
Neighbor Collective Communication & MPI_BOTTOM

Use **C** C/Ch9/ring_neighbor_alltoallw_skel.c
 or **Fortran** F_30/Ch9/ring_neighbor_alltoallw_skel_30.f90
 or **Python** PY/Ch9/ring_neighbor_alltoallw_skel.py

mpi4py may require `mem_from_bottom = MPI.memory.fromaddress(MPI.BOTTOM,0,0)` and passing `mem_from_bottom` instead of `MPI.BOTTOM`, e.g., in `(mem_from_bottom, snd_counts, snd_displs, snd_types)` as send buffer in `comm.Neighbor_alltoallw(...)`

You start again from the virtual topology example, but substitute the point-to-point communication by `MPI_NEIGHBOR_ALLTOALLW` with `MPI_BOTTOM` and absolute addresses of `rcv_buf` and `snd_buf`:

- I.e., `Isend-Recv-Wait` → one call to `MPI_Neighbor_alltoallw`
- Fortran: Do not forget to call `MPI_F_SYNC_REG` for the real variables behind `MPI_BOTTOM` (i.e., `snd_buf`, `rcv_buf`) **before & after** the communication call!



CAUTION: Officially, this example is not portable, because address differences are allowed only inside of structures or arrays, i.e., `snd_buf` and `rcv_buf` need to be part of a common space → [MPI-3.1, 4.1.12](#)
[MPI-4.0, 5.1.12](#)

Quiz on Chapter 9-(2) – Neighborhood communication

- A. Can you think of scenarios where collective neighborhood communication would be beneficial over nonblocking pt-to-pt communication, and what routines would you use?
1. _____
 2. _____
- B. Which alternatives should be considered? Not yet discussed
1. _____
 2. _____
- C. What must the application programmer in general do to enable these opportunities?
1. _____

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

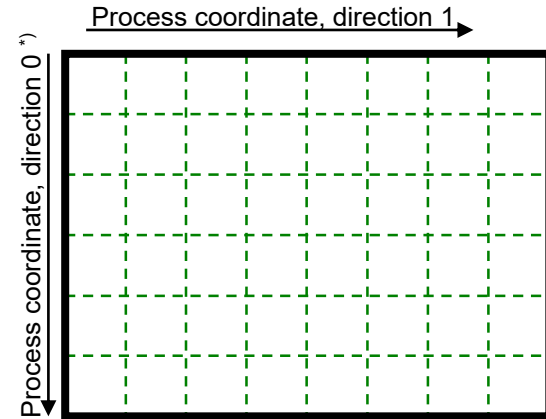
- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication


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

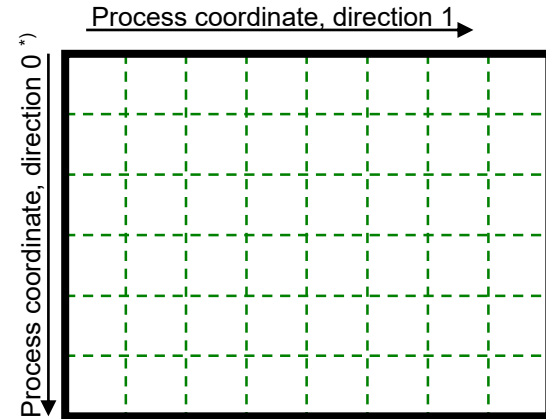
- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

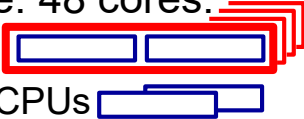

Re-numbering on a cluster of SMPs (cores / CPUs / nodes)

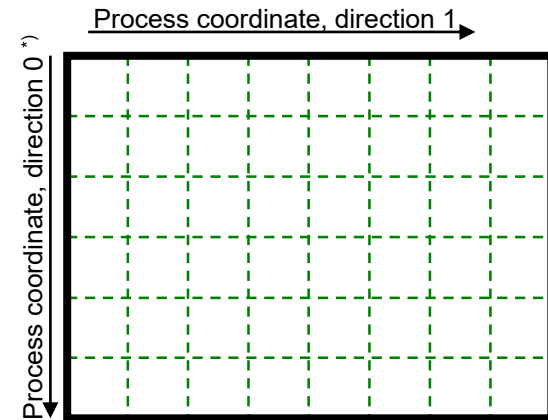
- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8 with 1000 x 1010 mesh points/core
- Hardware example: 48 cores:
 - 4 ccNUMA nodes 



*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

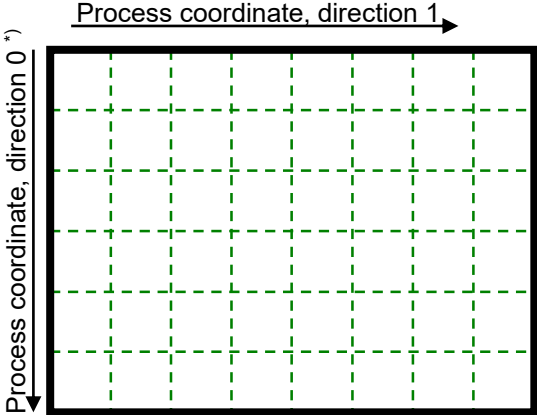
- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8 with 1000 x 1010 mesh points/core
- Hardware example: 48 cores:
 - 4 ccNUMA nodes 
 - each node with 2 CPUs 






*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8 with 1000 x 1010 mesh points/core

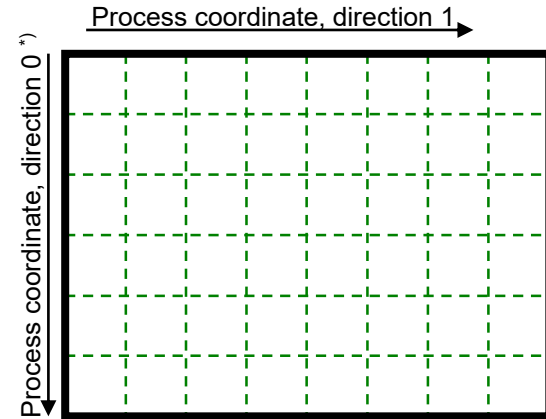





- Hardware example: 48 cores:
 - 4 ccNUMA nodes 
 - each node with 2 CPUs 
 - each CPU with 6 cores 

*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8 with 1000 x 1010 mesh points/core

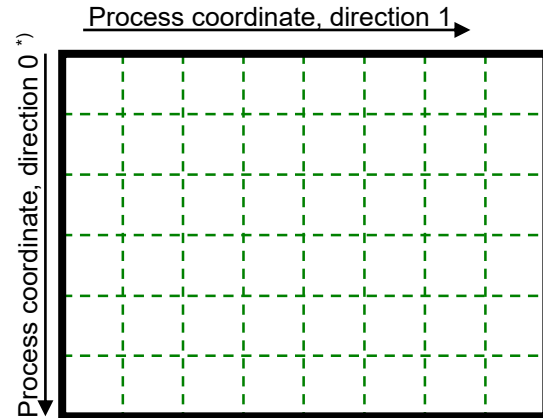





- Hardware example: 48 cores:
 - 4 ccNUMA nodes 
 - each node with 2 CPUs 
 - each CPU with 6 cores 
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD

*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



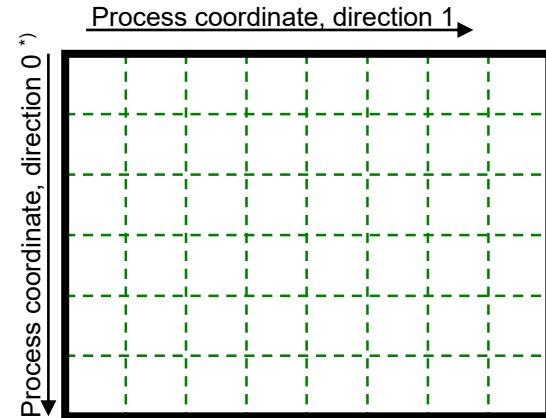
- Hardware example: 48 cores:
 - 4 ccNUMA nodes 
 - each node with 2 CPUs 
 - each CPU with 6 cores 
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47

*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



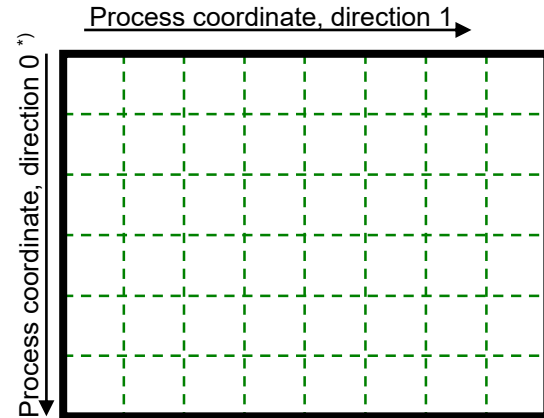
- Hardware example: 48 cores:
 - 4 ccNUMA nodes
 - each node with 2 CPUs
 - each CPU with 6 cores
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD




0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47

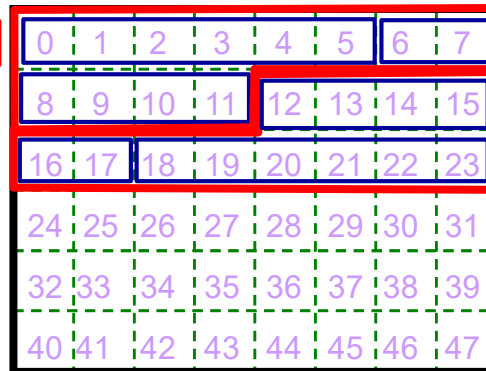
*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



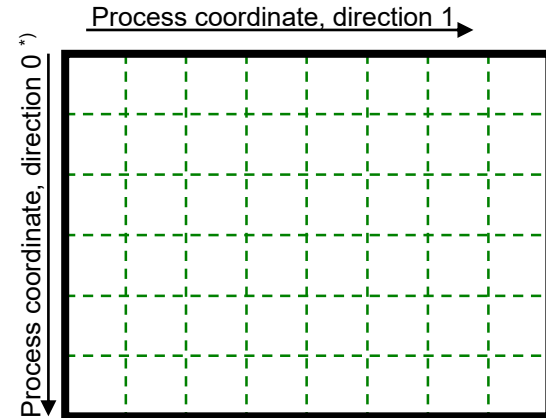
- Hardware example: 48 cores:
 - 4 ccNUMA nodes 
 - each node with 2 CPUs 
 - each CPU with 6 cores 
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD






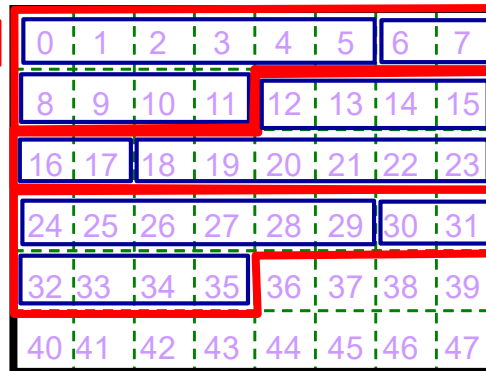
*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



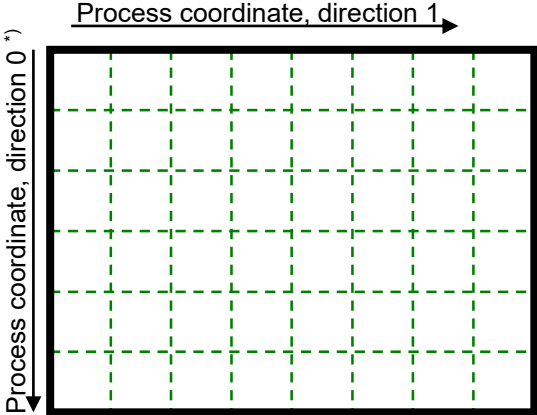
- Hardware example: 48 cores:
 - 4 ccNUMA nodes 
 - each node with 2 CPUs 
 - each CPU with 6 cores 
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD



*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8 with 1000 x 1010 mesh points/core



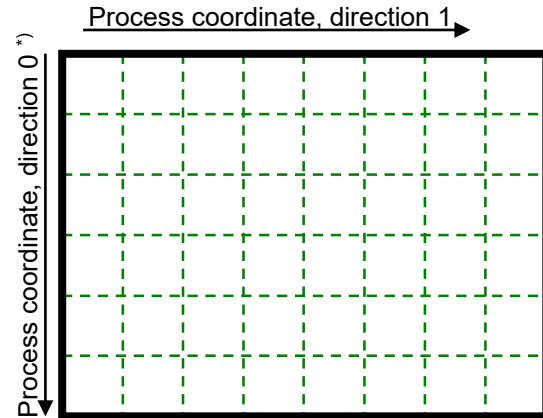
- Hardware example: 48 cores:
 - 4 ccNUMA nodes
 - each node with 2 CPUs
 - each CPU with 6 cores
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD






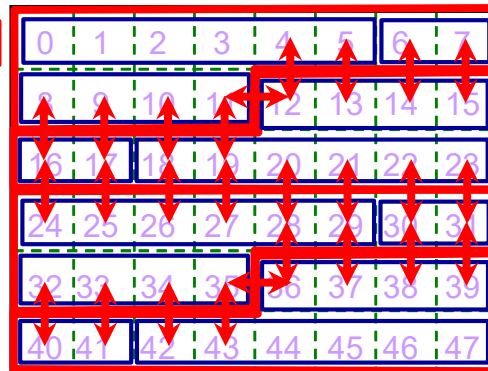
*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



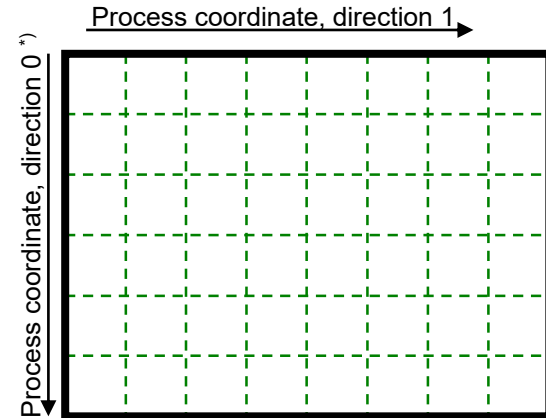
- Hardware example: 48 cores:
 - 4 ccNUMA nodes 
 - each node with 2 CPUs 
 - each CPU with 6 cores 
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD






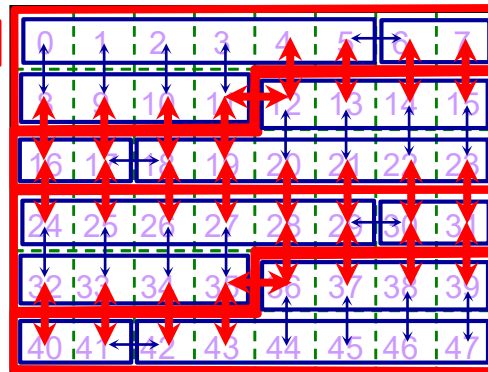
*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



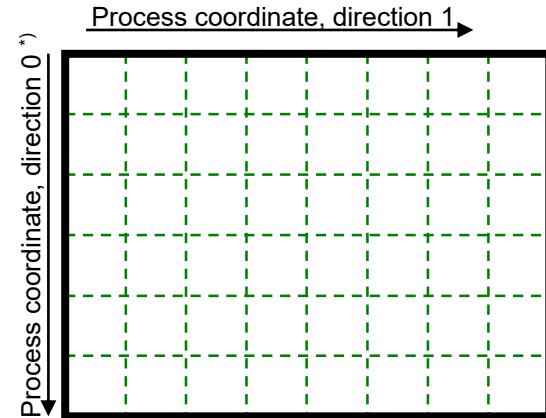
- Hardware example: 48 cores:
 - 4 ccNUMA nodes 
 - each node with 2 CPUs 
 - each CPU with 6 cores 
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD






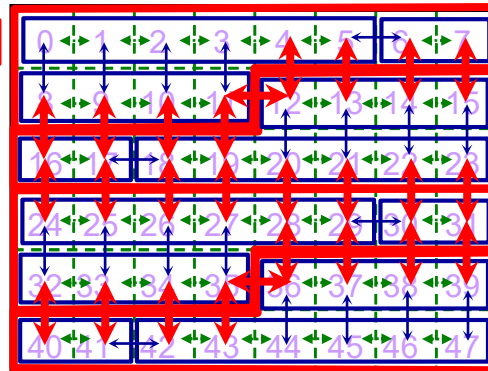
*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



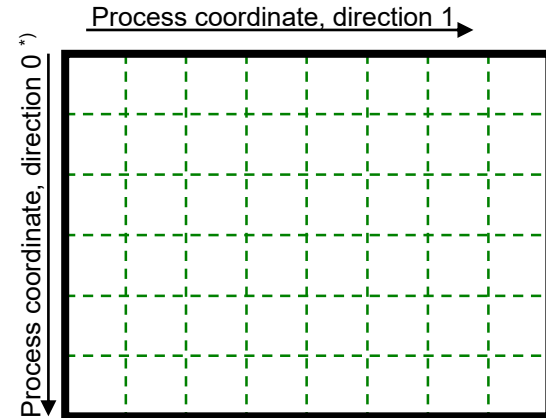
- Hardware example: 48 cores:
 - 4 ccNUMA nodes 
 - each node with 2 CPUs 
 - each CPU with 6 cores 
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD



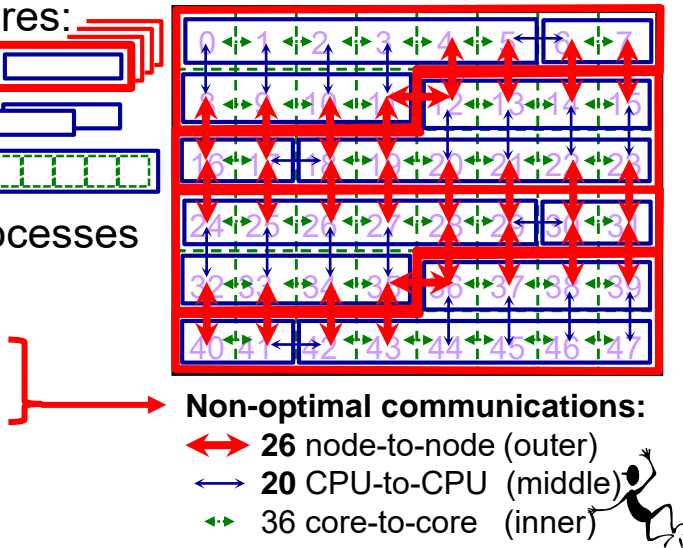
*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



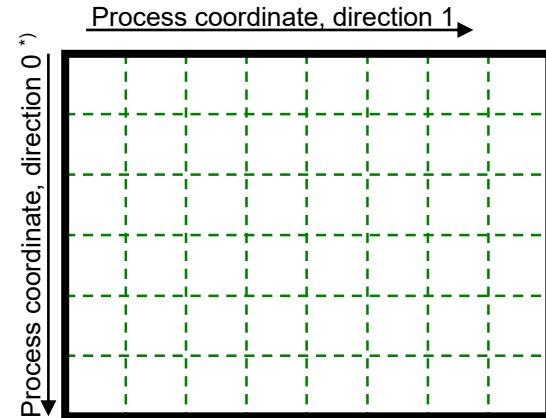
- Hardware example: 48 cores:
 - 4 ccNUMA nodes
 - each node with 2 CPUs
 - each CPU with 6 cores
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD



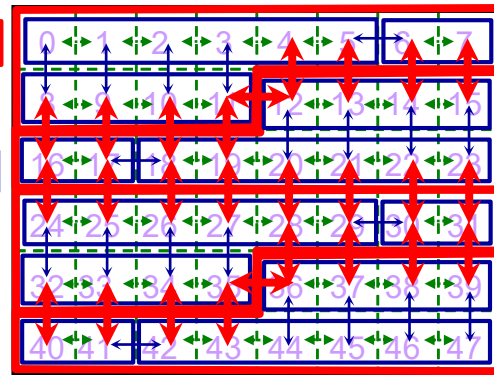
*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



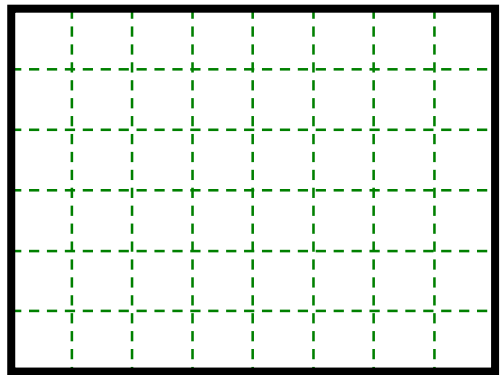
- Hardware example: 48 cores:
 - 4 ccNUMA nodes
 - each node with 2 CPUs
 - each CPU with 6 cores



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

Non-optimal communications:

- ↔ 26 node-to-node (outer)
- ↔ 20 CPU-to-CPU (middle)
- ↔ 36 core-to-core (inner)



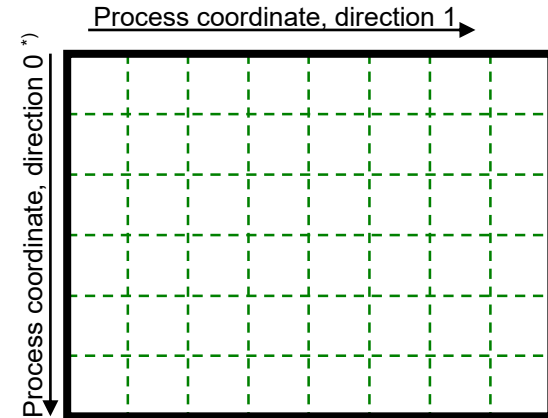
Optimized placement:



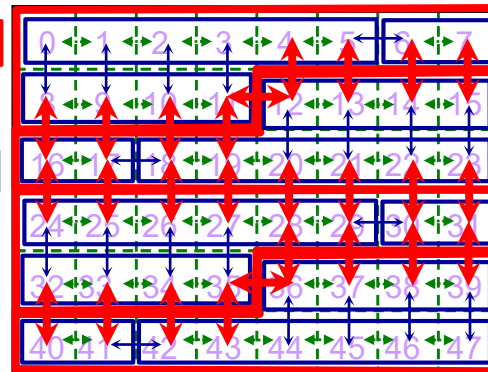
*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

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

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core

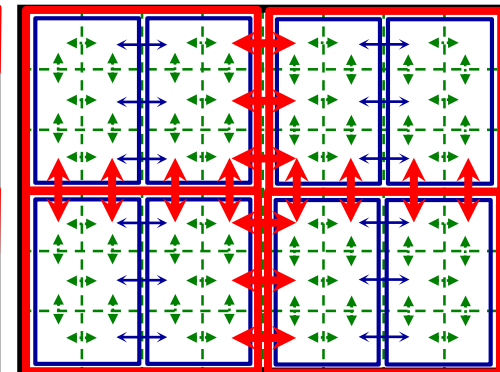


- Hardware example: 48 cores:
 - 4 ccNUMA nodes
 - each node with 2 CPUs
 - each CPU with 6 cores
- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD
 - Optimized placement



Non-optimal communications:

- ↔ 26 node-to-node (outer)
- ↔ 20 CPU-to-CPU (middle)
- ↔ 36 core-to-core (inner)



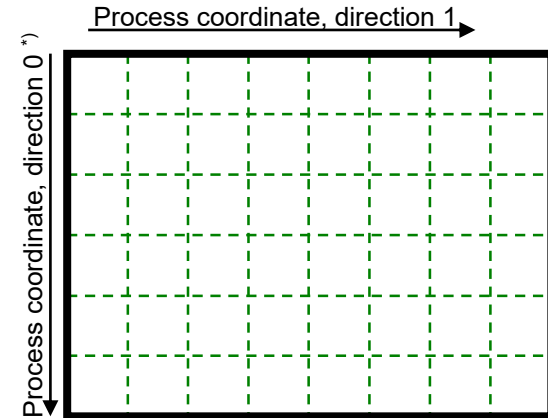
Optimized placement:

- ↔ Only 14 node-to-node
- ↔ Only 12 CPU-to-CPU
- ↔ 56 core-to-core

*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

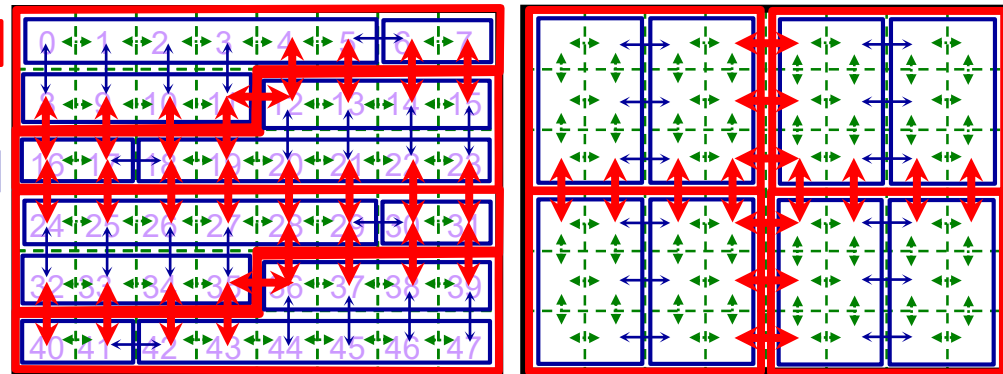
Re-numbering on a cluster of SMPs (cores / CPUs / nodes)

- Example:
 - 2-dim 6000 x 8080 data mesh points
 - To be parallelized on 48 cores
- Minimal communication
 - Subdomains as quadratic as possible
 - minimal circumference
 - minimal halo communication
 - virtual 2-dim process grid: 6 x 8
 - with 1000 x 1010 mesh points/core



- Hardware example: 48 cores:
 - 4 ccNUMA nodes
 - each node with 2 CPUs
 - each CPU with 6 cores

- How to locate the MPI processes on the hardware?
 - Using sequential ranks in MPI_COMM_WORLD
 - Optimized placement
 - See next slides and example code



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

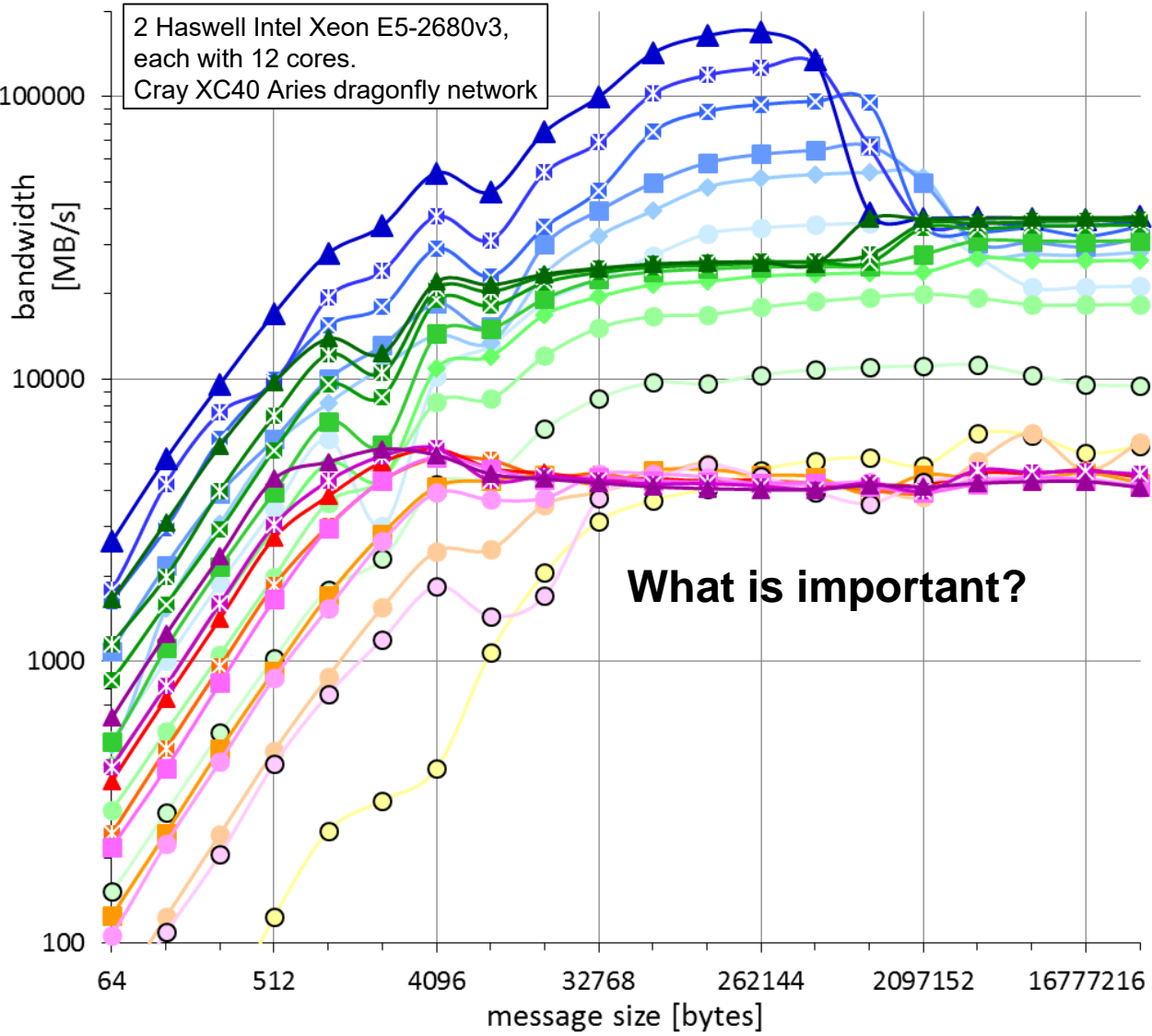
*) Figure: as in a math matrix, first index is vertical (i.e., not horizontal as in a x,y-diagram)

Duplex accumulated ring bandwidth per node

Further details on the benchmarks, see next slide

(each message is counted twice, as outgoing and incoming)

2 Haswell Intel Xeon E5-2680v3, each with 12 cores.
Cray XC40 Aries dragonfly network



- Intra-CPU 2 cores/cpu
- ◇ Intra-CPU 3 cores/cpu
- Intra-CPU 4 cores/cpu
- × Intra-CPU 6 cores/cpu
- * Intra-CPU 8 cores/cpu
- ▲ Intra-CPU 12 cores/cpu

- Intra-node 1 core/cpu
- Intra-node 2 cores/cpu
- ◆ Intra-node 3 cores/cpu
- Intra-node 4 cores/cpu
- × Intra-node 6 cores/cpu
- * Intra-node 8 cores/cpu
- ▲ Intra-node 12 cores/cpu

- Inter-node, 1 CPU, 1 core/cpu
- Inter-node, 1 CPU, 2 cores/cpu
- Inter-node, 1 CPU, 4 cores/cpu
- × Inter-node, 1 CPU, 8 cores/cpu
- ▲ Inter-node, 1 CPU, 12 cores/cpu

- Inter-node, 2 CPUs, 1 core/cpu
- Inter-node, 2 CPUs, 2 cores/cpu
- Inter-node, 2 CPUs, 4 cores/cpu
- × Inter-node, 2 CPUs, 8 cores/cpu
- ▲ Inter-node, 2 CPUs, 12 cores/cpu

(A)

(B)

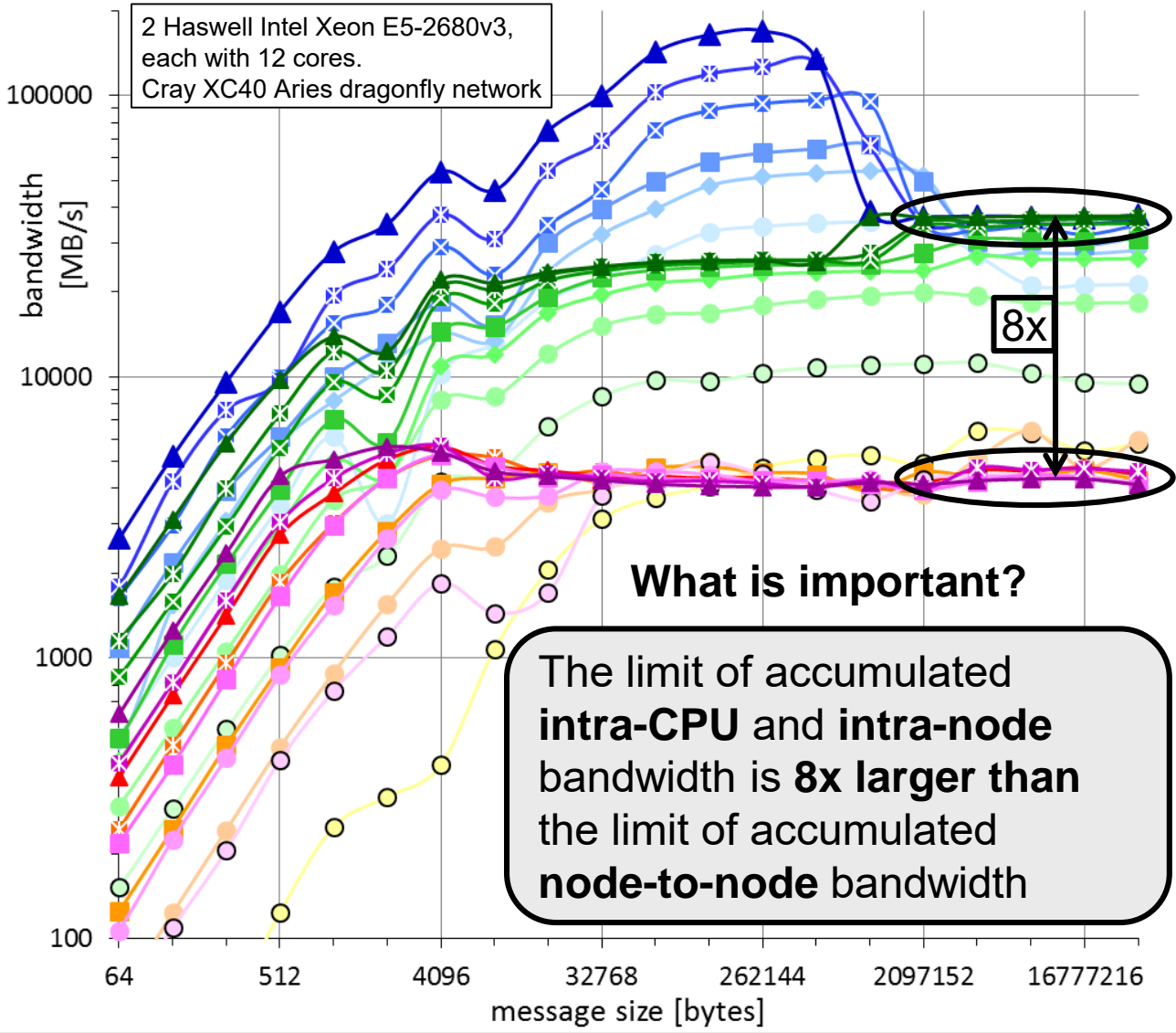
(C)

(D)

Duplex accumulated ring bandwidth per node

Further details on the benchmarks, see next slide

(each message is counted twice, as outgoing and incoming)



- Intra-CPU 2 cores/cpu
- ◇ Intra-CPU 3 cores/cpu
- Intra-CPU 4 cores/cpu
- × Intra-CPU 6 cores/cpu
- * Intra-CPU 8 cores/cpu
- ▲ Intra-CPU 12 cores/cpu

- Intra-node 1 core/cpu
- ◇ Intra-node 2 cores/cpu
- Intra-node 3 cores/cpu
- Intra-node 4 cores/cpu
- × Intra-node 6 cores/cpu
- * Intra-node 8 cores/cpu
- ▲ Intra-node 12 cores/cpu

- Inter-node, 1 CPU, 1 core/cpu
- Inter-node, 1 CPU, 2 cores/cpu
- Inter-node, 1 CPU, 4 cores/cpu
- × Inter-node, 1 CPU, 8 cores/cpu
- ▲ Inter-node, 1 CPU, 12 cores/cpu

- Inter-node, 2 CPUs, 1 core/cpu
- ◇ Inter-node, 2 CPUs, 2 cores/cpu
- Inter-node, 2 CPUs, 4 cores/cpu
- × Inter-node, 2 CPUs, 8 cores/cpu
- ▲ Inter-node, 2 CPUs, 12 cores/cpu

(A)

(B)

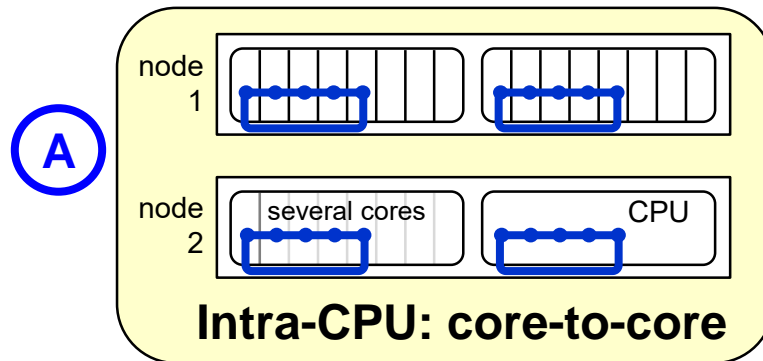
(C)

(D)

Multiple communicating rings

Benchmark MPI/tasks/C/halo-benchmarks/halo_irecv_send_multiplelinks_toggle.c

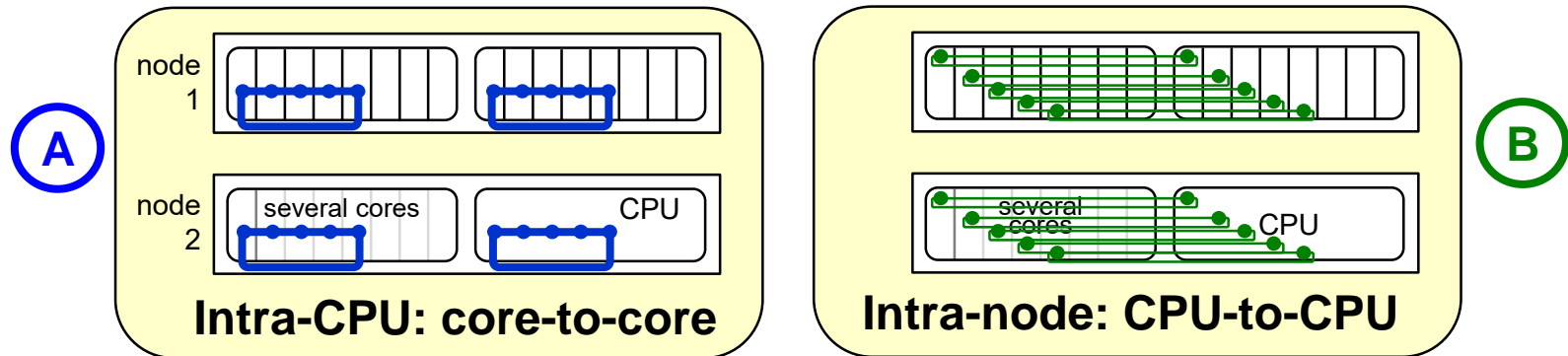
- Varying message size,
- number of **communication cores per CPU**, and
- four communication schemes (example with 5 **communicating cores per CPU**)



Multiple communicating rings

Benchmark MPI/tasks/C/halo-benchmarks/halo_irecv_send_multiplelinks_toggle.c

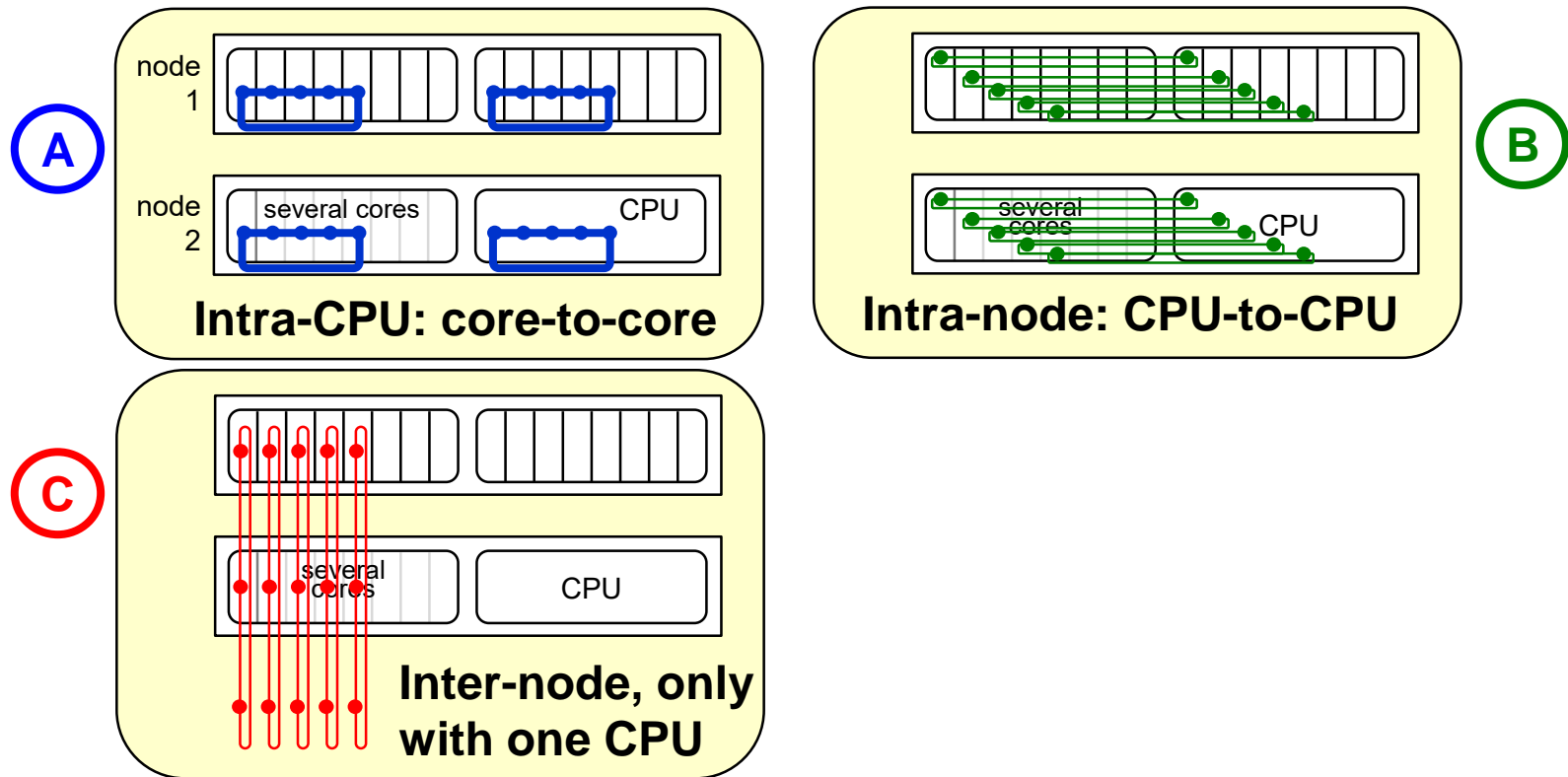
- Varying message size,
- number of **communication cores per CPU**, and
- four communication schemes (example with 5 **communicating cores per CPU**)



Multiple communicating rings

Benchmark MPI/tasks/C/halo-benchmarks/halo_irecv_send_multiplelinks_toggle.c

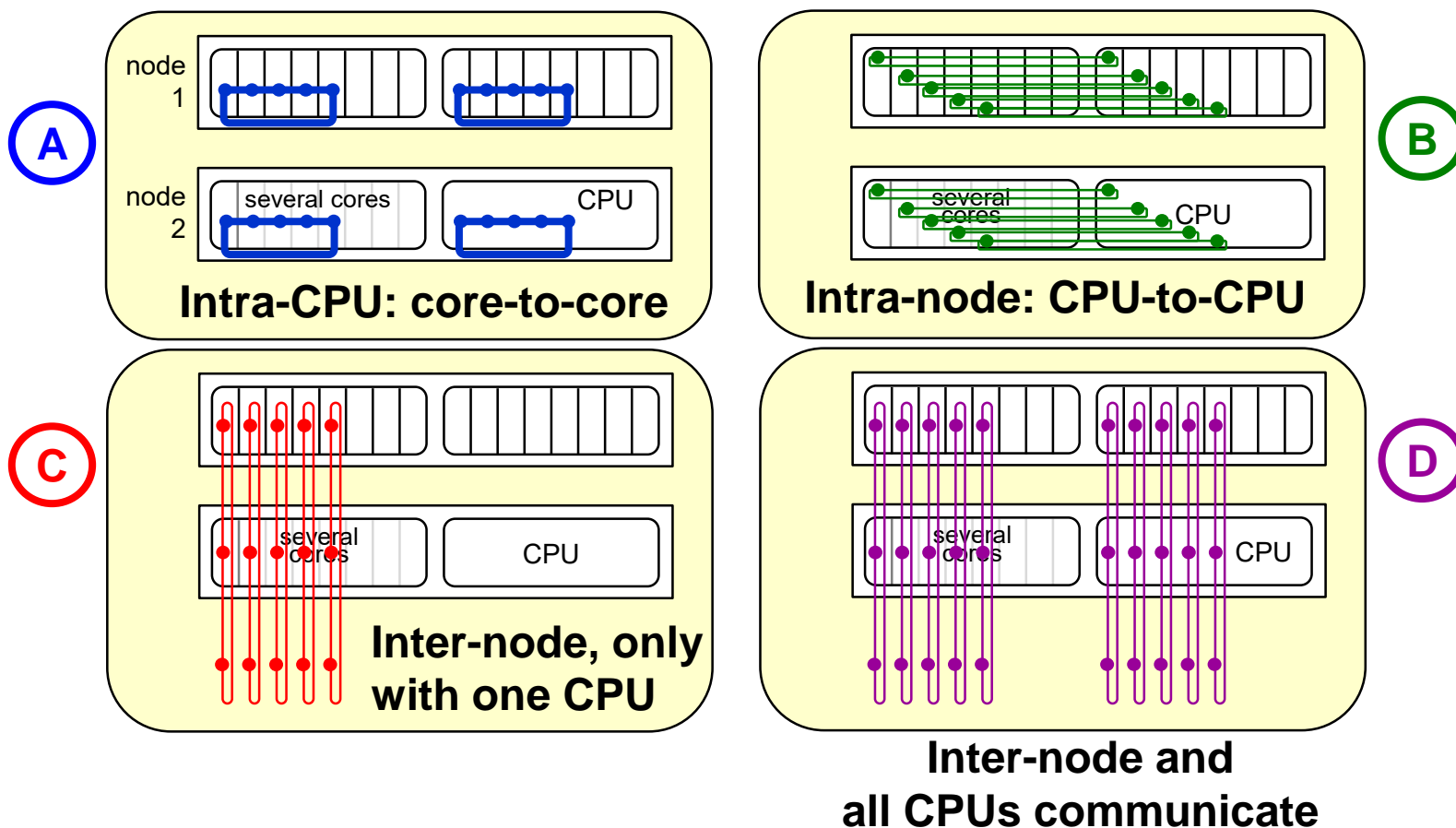
- Varying message size,
- number of **communication cores per CPU**, and
- four communication schemes (example with 5 **communicating cores per CPU**)



Multiple communicating rings

Benchmark MPI/tasks/C/halo-benchmarks/halo_irecv_send_multiplelinks_toggle.c

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



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 Bill Gropp's solution**

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>.

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 Bill Gropp's solution**

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/nodcart-final.pdf>.

2. The existing MPI-4.1 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 data mesh sizes
or direction-dependent bandwidth

- by MPIX_Cart_weighted_create(... weights)

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 Bill Gropp's solution**

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>.

2. The existing MPI-4.1 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 data mesh sizes
or direction-dependent bandwidth

- by MPIX_Cart_weighted_create(... weights)

3. Caution: The application must be prepared for rank re-numbering

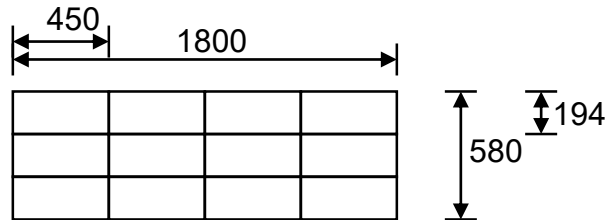
- All communication through the newly created
Cartesian communicator with re-numbered ranks!
- One must not load data based on MPI_COMM_WORLD ranks!

Examples

- Application topology awareness

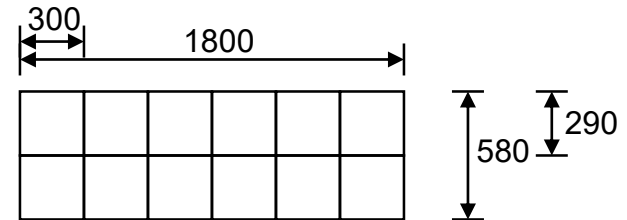
- 2-D example with 12 MPI processes and data mesh size 1800x580

- **MPI_Dims_create** → 4x3



Boundary of a subdomain = $2(450+194) = 1288$ 😞

- **data mesh aware** → 6x2 processes



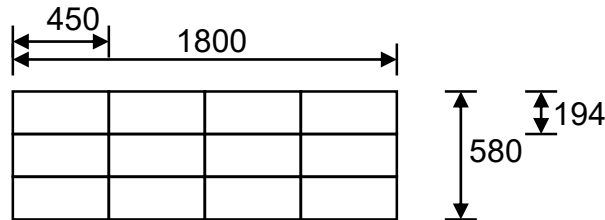
Boundary of a subdomain = $2(300+290) = 1180$ 😊

Examples

- Application topology awareness

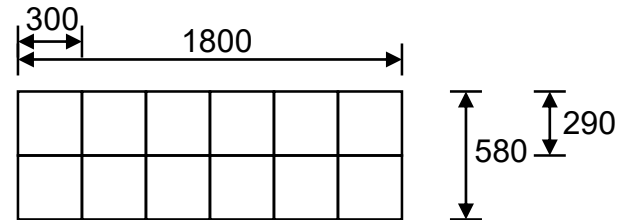
- 2-D example with 12 MPI processes and data mesh size 1800x580

- **MPI_Dims_create** → 4x3



Boundary of a subdomain = $2(450+194) = 1288$ 😞

- **data mesh aware** → 6x2 processes

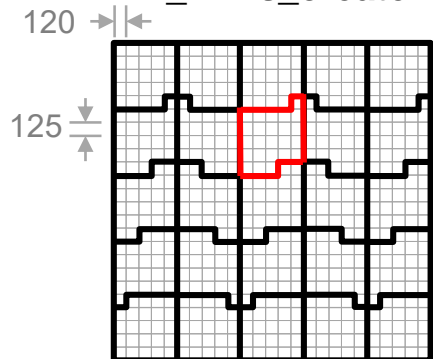


Boundary of a subdomain = $2(300+290) = 1180$ 😊

- Hardware topology awareness

- 2-D example with 25 nodes x 24 cores and data mesh size 3000x3000

- **MPI_Dims_create** → 25 x 24

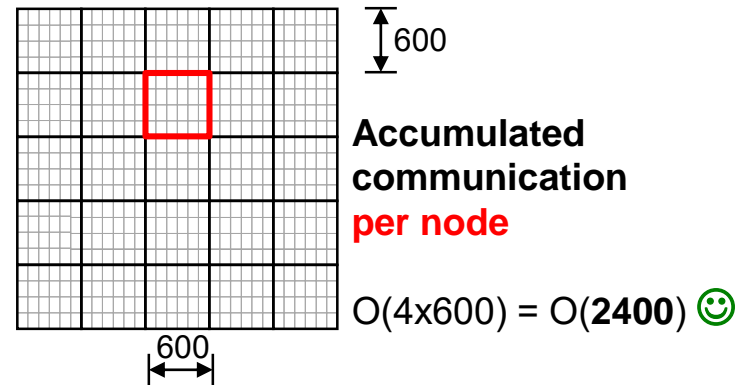


Accumulated communication per node

$O(10 \times 120 + 12 \times 125)$
 $= O(2700)$ 😞

- **Hardware aware** → 30 x 20

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



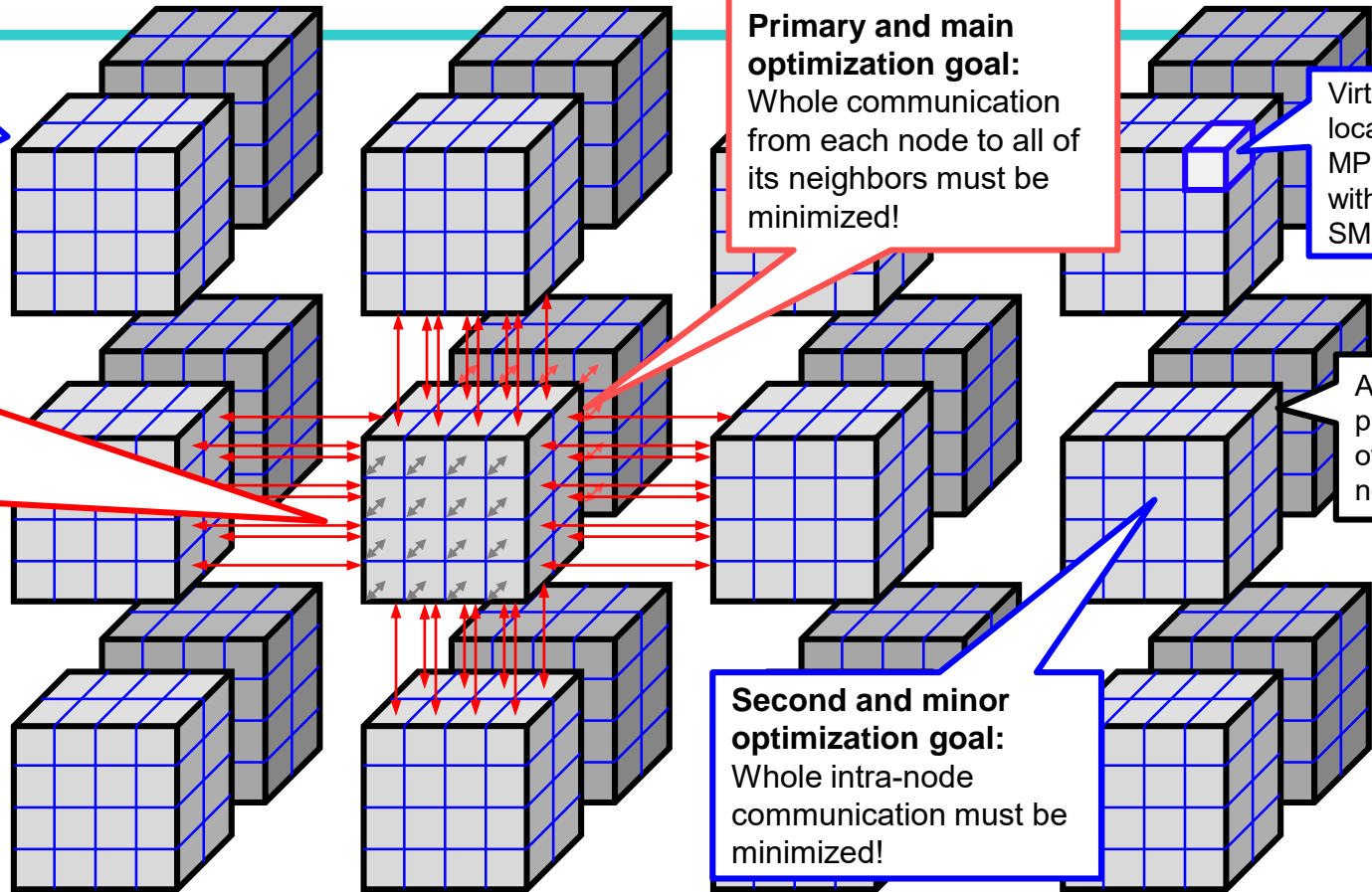
Accumulated communication per node

$O(4 \times 600) = O(2400)$ 😊

Hierarchical Cartesian Domain Decomposition

Example:
24 SMP nodes
X
32 cores/node

Per node:
maximal
 $8+8+8+8+16+16^*)=$
48 or 64^{*)}
connections
to neighbor
nodes
*) with cyclic communication



Primary and main optimization goal:
Whole communication from each node to all of its neighbors must be minimized!

Virtual location of an MPI process within an SMP node

All MPI processes of an SMP node

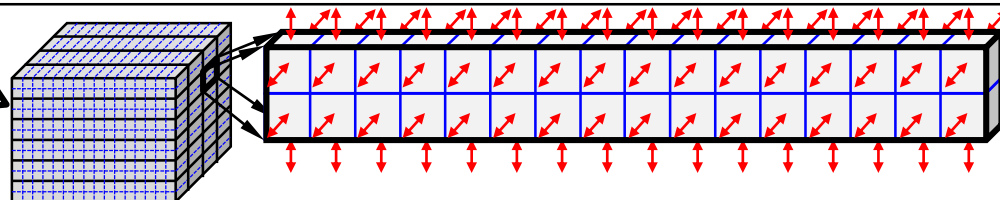
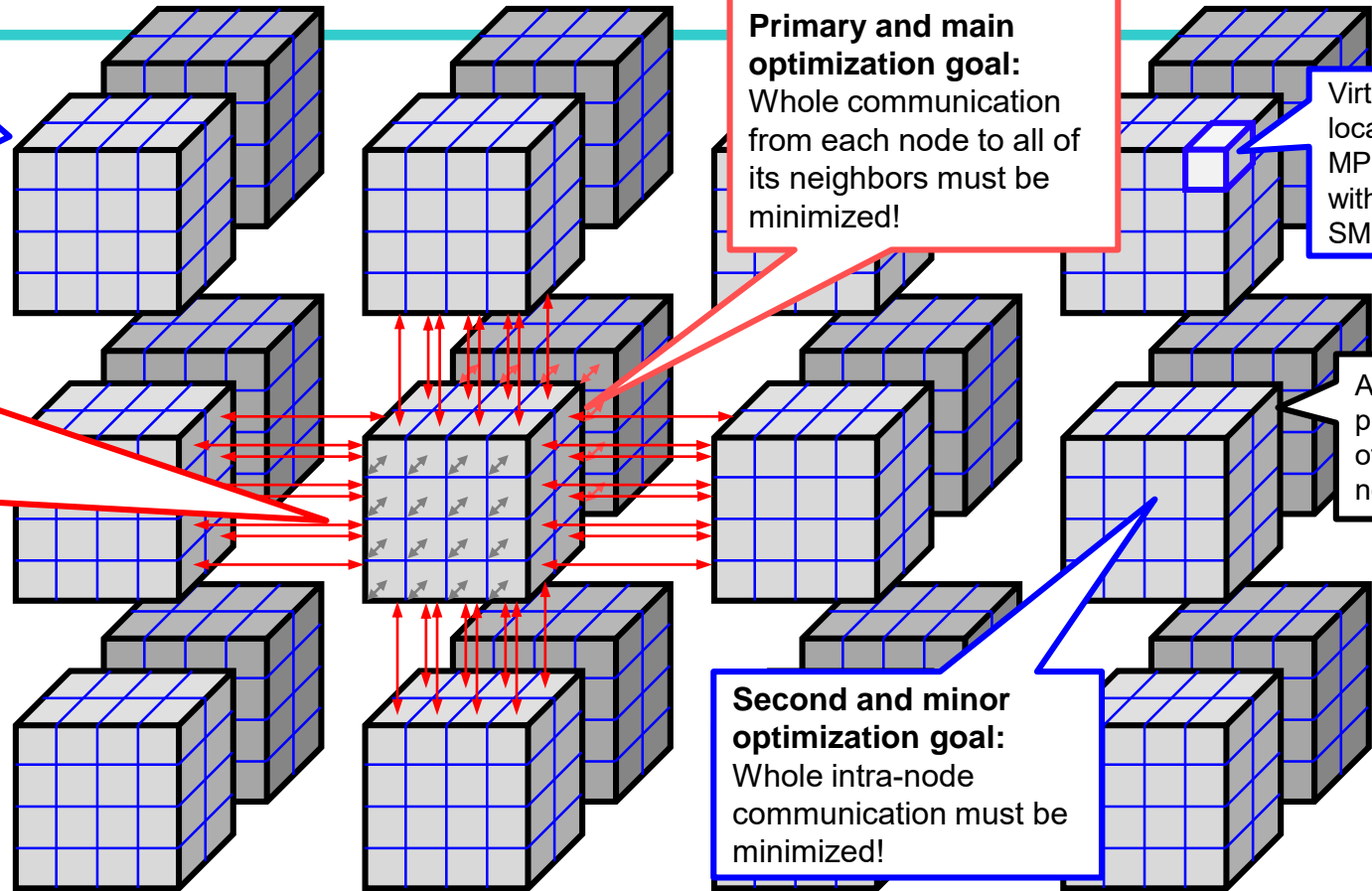
Second and minor optimization goal:
Whole intra-node communication must be minimized!

Hierarchical Cartesian Domain Decomposition

Example:
24 SMP nodes
X
32 cores/node

Per node:
maximal
 $8+8+8+8+16+16^{*})=$
48 or 64^{*)}
connections
to neighbor
nodes
*) with cyclic communication

Without
topology-
optimization:
96 connections
to other nodes

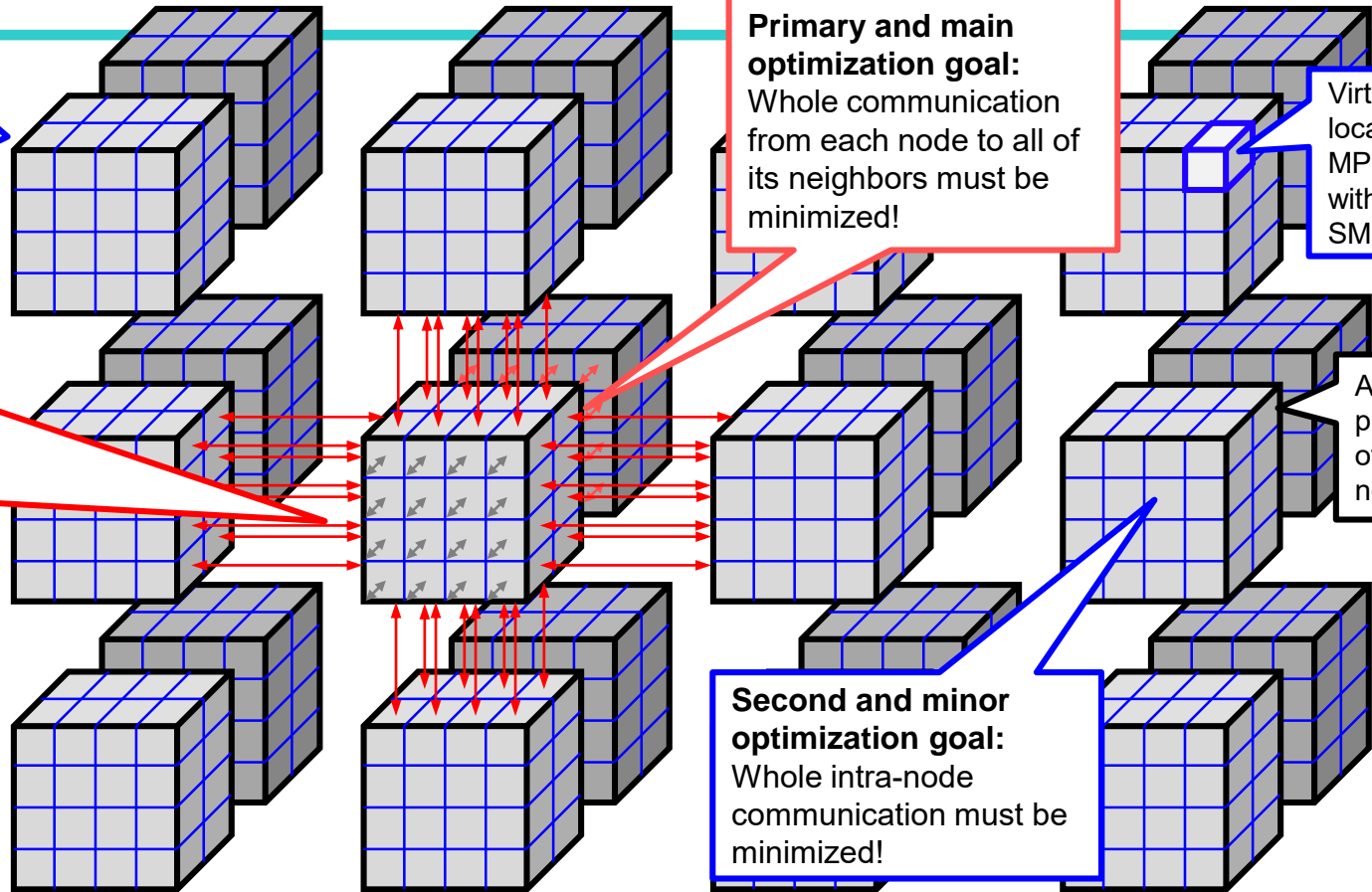


Hierarchical Cartesian Domain Decomposition

Example:
24 SMP nodes
X
32 cores/node

Per node:
maximal
 $8+8+8+8+16+16^{*)} =$
48 or 64^{*)}
connections
to neighbor
nodes
*) with cyclic communication

Without
topology-
optimization:
96 connections
to other nodes

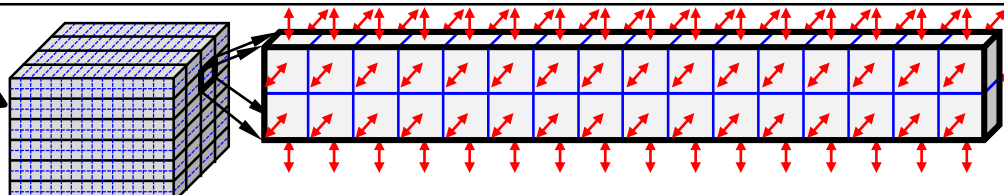


Primary and main optimization goal:
Whole communication from each node to all of its neighbors must be minimized!

Virtual location of an MPI process within an SMP node

All MPI processes of an SMP node

Second and minor optimization goal:
Whole intra-node communication must be minimized!



2 or 1.6^{*)} times slower communication

Goals of MPI_Dims_create + MPI_Cart_create

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

The limits of MPI_Dims_create + MPI_Cart_create

- Not application topology aware
 - MPI_Dims_create can **only** map **evenly balanced** Cartesian topologies
 - Factorization of 48,000 processes into 20 x 40 x 60 processes (e.g. for a mesh with 200 x 400 x 600 mesh points)
 - no chance with current interface

The limits of MPI_Dims_create + MPI_Cart_create

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

The limits of MPI_Dims_create + MPI_Cart_create

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

Major problems:

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

Goals of Cartesian MPI_Dims + Cart_create

- Remark: On a hierarchical hardware,
 - **optimized factorization and reordering** typically means **minimal node-to-node** communication,
 - which typically means that the communicating surfaces of the data on each node is as quadratic¹⁾ as possible (or the subdomain as cubic¹⁾ as possible)
- ¹⁾ “quadratic” and “cubic” may be qualified due to different communication bandwidth in each direction caused by sending (fast) non-strided or (slow) strided data
- The current API, i.e.,
 - due to the missing weights
 - and the non-hardware aware MPI_Dims_create,does **not** allow such an optimized factorization and reordering in many cases.

The new interface – proposed for MPI-4.1

- **MPI_Dims_create_weighted** (

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

input for application-topology-awareness

- Arguments have same meaning as in `MPI_Dims_create`

- Goal (in absence of an info argument):

- `dims[i]·dim_weights[i]` should be as close as possible,
- i.e., the $\sum_{i=0..(ndims-1)} \text{dims}[i] \cdot \text{dim_weights}[i]$ as small as possible (advice to implementors)

The new interface – proposed for MPI-4.1

- **MPI_Dims_create_weighted** (

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

input for application-topology-awareness

- Arguments have same meaning as in MPI_Dims_create

- Goal (in absence of an info argument):

- $\text{dims}[i] \cdot \text{dim_weights}[i]$ should be as close as possible,
- i.e., the $\sum_{i=0..(ndims-1)} \text{dims}[i] \cdot \text{dim_weights}[i]$ as small as possible (advice to implementors)

A new
courtesy
function:
**Weighted
factorization**

The new interface – proposed for MPI-4.1, continued

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

input for hardware-awareness

input for application-topology-awareness

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

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

The new interface – proposed for MPI-4.1, continued

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

input for hardware-awareness

input for application-topology-awareness

The new application & hardware topology aware interface

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

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

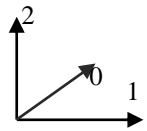
How to specify the dim_weights?

- Given: comm_old (e.g., MPI_COMM_WORLD), ndims (e.g., 3 dimensions)
- This means, **the domain decomposition has not yet taken place!**
- Goals for dim_weights and the API at all:
 - Easy to understand
 - Easy to calculate
 - Relevant for typical Cartesian communication patterns (MPI_Neighbor_alltoall or alternatives)
 - Rules fit to usual design criteria of MPI
 - E.g., reusing MPI_UNWEIGHTED → integer array
 - Can be enhanced by vendors for their platforms → additional info argument for further specification
 - To provide also the less optimal two stage interface (in addition to the combined routine)

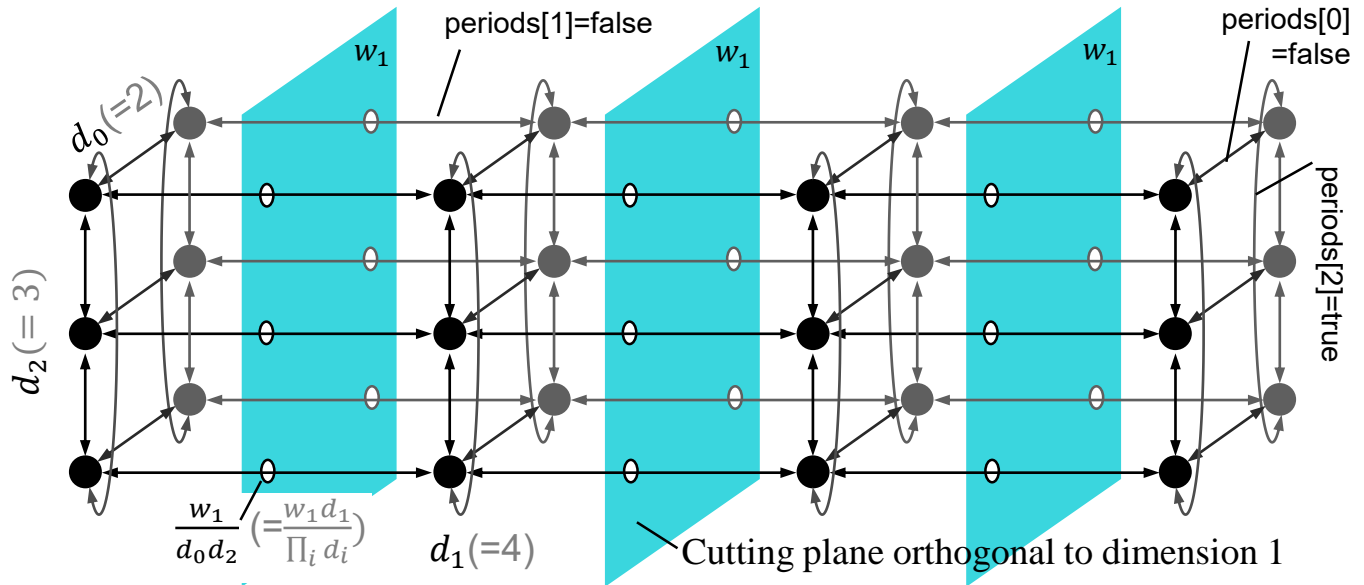
The dim_weights[i], example with 3 dimensions

Abbreviations:

$d_i = \text{dims}[i]$
 $w_i = \text{dim_weights}[i]$
 with
 $i = 0..(\text{ndims}-1)$



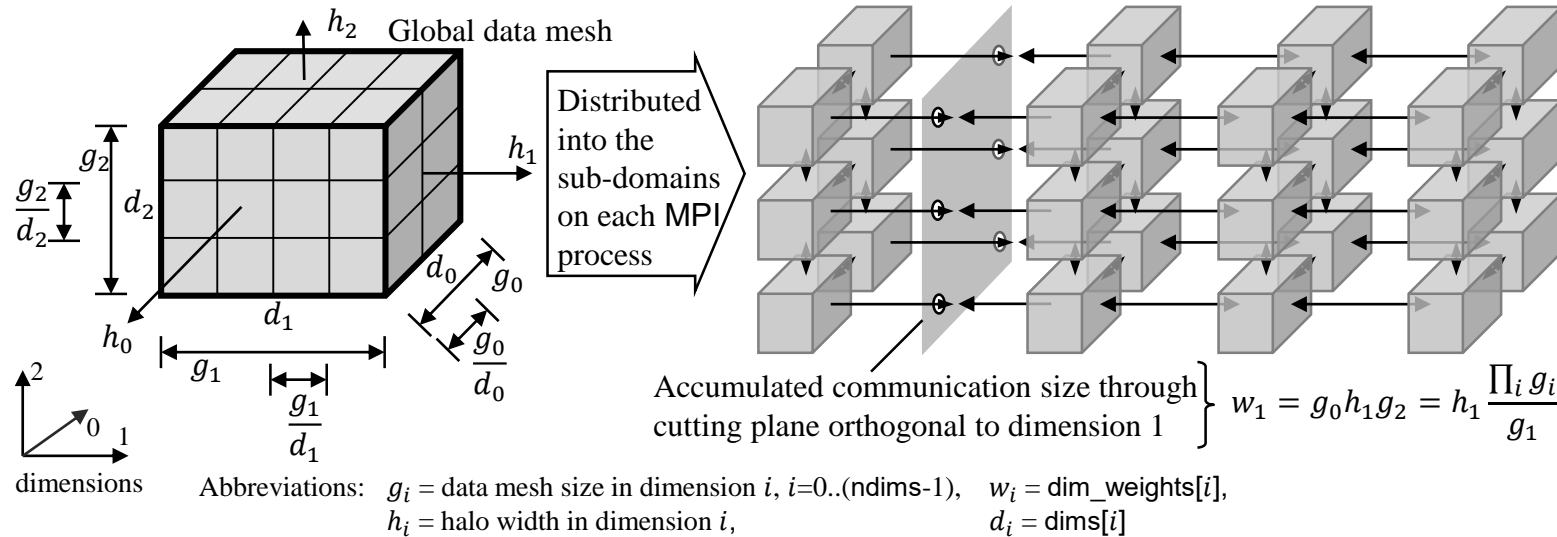
Three dimensions,
 i.e., ndims=3



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

¹⁾ If the communication bandwidth is different in each direction i , then w_i should be divided by the expected communication bandwidth.

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

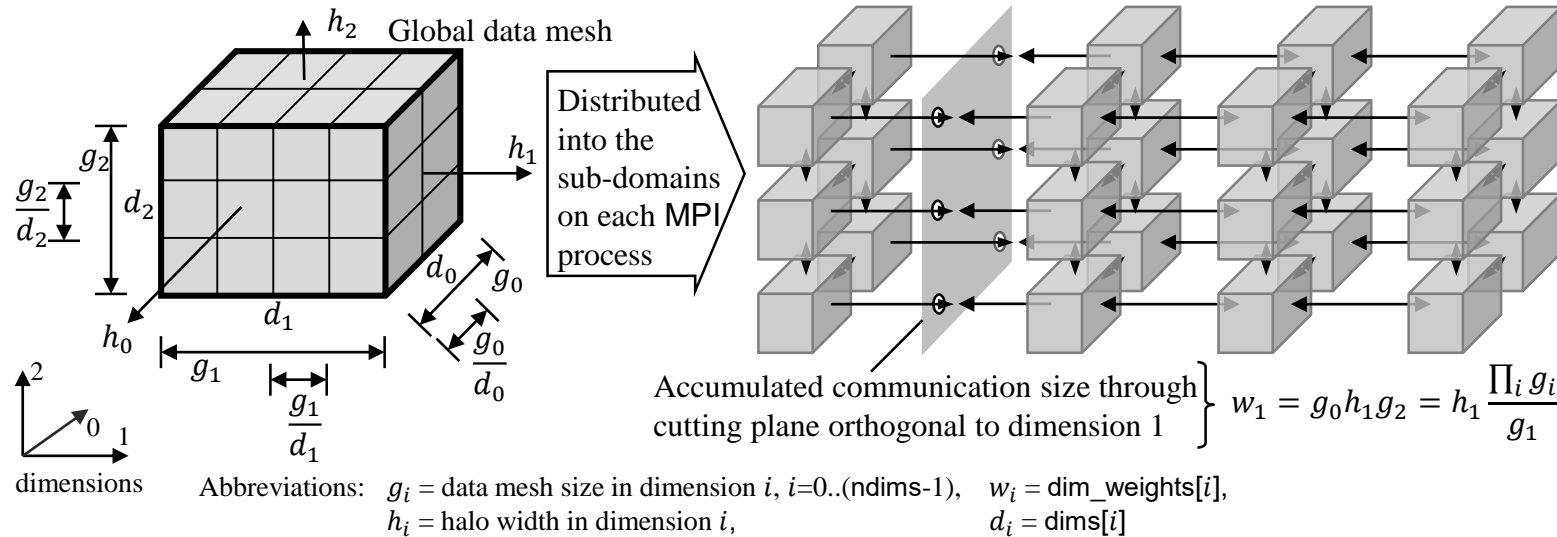


Example for the calculation of the accumulated communication size $w_{i,i=0..2}$ in each dimension.

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

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

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



Important:

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

- Result¹⁾ was

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

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

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

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

¹⁾ If the communication bandwidth is different in each direction i , then w_i should be divided by the expected communication bandwidth.

Simple answers to our problems / examples

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

The new interfaces – a real implementation

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: (in MPI/tasks/C/Ch9/MPIX/)

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

The new interfaces – a real implementation

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: (in MPI/tasks/C/Ch9/MPIX/)

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

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*);

Further Interfaces

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

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.

Full paper:

- Christoph Niethammer, Rolf Rabenseifner:
An MPI interface for application and hardware aware cartesian topology optimization.
EuroMPI 2019. Proceedings of the 26th European MPI Users' Group Meeting, September 2019, article No. 6, pages 1-8, <https://doi.org/10.1145/3343211.3343217>

Further Interfaces

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

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.

Full paper:

- Christoph Niethammer, Rolf Rabenseifner:
An MPI interface for application and hardware aware cartesian topology optimization.
EuroMPI 2019. Proceedings of the 26th European MPI Users' Group Meeting, September 2019, article No. 6, pages 1-8, <https://doi.org/10.1145/3343211.3343217>

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) additionally 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 w_i based on $\frac{G}{g_i}$, it is important
 - that the data of the mesh 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

skipped

Typical use of MPIX_Cart_weighted_create

```
#define ndims 3
int i, nnodes, world_myrank, cart_myrank, dims[ndims], periods[ndims], my_coords[ndims];
int global_array_dim[ndims], halo_width[ndims], local_array_dim[ndims], local_array_size=1;
double dim_weights[ndims], global_array_size=1.0;
MPI_Comm comm_cart;
MPI_Init(NULL, NULL);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &world_myrank);
for (i=0; i<ndims; i++) {
    dims[i]=0; periods[i]=...;
    global_array_dim[i]=...; halo_width[i]=...;
    global_array_size = global_array_size * (double)(global_array_dim[i]);
}
for (i=0; i<ndims; i++) {
    dim_weights[i] = (double)(halo_width[i]) * global_array_size / (double)(global_array_dim[i]);
}
MPIX_Cart_weighted_create(MPI_COMM_WORLD, ndims, dim_weights, dims, periods, MPI_INFO_NULL, dims,
                        &comm_cart);

MPI_Comm_rank(comm_cart, &cart_myrank);
MPI_Cart_coords(comm_cart, cart_myrank, ndims, my_coords, ierror)
```

$$\text{Weights: } w_i = h_i \frac{\prod_j g_j}{g_i}$$

```
for (i=0; i<ndims; i++) {
    local_array_dim[i] = global_array_dim[i] / dims[i];
    local_array_dim[i] ... adjust it if the division has a remainder
    local_array_size = local_array_size * local_array_dim[i];
}
local_data_array = malloc(sizeof(...) * local_array_size);
```

From now on:

- all communication should be based on **comm_cart** & **cart_myrank** & **my_coords**
- one can setup the sub-domains & read in the application data

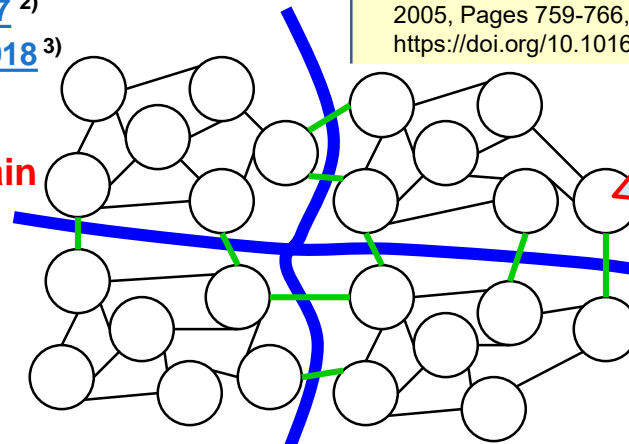
Unstructured Grid / Data Mesh

Mesh partitioning with special load balancing libraries

- Metis & ParMetis (George Karypis, University of Minnesota)
 - <http://glaros.dtc.umn.edu/gkhome/views/metis/metis.html>
 - Scotch & PT-Scotch (Francois Pellegrini, LaBRI, France)
 - <https://www.labri.fr/perso/pelegrin/scotch/>
 - Alternative partitioning via space-filling curves, e.g.,
- Goals:

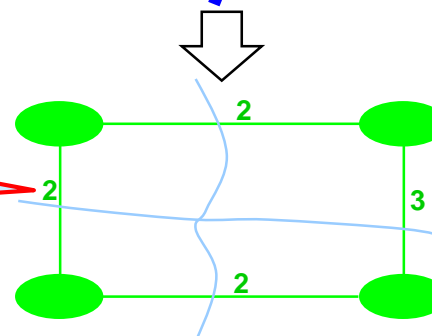
- **Same work load in each sub-domain**
- **Minimizing the maximal number of neighbor-connections between sub-domains**
- **Minimizing the total number of neighbor sub-domains of each sub-domain**

- 1) Ricard Borrell, Juan Carlos García Cajas, Daniel Mira, Ahmed Taha, Seid Koric, et al.. Parallel mesh partitioning based on space filling curves. Computers and Fluids, 2018, 173, pp.264-272. [ff10.1016/j.compfluid.2018.01.040](https://doi.org/10.1016/j.compfluid.2018.01.040) [ffhal-01969026f](https://doi.org/10.1016/j.compfluid.2018.01.040)
- 2) D. F. Harlacher, H. Klimach, S. Roller, C. Siebert and F. Wolf, "Dynamic Load Balancing for Unstructured Meshes on Space-Filling Curves," 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, Shanghai, China, 2012, pp. 1661-1669, doi: 10.1109/IPDPSW.2012.207.
- 3) Stefan Schamberger, Jens-Michael Wierum, Partitioning finite element meshes using space-filling curves, Future Generation Computer Systems, Volume 21, Issue 5, 2005, Pages 759-766, ISSN 0167-739X, <https://doi.org/10.1016/j.future.2004.05.018>.



Result of mesh partitioning:
Sort out all mesh elements into sub-domains

The weighted communication graph of the virtual process grid can be used as input for `MPI_Dist_graph_create(_adjacent)`



Each sub-domain is stored on one MPI process

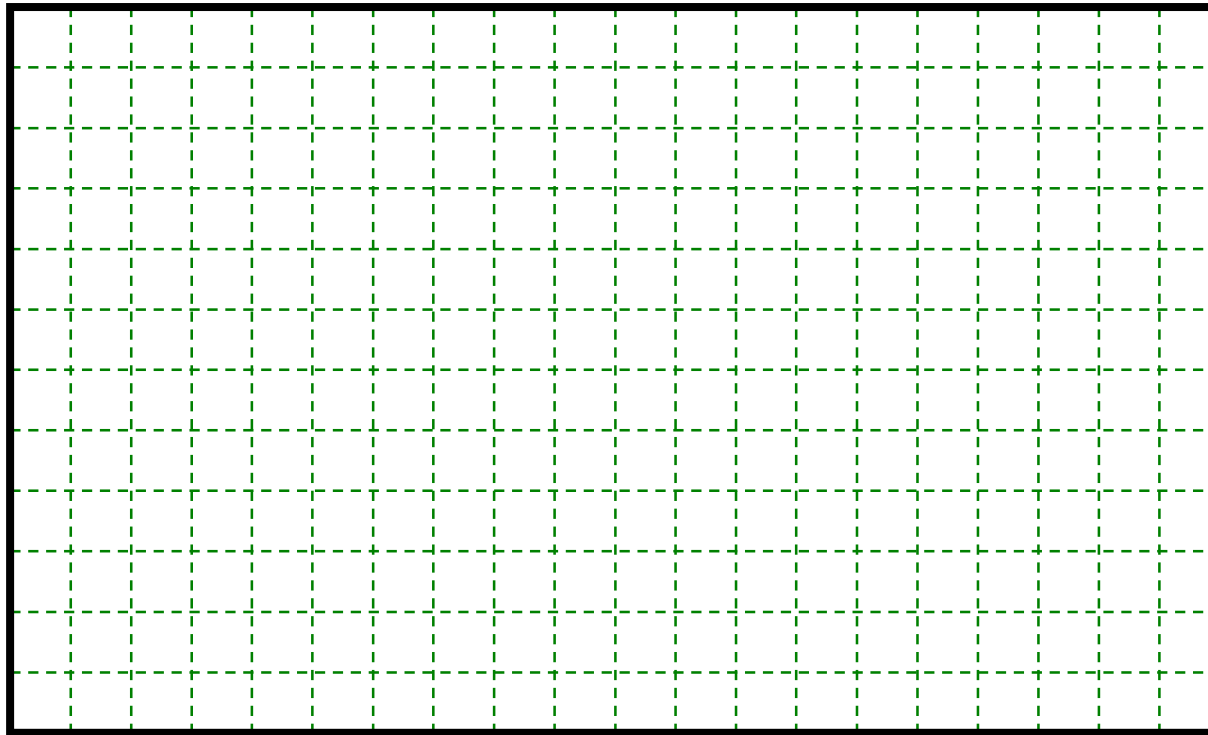
Unstructured Grid / Data Mesh – Multi-level Domain Decomposition through Recombination



Unstructured Grid / Data Mesh – Multi-level Domain Decomposition through Recombination

1. Core-level DD: partitioning of (large) application's data grid —

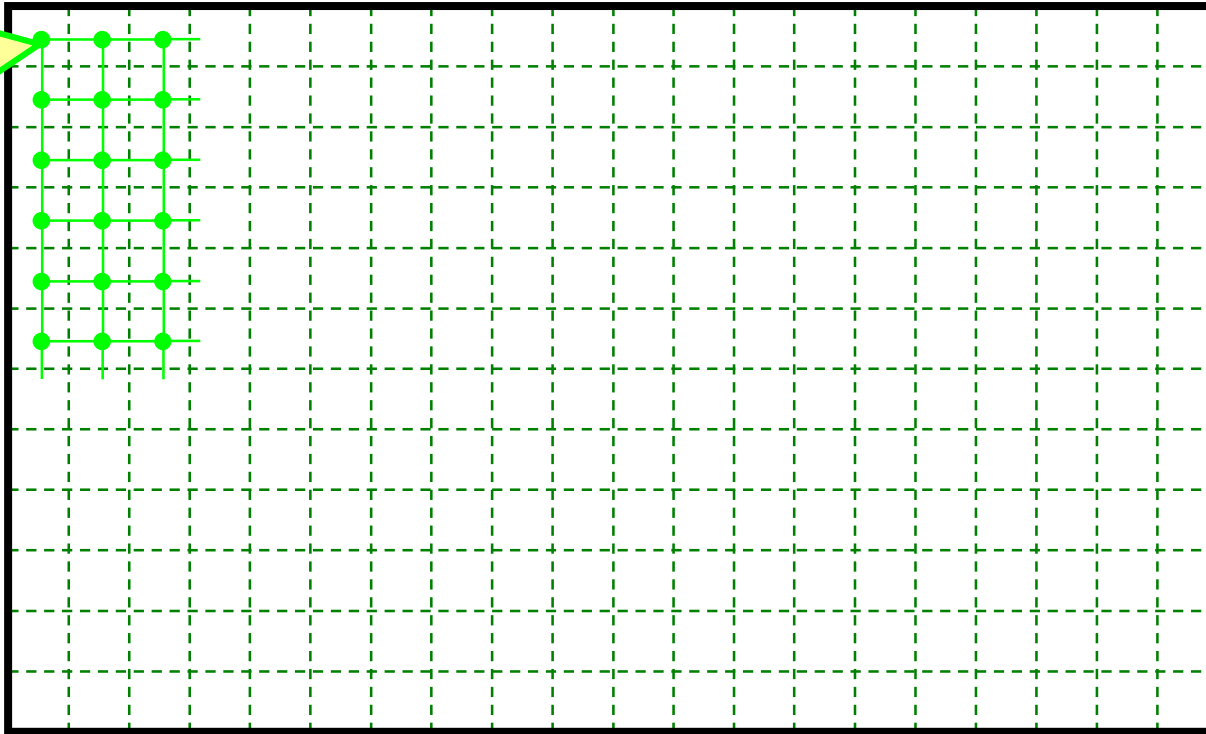
e.g., with Metis / Scotch or
via space-filling curves



Unstructured Grid / Data Mesh – Multi-level Domain Decomposition through Recombination

1. **Core-level DD:** partitioning of (large) application's data grid — e.g., with Metis / Scotch or via space-filling curves

Graph of all sub-domains (core-sized)

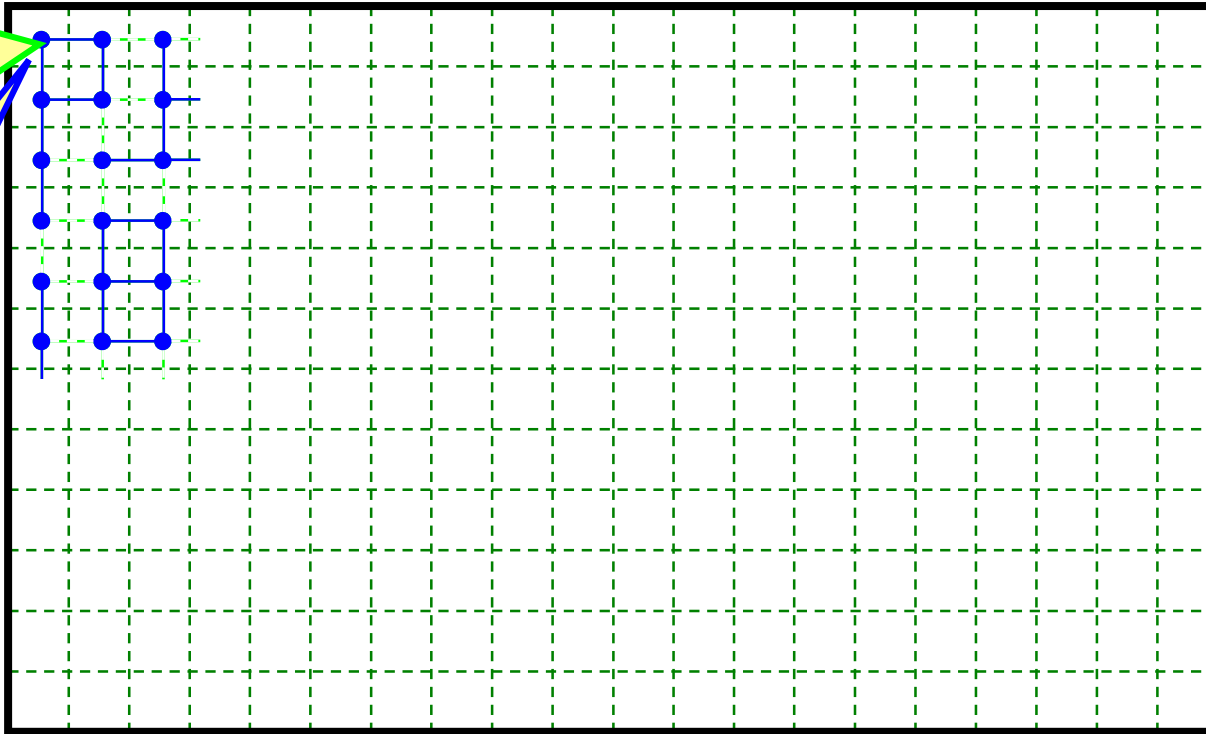


Unstructured Grid / Data Mesh – Multi-level Domain Decomposition through Recombination

1. Core-level DD: partitioning of (large) application's data grid — e.g., with Metis / Scotch or via space-filling curves

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket

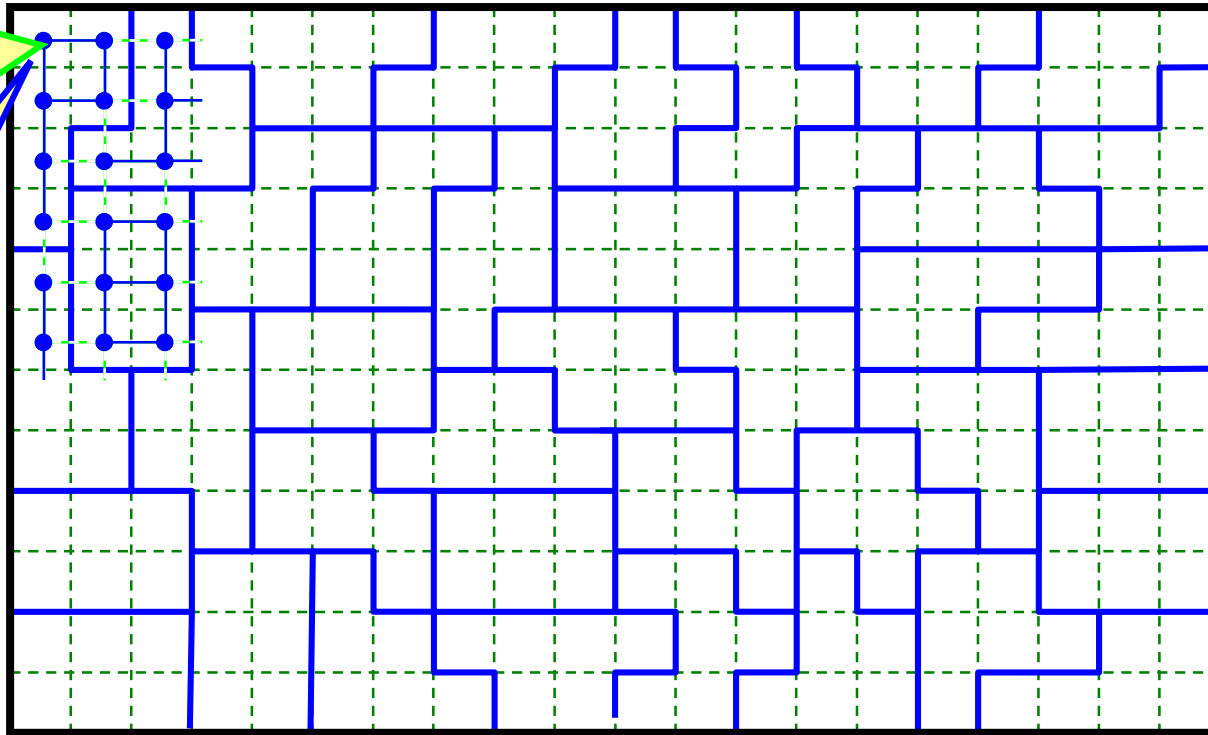


Unstructured Grid / Data Mesh – Multi-level Domain Decomposition through Recombination

1. Core-level DD: partitioning of (large) application's data grid
 2. Numa-domain-level DD: recombining of core-domains
- e.g., with Metis / Scotch or via space-filling curves

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket

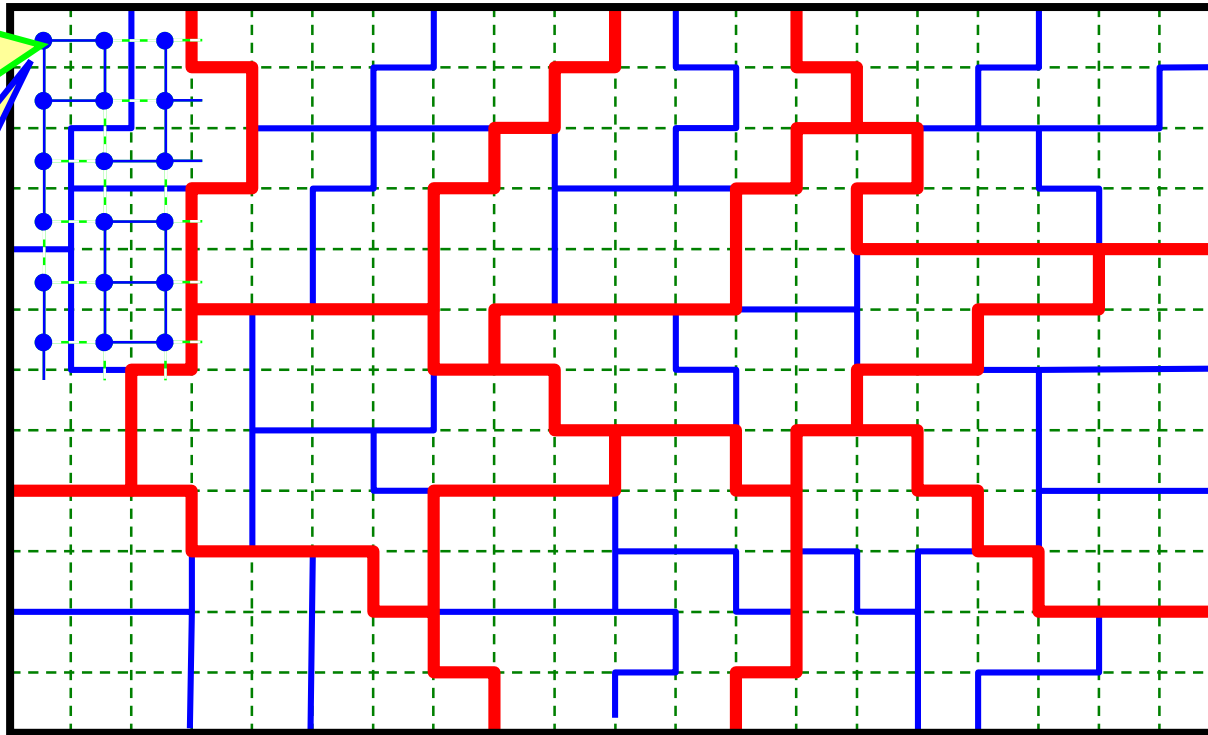


Unstructured Grid / Data Mesh – Multi-level Domain Decomposition through Recombination

1. **Core-level DD:** partitioning of (large) application's data grid — e.g., with Metis / Scotch or via space-filling curves
2. **Numa-domain-level DD:** recombining of core-domains
3. **SMP node level DD:** recombining of socket-domains

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket



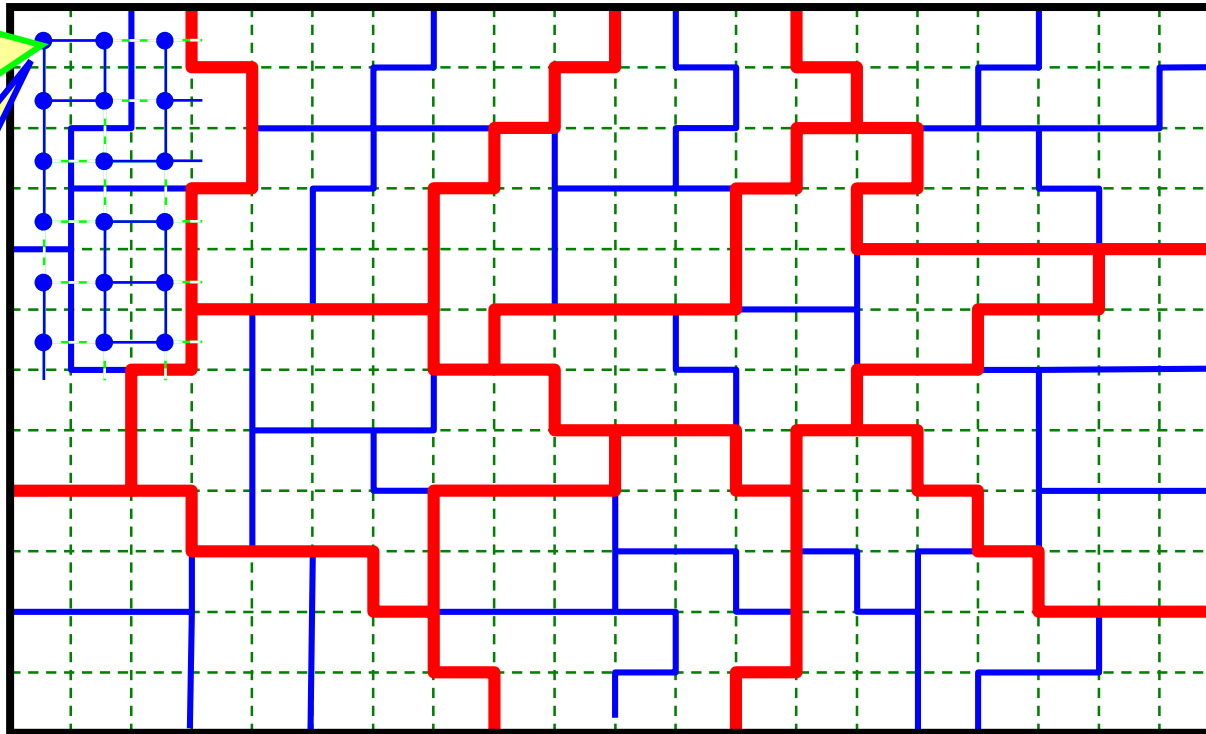
Unstructured Grid / Data Mesh –

Multi-level Domain Decomposition through Recombination

1. **Core-level DD:** partitioning of (large) application's data grid
 2. **Numa-domain-level DD:** recombining of core-domains
 3. **SMP node level DD:** recombining of socket-domains
- e.g., with Metis / Scotch or via space-filling curves

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket



- **Problem:** Recombination must **not** calculate patches that are smaller or larger than the average
- In this example the load-balancer (e.g., Metis or Scotch) **must** combine always
 - 6 cores, and
 - 4 numa-domains (i.e., sockets or dies)
- **Advantage:** Communication is balanced!

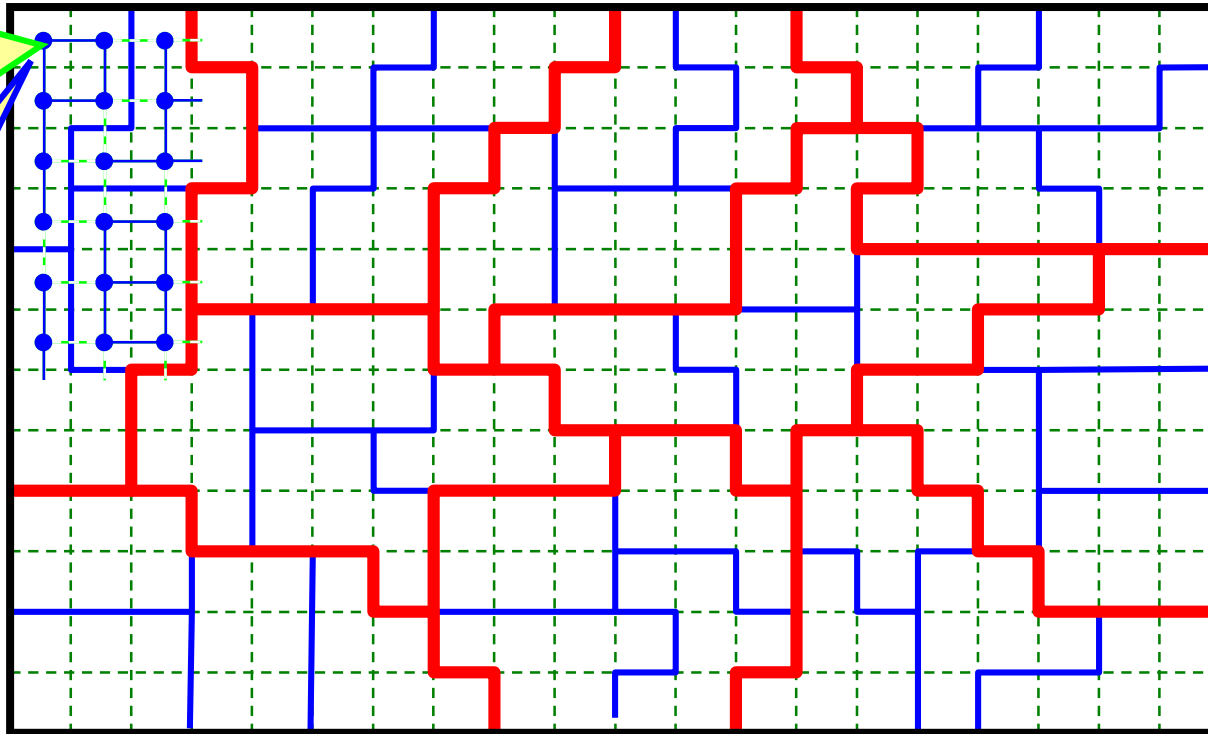
Unstructured Grid / Data Mesh –

Multi-level Domain Decomposition through Recombination

1. **Core-level DD:** partitioning of (large) application's data grid
 2. **Numa-domain-level DD:** recombining of core-domains
 3. **SMP node level DD:** recombining of socket-domains
 4. **Numbering** from core to socket to node as done in `MPI_COMM_WORLD` (e.g., sequentially)
- e.g., with Metis / Scotch or via space-filling curves

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket



- **Problem:** Recombination must **not** calculate patches that are smaller or larger than the average
- In this example the load-balancer (e.g., Metis or Scotch) **must** combine always
 - 6 cores, and
 - 4 numa-domains (i.e., sockets or dies)
- **Advantage:** Communication is balanced!

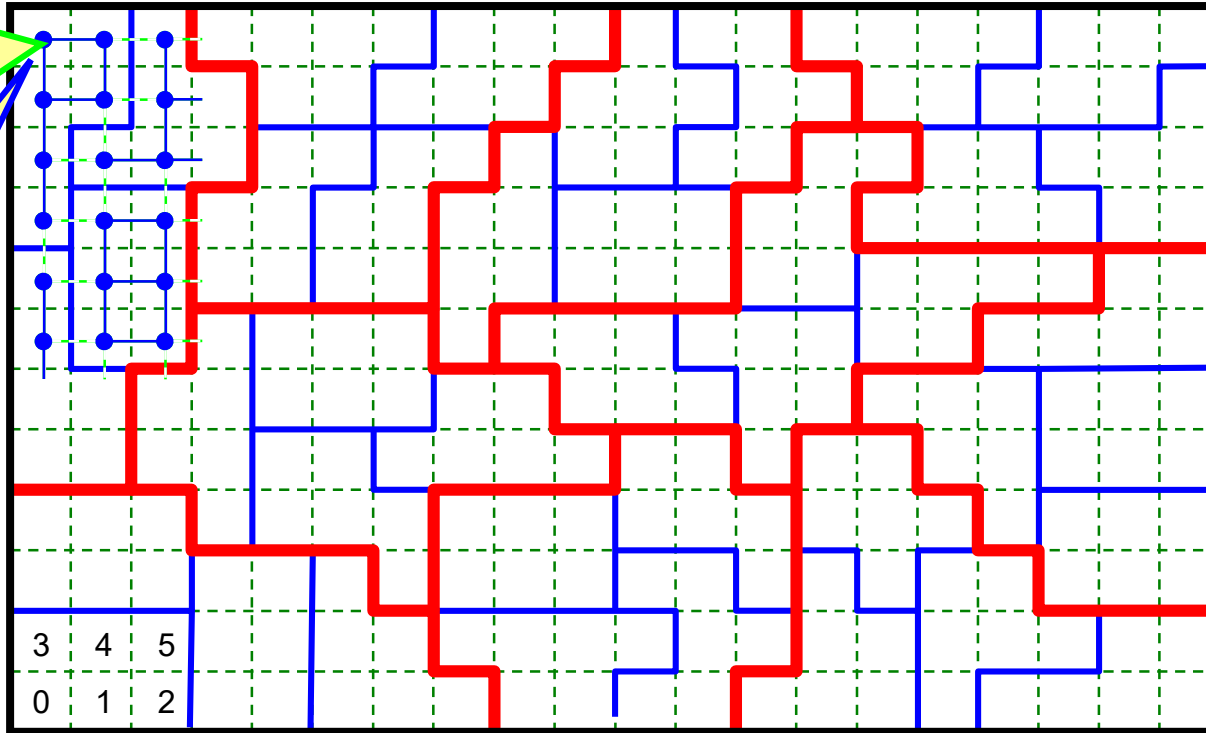
Unstructured Grid / Data Mesh –

Multi-level Domain Decomposition through Recombination

1. **Core-level DD:** partitioning of (large) application's data grid
 2. **Numa-domain-level DD:** recombining of core-domains
 3. **SMP node level DD:** recombining of socket-domains
 4. **Numbering** from core to socket to node as done in `MPI_COMM_WORLD` (e.g., sequentially)
- e.g., with Metis / Scotch or via space-filling curves

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket



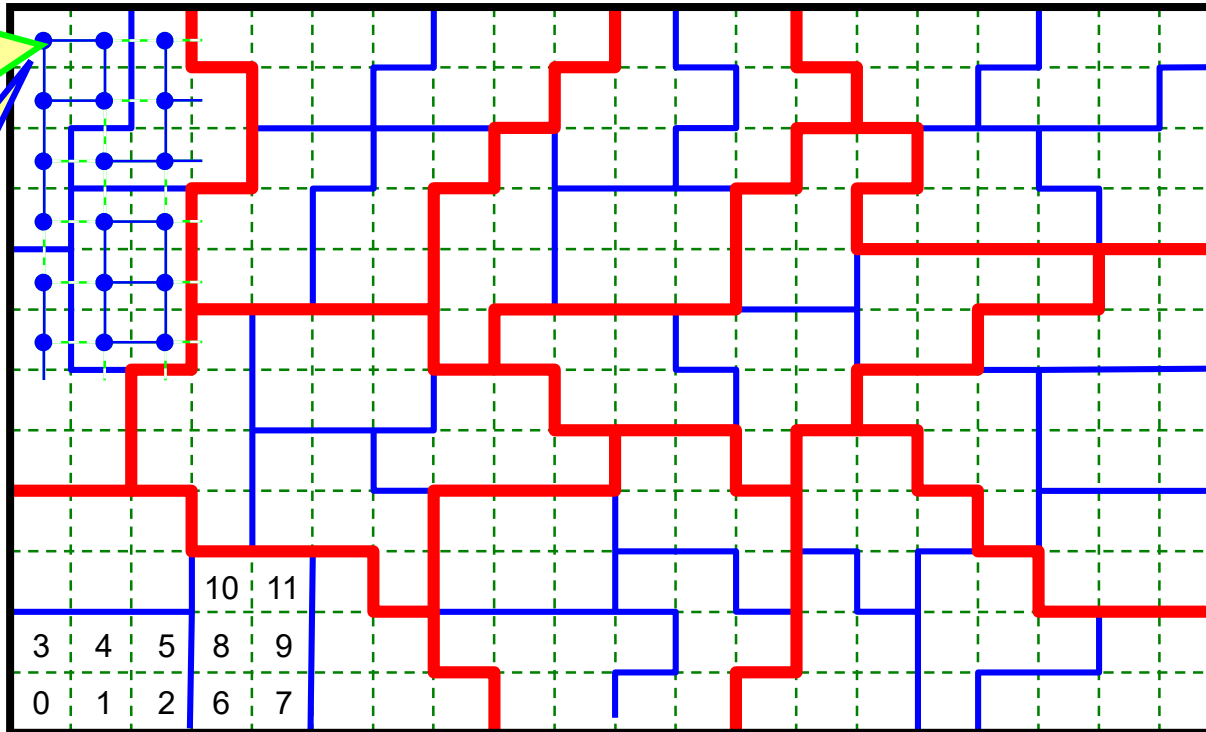
- **Problem:** Recombination must **not** calculate patches that are smaller or larger than the average
- In this example the load-balancer (e.g., Metis or Scotch) **must** combine always
 - 6 cores, and
 - 4 numa-domains (i.e., sockets or dies)
- **Advantage:** Communication is balanced!

Unstructured Grid / Data Mesh – Multi-level Domain Decomposition through Recombination

1. **Core-level DD:** partitioning of (large) application's data grid
 2. **Numa-domain-level DD:** recombining of core-domains
 3. **SMP node level DD:** recombining of socket-domains
 4. **Numbering** from core to socket to node as done in MPI_COMM_WORLD (e.g., sequentially)
- e.g., with Metis / Scotch or via space-filling curves

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket



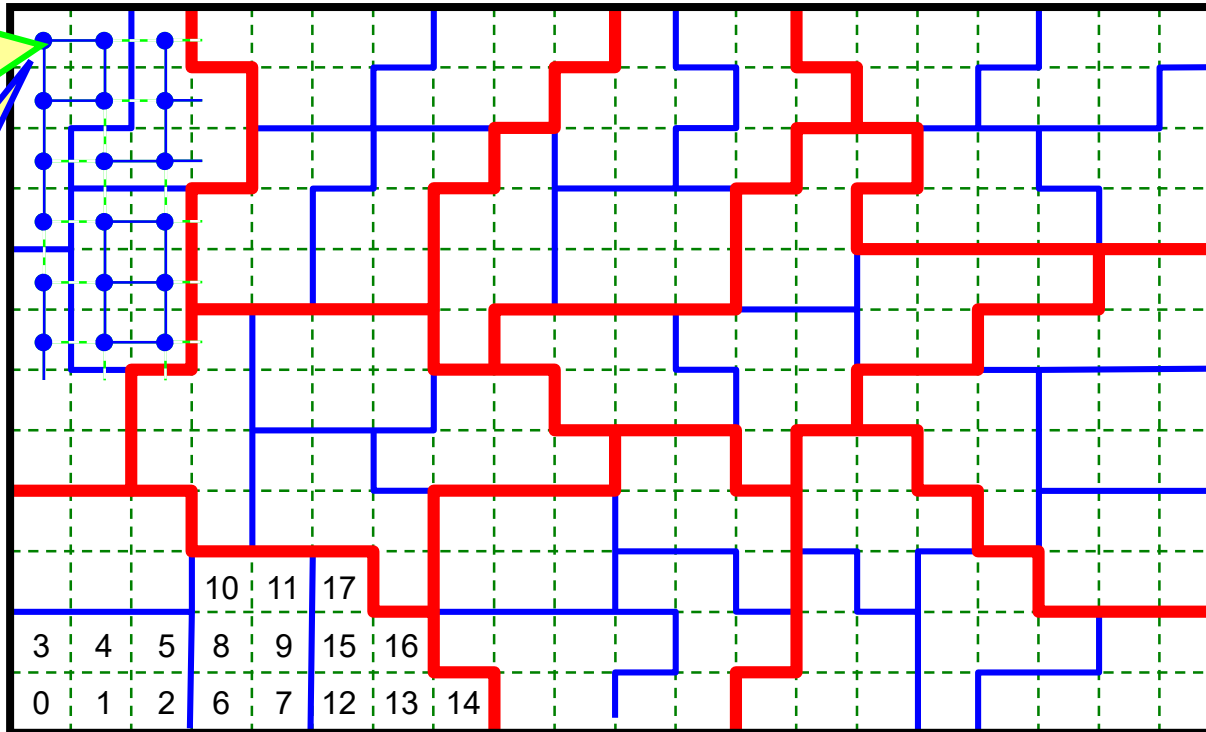
- **Problem:** Recombination must **not** calculate patches that are smaller or larger than the average
- In this example the load-balancer (e.g., Metis or Scotch) **must** combine always
 - 6 cores, and
 - 4 numa-domains (i.e., sockets or dies)
- **Advantage:** Communication is balanced!

Unstructured Grid / Data Mesh – Multi-level Domain Decomposition through Recombination

1. **Core-level DD:** partitioning of (large) application's data grid
 2. **Numa-domain-level DD:** recombining of core-domains
 3. **SMP node level DD:** recombining of socket-domains
 4. **Numbering** from core to socket to node as done in MPI_COMM_WORLD (e.g., sequentially)
- e.g., with Metis / Scotch or via space-filling curves

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket



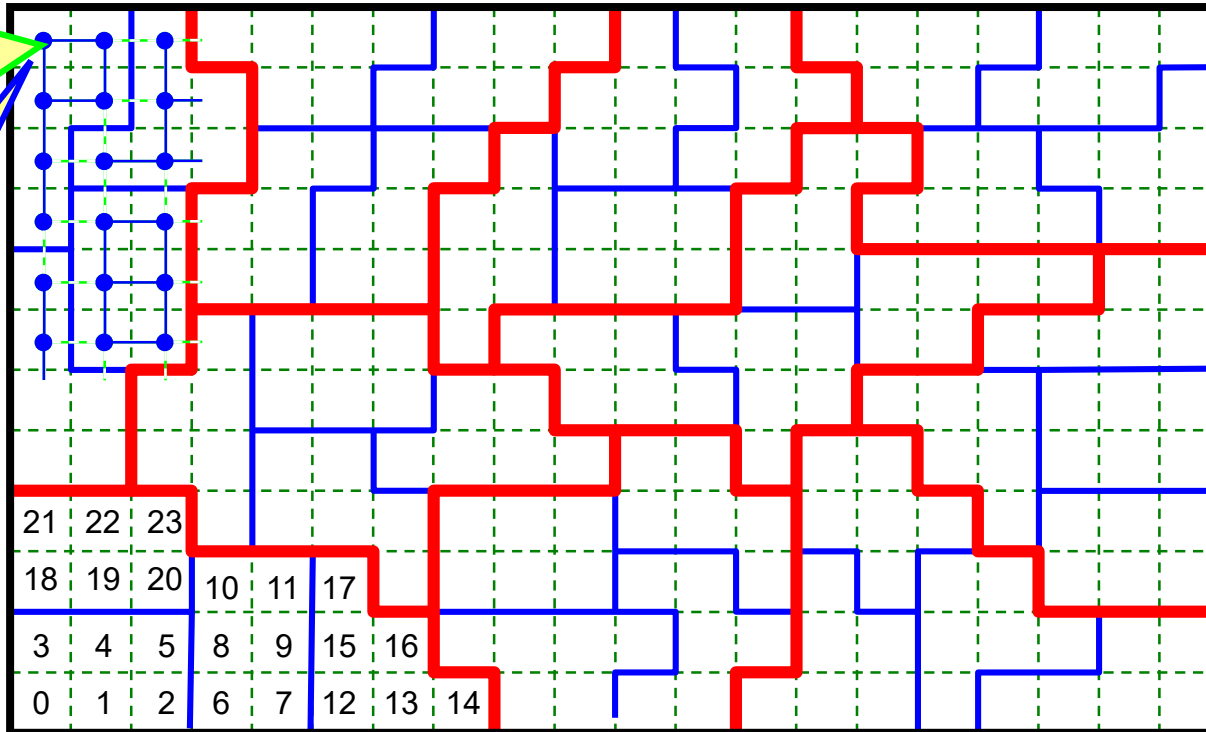
- **Problem:** Recombination must **not** calculate patches that are smaller or larger than the average
- In this example the load-balancer (e.g., Metis or Scotch) **must** combine always
 - 6 cores, and
 - 4 numa-domains (i.e., sockets or dies)
- **Advantage:** Communication is balanced!

Unstructured Grid / Data Mesh – Multi-level Domain Decomposition through Recombination

1. **Core-level DD:** partitioning of (large) application's data grid
 2. **Numa-domain-level DD:** recombining of core-domains
 3. **SMP node level DD:** recombining of socket-domains
 4. **Numbering** from core to socket to node as done in `MPI_COMM_WORLD` (e.g., sequentially)
- e.g., with Metis / Scotch or via space-filling curves

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket



- **Problem:** Recombination must **not** calculate patches that are smaller or larger than the average
- In this example the load-balancer (e.g., Metis or Scotch) **must** combine always
 - 6 cores, and
 - 4 numa-domains (i.e., sockets or dies)
- **Advantage:** Communication is balanced!

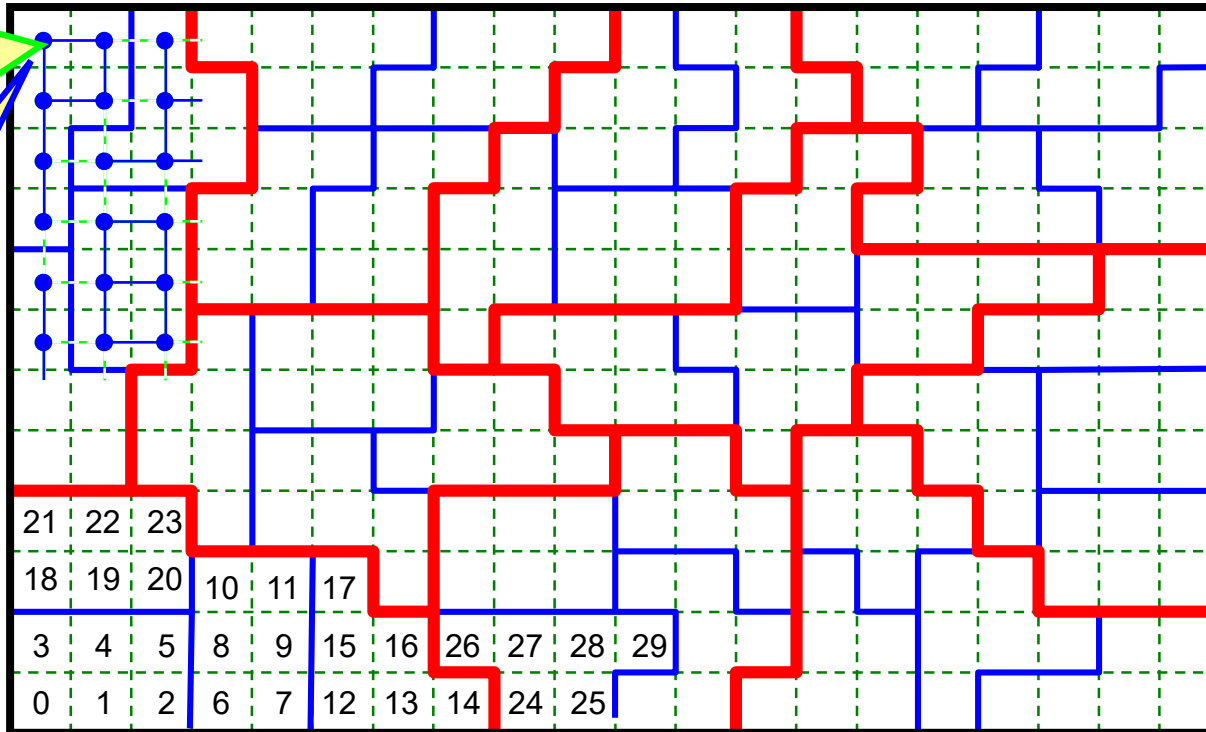
Unstructured Grid / Data Mesh –

Multi-level Domain Decomposition through Recombination

1. **Core-level DD:** partitioning of (large) application's data grid
 2. **Numa-domain-level DD:** recombining of core-domains
 3. **SMP node level DD:** recombining of socket-domains
 4. **Numbering** from core to socket to node as done in MPI_COMM_WORLD (e.g., sequentially)
- e.g., with Metis / Scotch or via space-filling curves

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket



- **Problem:** Recombination must **not** calculate patches that are smaller or larger than the average
- In this example the load-balancer (e.g., Metis or Scotch) **must** combine always
 - 6 cores, and
 - 4 numa-domains (i.e., sockets or dies)
- **Advantage:** Communication is balanced!

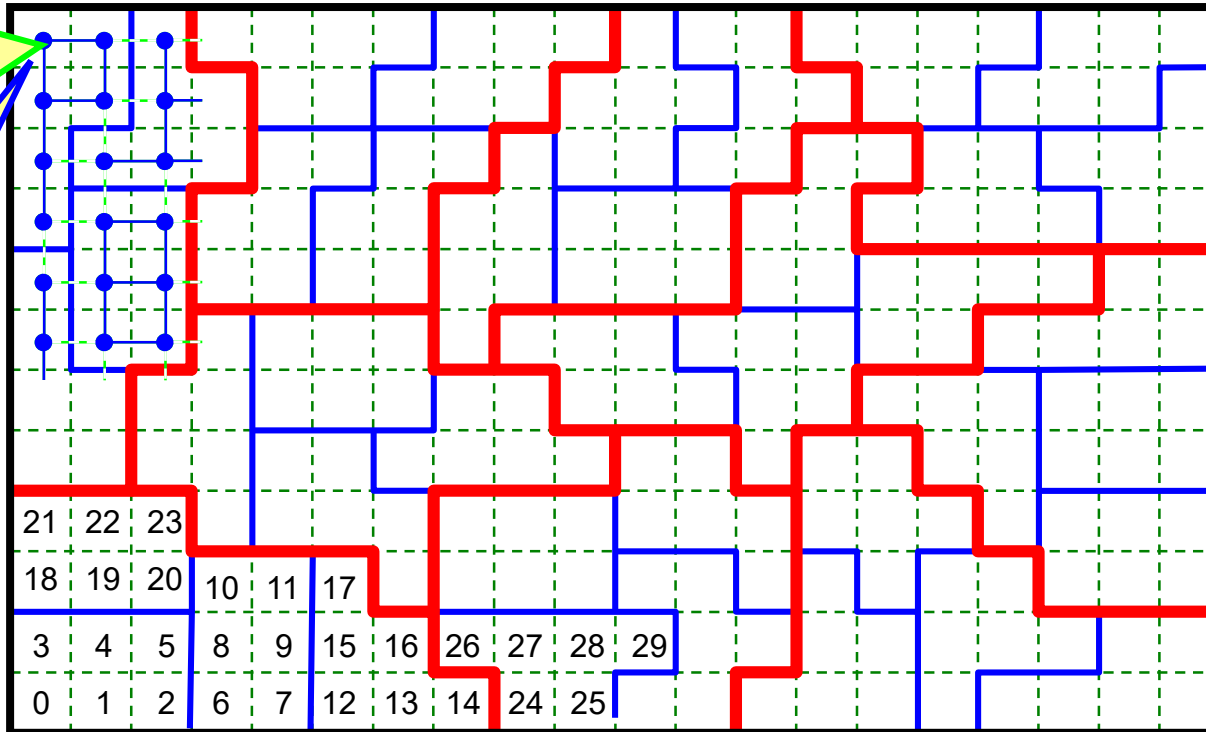
Unstructured Grid / Data Mesh –

Multi-level Domain Decomposition through Recombination

1. **Core-level DD:** partitioning of (large) application's data grid – e.g., with Metis / Scotch or via space-filling curves
2. **Numa-domain-level DD:** recombining of core-domains
3. **SMP node level DD:** recombining of socket-domains
4. **Numbering** from core to socket to node as done in MPI_COMM_WORLD (e.g., sequentially)

Graph of all sub-domains (core-sized)

Grouped into sub-graphs for each socket



5. **Subdomain data** that was read **before** the virtual graph topology was created needs **to be sent** to the appropriate process **after** reordering,

- **Problem:** Recombination must **not** calculate patches that are smaller or larger than the average
- In this example the load-balancer (e.g., Metis or Scotch) **must** combine always
 - 6 cores, and
 - 4 numa-domains (i.e., sockets or dies)
- **Advantage:** Communication is balanced!

Exercise:

Adding a Cartesian Topology

- This exercise is part of our **Hybrid MPI+X** course
- It is **not part of our MPI courses**, but
- we provide it here for you as a self-study exercise / example.

- Given: a 3-D halo communication benchmark using `irecv + send`
 - `cd MPI/tasks/C/Ch9/MPIX/`
 - `mpicc course/C/Ch9/halo_irecv_send_toggle_3dim_grid_skel.c` **MPIX*.c -lm**
 - The application uses a 3-D Cartesian communicator.
 - From this one, it uses 1-D line communicators for communicating in the 3 dimensions
- Overview on the to-do's:
 - “*substituting*” the not reordered Cartesian topology (`cart_method==1`) through an optimizing algorithm (`cart_method==2,3,4`)
 - `cart_method==2`: Add `MPIX_Cart...` (...`MPI_WEIGHTS_EQUAL...`)
 - `cart_method==3`: Calculate the weights based on `meshsize_avg_per_proc_startval`
Add `MPIX_Cart...` (...`weights...`)
 - `cart_method==4`: same as with `cart_method==3`, but without weights-calculation
 - Or just use `halo_irecv_send_toggle_3dim_grid.c` and look at the diff
 - `diff halo_irecv_send_toggle_3dim_grid_skel.c halo_irecv_send_toggle_3dim_grid.c`
 - Measure the communication bandwidth win
 - For default mesh size `2 / 2 / 2`
 - For other mesh sizes, e.g., `1 / 2 / 4`

My apologies for missing Fortran

See
/* TODO
lines

Exercise: Explanations

- Input per measurement, e.g. on 8 nodes x 2 CPUs x 12 cores:

Example
2

Column 1

- cart_method:
 - 1=Dims_create+Cart_create,
 - 2=Cart_weighted_create (MPIX_WEIGHTS_EQUAL),
 - 3=dito(weights),
 - 4=dito manually,
 - 5=Cart_ml_create(dims_ml),
 - 0=end of input

Columns 2-4

- Data mesh sizes, integer start values (= ratio)

1 2 4

Columns 5-10

- Using MPI_Type_vector, for each dimension a pair of blocklength&stride

0 0 0 0 0 0

Columns 11-13

- weights (double values) (only with cart_method==4)

1.00 0.50 0.25

Column 11

- 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

Columns 12-14

- dims_ml[d=0] =

1 2 2

15-17

- dims_ml[d=1] =

2 1 3

18-20

- dims_ml[d=2] =

4 1 2

Start a 8 or 12-node batch-job with your own input file:
Report your acceleration factors to the course group

Exercise: Explanations

- Input per measurement, e.g. on 8 nodes x 2 CPUs x 12 cores:

Example
2

Column 1

- cart_method:
 - 1=Dims_create+Cart_create,
 - 2=Cart_weighted_create (MPIX_WEIGHTS_EQUAL),
 - 3=dito(weights),
 - 4=dito manually,
 - 5=Cart_ml_create(dims_ml),
 - 0=end of input

Columns 2-4

- Data mesh sizes, integer start values (= ratio) 1 2 4

Columns 5-10

- Using MPI_Type_vector, for each dimension a pair of blocklength&stride 0 0 0 0 0 0

Columns 11-13

- weights (double values) (only with cart_method==4) 1.00 0.50 0.25

Column 11

- 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

Columns 12-14

- dims_ml[d=0] = 1 2 2

15-17

- dims_ml[d=1] = 2 1 3

18-20

- dims_ml[d=2] = 4 1 2

Exercise: Explanations

Start a 8 or 12-node batch-job with your own input file:
Report your acceleration factors to the course group

- Input per measurement, e.g. on 8 nodes x 2 CPUs x 12 cores:

Example

Column 1

- cart_method:
 - 1=Dims_create+Cart_create,
 - 2=Cart_weighted_create (MPIX_WEIGHTS_EQUAL),
 - 3=dito(weights),
 - 4=dito manually,
 - 5=Cart_ml_create(dims_ml),
 - 0=end of input

These base values (per process) are multiplied with $\sqrt[3]{\#processes}$ and then with 1, 2, 4, 8, ... 512, e.g., with 192 processes: $2 \cdot \sqrt[3]{192} \cdot 512 = 5910$ (rounded to a multiple of the dim. of the process grid). See also later the slide explaining the output. Recommendation for several experiments: **Use the same initial mesh volume** (here 8), e.g., 1x2x4, 2x2x2, 4x2x1. Note that this application data mesh volume is **completely independent** of the number of hardware nodes, CPUs, cores.

2

Columns 2-4

- Data mesh sizes, integer start values (= ratio)

1 2 4

Columns 5-10

- Using MPI_Type_vector, for each dimension a pair of blocklength&stride

0 0 0 0 0 0

Columns 11-13

- weights (double values) (only with cart_method==4)

1.00 0.50 0.25

Column 11

- 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

Columns 12-14

- dims_ml[d=0] =

1 2 2

15-17

- dims_ml[d=1] =

2 1 3

18-20

- dims_ml[d=2] =

4 1 2

Exercise: Explanations

Start a 8 or 12-node batch-job with your own input file:
Report your acceleration factors to the course group

- Input per measurement, e.g. on 8 nodes x 2 CPUs x 12 cores:

Example

Column 1

- cart_method:
 - 1=Dims_create+Cart_create,
 - 2=Cart_weighted_create (MPIX_WEIGHTS_EQUAL),
 - 3=dito(weights),
 - 4=dito manually,
 - 5=Cart_ml_create(dims_ml),
 - 0=end of input

These base values (per process) are multiplied with $\sqrt[3]{\#processes}$ and then with 1, 2, 4, 8, ... 512, e.g., with 192 processes: $2 \cdot \sqrt[3]{192} \cdot 512 = 5910$ (rounded to a multiple of the dim. of the process grid). See also later the slide explaining the output. Recommendation for several experiments: **Use the same initial mesh volume** (here 8), e.g., 1x2x4, 2x2x2, 4x2x1. Note that this application data mesh volume is **completely independent** of the number of hardware nodes, CPUs, cores.

2

Columns 2-4

- Data mesh sizes, integer start values (= ratio)

0 0 = contiguous

1 2 4

Columns 5-10

- Using MPI_Type_vector, for each dimension a pair of blocklength&stride

0 0 0 0 0 0

Columns 11-13

- weights (double values) (only with cart_method==4)

1.00 0.50 0.25

Column 11

- 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

Columns 12-14

- dims_ml[d=0] =

1 2 2

15-17

- dims_ml[d=1] =

2 1 3

18-20

- dims_ml[d=2] =

4 1 2

Exercise: Explanations, continued

- 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	<u>256 1024</u> <u>4 32</u> 0 0	
▪ 0			

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create
+ MPI_Cart_create

- 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

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create
+ MPI_Cart_create

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

- 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 4. 2. 1.
- 5 1 2 4 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

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create
+ MPI_Cart_create

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

- 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
 - 5 1 2 4 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

Contiguous data in all three directions

4. 2. 1.

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create
+ MPI_Cart_create

▪ 1 1 2 4 0 0 0 0 0 0

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

Contiguous data in all three directions

MPIX_Cart_weighted_create
with MPIX_WEIGHTS_EQUAL

▪ 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

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create
+ MPI_Cart_create

▪ 1 1 2 4 0 0 0 0 0 0

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

Contiguous data in all three directions

MPIX_Cart_weighted_create
with MPIX_WEIGHTS_EQUAL

▪ 2 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with weights calculated as reciprocal value of the mesh sizes, i.e., 1./1 , 1./2 , 1./4

▪ 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

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create
+ MPI_Cart_create

▪ 1 1 2 4 0 0 0 0 0 0

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

Contiguous data in all three directions

MPIX_Cart_weighted_create
with MPIX_WEIGHTS_EQUAL

▪ 2 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with weights calculated as reciprocal value of the mesh sizes, i.e., 1./1 , 1./2, 1./4

▪ 3 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with given weights

▪ 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

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create
+ MPI_Cart_create

▪ 1 1 2 4 0 0 0 0 0 0

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

Contiguous data in all three directions

MPIX_Cart_weighted_create
with MPIX_WEIGHTS_EQUAL

▪ 2 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with weights calculated as reciprocal value of the mesh sizes, i.e., 1./1 , 1./2, 1./4

▪ 3 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with given weights

▪ 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

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create
+ MPI_Cart_create

▪ 1 1 2 4 0 0 0 0 0 0

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

Contiguous data in all three directions

MPIX_Cart_weighted_create
with MPIX_WEIGHTS_EQUAL

▪ 2 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with weights calculated as reciprocal value of the mesh sizes, i.e., 1./1 , 1./2 , 1./4

▪ 3 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with given weights

▪ 4 1 2 4 0 0 0 0 0 0 4. 2. 1.

MPIX_Cart_ml_create

▪ 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

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create
+ MPI_Cart_create

▪ 1 1 2 4 0 0 0 0 0 0

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

Contiguous data in all three directions

MPIX_Cart_weighted_create
with MPIX_WEIGHTS_EQUAL

▪ 2 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with weights calculated as reciprocal value of the mesh sizes, i.e., 1./1 , 1./2 , 1./4

▪ 3 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with given weights

▪ 4 1 2 4 0 0 0 0 0 0

4. 2. 1.

With 3 hardware levels (e.g. nodes, CPUs, cores)

MPIX_Cart_ml_create

▪ 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

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create + MPI_Cart_create

▪ 1 1 2 4 0 0 0 0 0 0

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

Contiguous data in all three directions

MPIX_Cart_weighted_create with MPIX_WEIGHTS_EQUAL

▪ 2 1 2 4 0 0 0 0 0 0

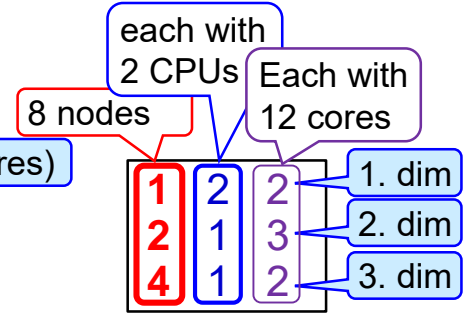
MPIX_Cart_weighted_create with weights calculated as reciprocal value of the mesh sizes, i.e., 1./1 , 1./2 , 1./4

▪ 3 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with given weights

▪ 4 1 2 4 0 0 0 0 0 0 4. 2. 1.

With 3 hardware levels (e.g. nodes, CPUs, cores)



MPIX_Cart_ml_create

▪ 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

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create + MPI_Cart_create

▪ 1 1 2 4 0 0 0 0 0 0

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

Contiguous data in all three directions

MPIX_Cart_weighted_create with MPIX_WEIGHTS_EQUAL

▪ 2 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with weights calculated as reciprocal value of the mesh sizes, i.e., 1./1 , 1./2 , 1./4

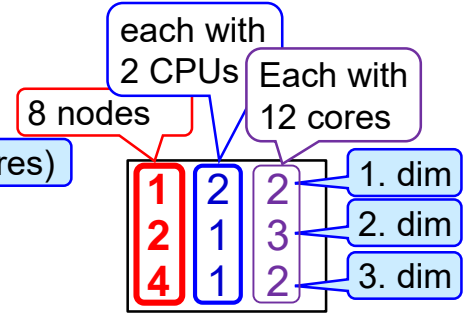
▪ 3 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with given weights

▪ 4 1 2 4 0 0 0 0 0 0

4. 2. 1.

With 3 hardware levels (e.g. nodes, CPUs, cores)



MPIX_Cart_ml_create

▪ 5 1 2 4 0 0 0 0 0 0 3 1 2 2 2 1 3 4 1 2

Whereas last experiment is with cubic data mesh and same start mesh volume = 8

▪ 3 2 2 2 256 1024 4 32 0 0

▪ 0

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create + MPI_Cart_create

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

Contiguous data in all three directions

MPIX_Cart_weighted_create with MPIX_WEIGHTS_EQUAL

MPIX_Cart_weighted_create with weights calculated as reciprocal value of the mesh sizes, i.e., 1./1 , 1./2, 1./4

MPIX_Cart_weighted_create with given weights

MPIX_Cart_ml_create

With 3 hardware levels (e.g. nodes, CPUs, cores)

each with 2 CPUs
Each with 12 cores
8 nodes

1. dim
2. dim
3. dim

Whereas last experiment is with cubic data mesh and same start mesh volume = 8

examples for strided data in direction 0 & 1

```

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
  
```

Exercise: Explanations, continued

- Input can be concatenated to one line per experiment:

MPI_Dims_create + MPI_Cart_create

▪ 1 1 2 4 0 0 0 0 0 0

All these experiments use a data mesh ratio of 1 x 2 x 4 and start mesh volume = 8

Contiguous data in all three directions

MPIX_Cart_weighted_create with MPIX_WEIGHTS_EQUAL

▪ 2 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with weights calculated as reciprocal value of the mesh sizes, i.e., 1./1 , 1./2, 1./4

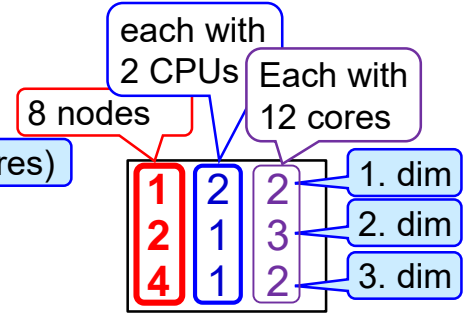
▪ 3 1 2 4 0 0 0 0 0 0

MPIX_Cart_weighted_create with given weights

▪ 4 1 2 4 0 0 0 0 0 0

4. 2. 1.

With 3 hardware levels (e.g. nodes, CPUs, cores)



MPIX_Cart_ml_create

▪ 5 1 2 4 0 0 0 0 0 0 3 1 2 2 2 1 3 4 1 2

Whereas last experiment is with cubic data mesh and same start mesh volume = 8

▪ 3 2 2 2 256 1024 4 32 0 0

examples for strided data in direction 0 & 1

▪ 0

0: marks end of input