MPI-Topologien

Vortrag von Irina Lindt
im Seminar Effizientes Programmieren

Themen

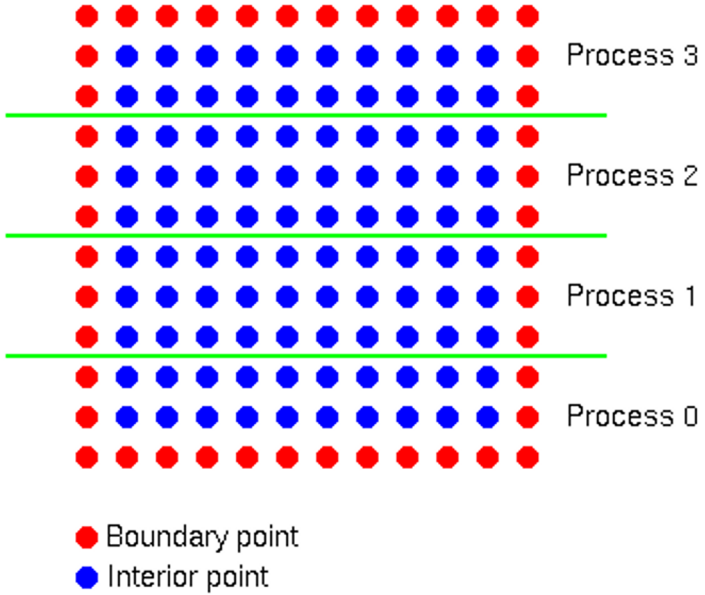
1. Einführung und Motivation
2. Communicator
3. Topologien
4. Zusammenfassung

1. Einführung und Motivation

- warum brauchen wir Communicator und Topologien?
- in welchem Schritt der Berechnung mit MPI kommen sie zum Einsatz?

Motivation

Beispiel Jacobi-Verfahren aus Hochleistungsrechnen [4]:



- Prozesse kommunizieren nur mit dem Nachbar von oben und von unten

Quelle: [10]

Motivation

Ausgangslage: Prozesse sind in MPI einfach durchnummeriert

Kommuniziert Prozess 1 viel mit Prozess 2?

Oder vielleicht mit Prozess 10?

MPI weiß nichts darüber.

Folge: keine Optimierung der Kommunikation möglich

Motivation

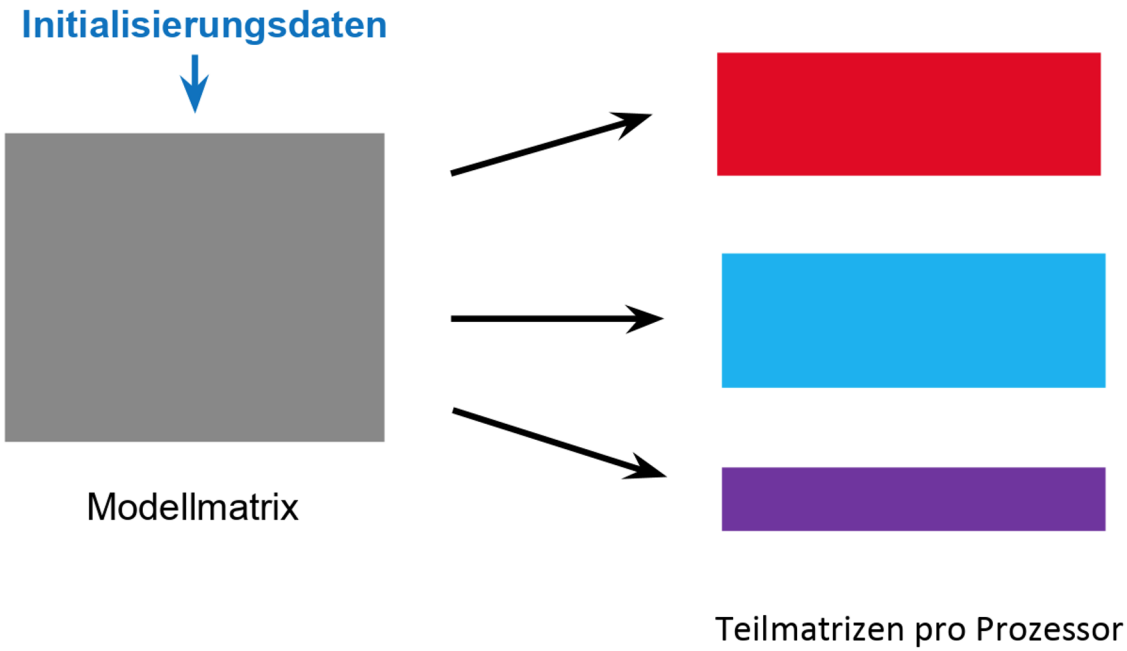
Wunsch:

- MPI soll über Kommunikationspartner Bescheid wissen
- MPI soll daraus bessere Performance ableiten
- MPI soll daraus bessere Modularität ableiten

MPI Phasen

- es gibt drei Phasen von Berechnungen auf großen Matrizen
- in welcher Phase sind Topologien relevant?

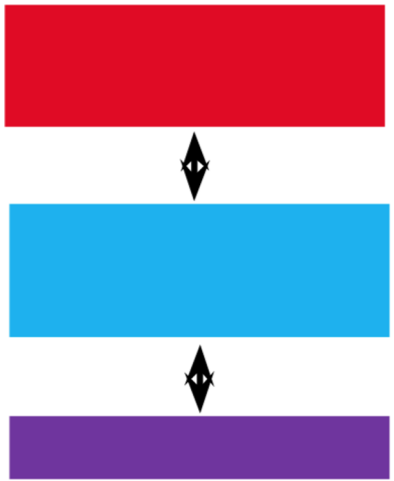
Initialisierung



Quelle: [3]

Berechnung auf Teilmatrizen

Modellmatrix

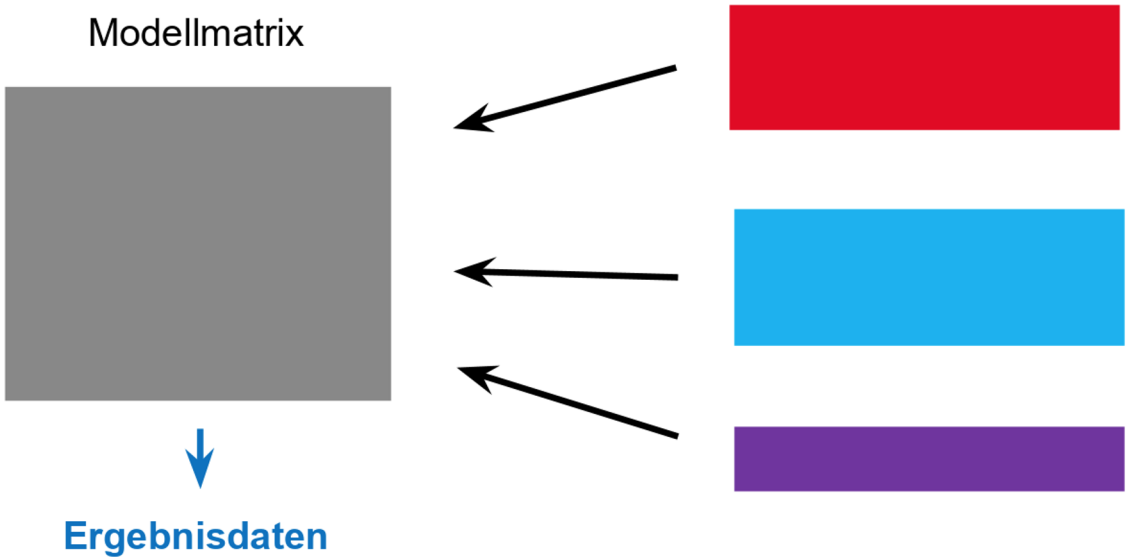


Teilmatrizen pro Prozessor

Hier kommen Topologien zum Einsatz!

Quelle: [3]

Finalisierung



Quelle: [3]

2. Communicator

Communicator hat:

- eine Gruppe von Prozessen, nummeriert von 0 bis n
- einen Kontext für diese Prozesse

Inter- und Intra-Communicator

- Intra-Communicator kommunizieren mit Prozessen in derselben Gruppe
- für die Kommunikation zwischen mehreren Gruppen gibt es Inter-Communicator
- Topologien können nur auf Intra-Communicator definiert werden
- **Communicator in diesem Vortrag steht für Intra-Communicator**

MPI_Comm_world

- der initiale Communicator
- wird von MPI beim Start erstellt
- enthält alle Prozesse von 0 bis size - 1

MPI_Comm_rank

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

(in) comm: communicator (handle)

(out) rank: rank of the calling process in the group of comm (integer)

MPI_Comm_size

int MPI_Comm_size(MPI_Comm comm, int *size)

(in) comm: communicator (handle)

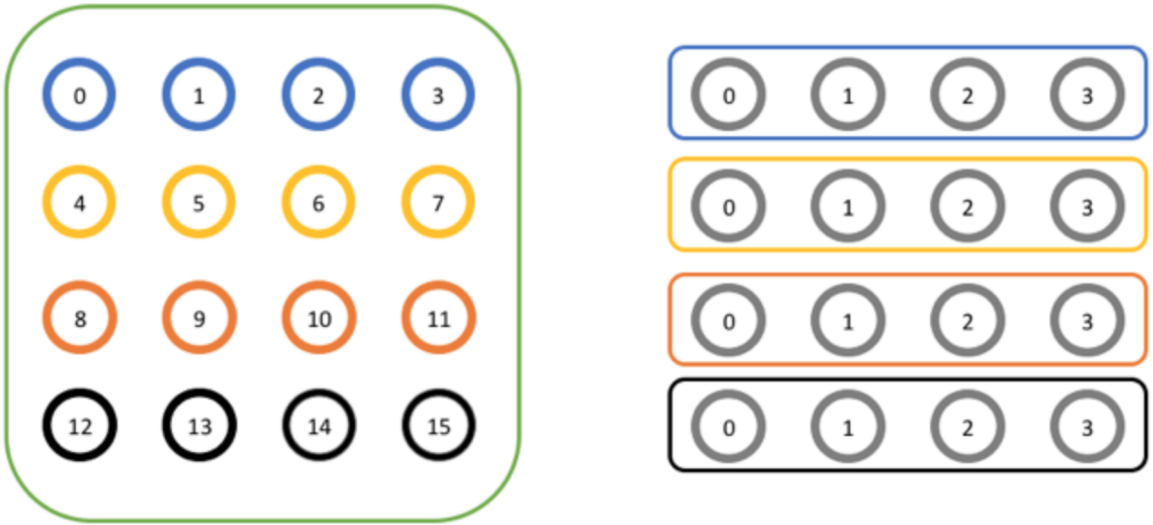
(out) size: number of processes in the group of comm (integer)

Codebeispiel

```
int world_rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

MPI_Comm_split

Teilt einen Communicator in mehrere kleinere auf.



Quelle: [8]

MPI_Comm_split

Teilt einen Communicator in mehrere kleinere auf.

int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm * newcomm)

(in) comm: communicator (handle)

(in) color: control of subset assignment (nonnegative integer). Processes with the same color are in the same new communicator

(in) key: control of rank assignment (integer)

(out) newcomm: new communicator (handle)

Codebeispiel

```c
#include<mpi.h>
#include<stdio.h>
 /* An example for a communicator split into smaller ones, 4 processes each */
int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);

    int world_rank, world_size;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int color = world_rank / 4; // Determine color based on row

    // Split the communicator based on the color and use the
    // original rank for ordering
    MPI_Comm row_comm;
    MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &row_comm);

    int row_rank, row_size;
    MPI_Comm_rank(row_comm, &row_rank);
    MPI_Comm_size(row_comm, &row_size);

    printf("WORLD RANK/SIZE: %d/%d \t ROW RANK/SIZE: %d/%d\n",
        world_rank, world_size, row_rank, row_size);

    MPI_Comm_free(&row_comm);
    MPI_Finalize();
    return 0;
}
```

Ausgabe

```
Compiling
Compilation is OK
Execution ...
WORLD RANK/SIZE: 0/12        ROW RANK/SIZE: 0/4
WORLD RANK/SIZE: 1/12        ROW RANK/SIZE: 1/4
WORLD RANK/SIZE: 2/12        ROW RANK/SIZE: 2/4
WORLD RANK/SIZE: 3/12        ROW RANK/SIZE: 3/4
WORLD RANK/SIZE: 4/12        ROW RANK/SIZE: 0/4
WORLD RANK/SIZE: 5/12        ROW RANK/SIZE: 1/4
WORLD RANK/SIZE: 6/12        ROW RANK/SIZE: 2/4
WORLD RANK/SIZE: 7/12        ROW RANK/SIZE: 3/4
WORLD RANK/SIZE: 8/12        ROW RANK/SIZE: 0/4
WORLD RANK/SIZE: 9/12        ROW RANK/SIZE: 1/4
WORLD RANK/SIZE: 10/12       ROW RANK/SIZE: 2/4
WORLD RANK/SIZE: 11/12       ROW RANK/SIZE: 3/4
Done.
```

3. Topologien

- einem Communicator kann eine Topologie zugewiesen werden
- diese enthält seine Kommunikationspartner

Anwendungsbeispiel: 1D, 2D, 3D Torus

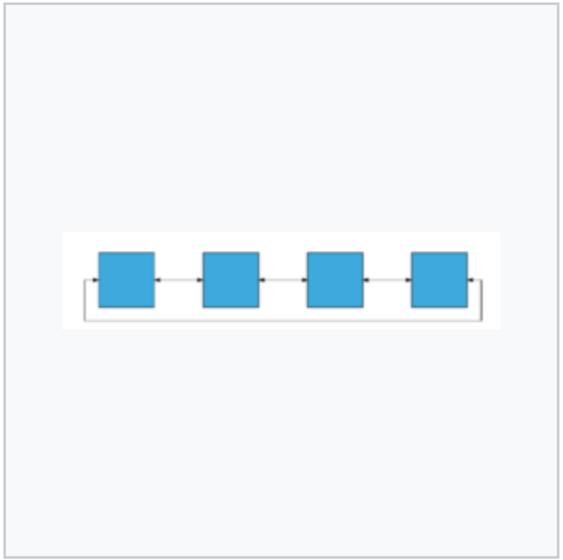


illustration of 1D Torus

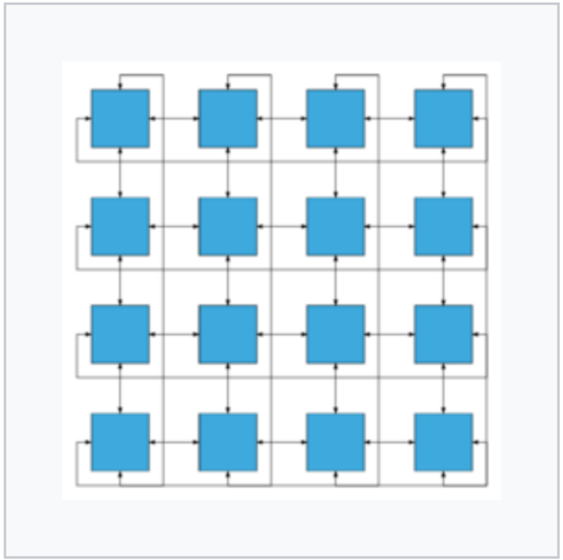


illustration of 2D Torus

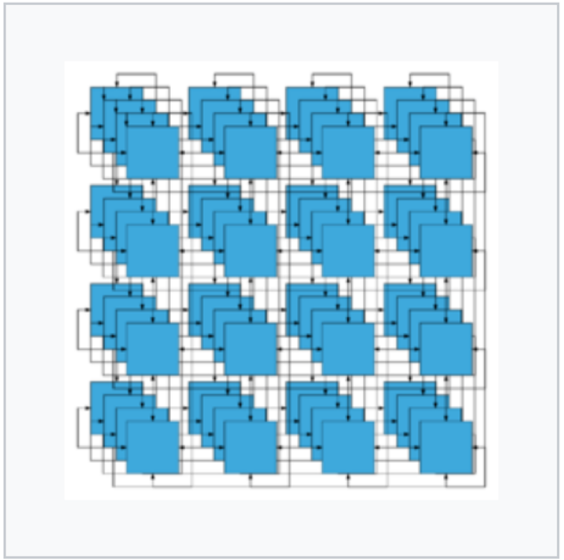
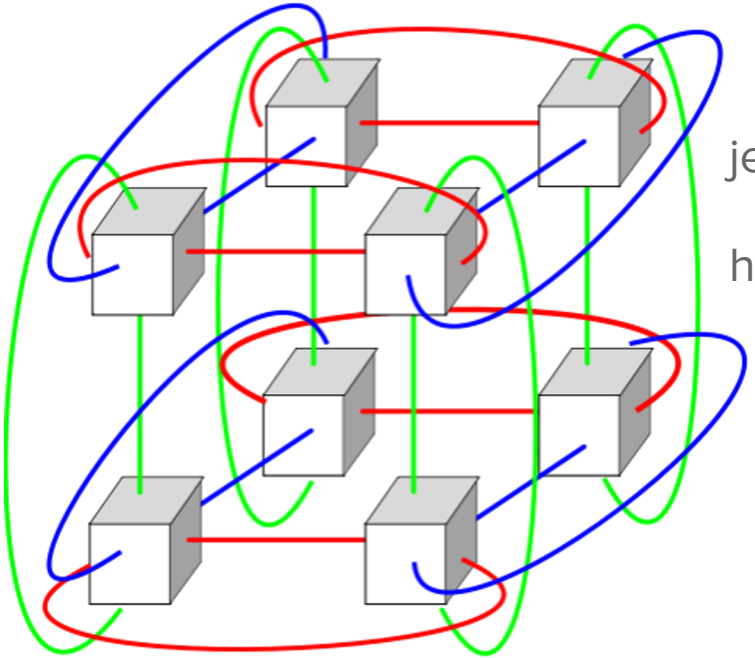


illustration of 3D Torus

Quelle: https://en.wikipedia.org/wiki/Torus_interconnect

Anwendungsbeispiel: 3D Torus



jeder Prozess

hat jeweils 6 Nachbarn

Quelle: https://en.wikipedia.org/wiki/Torus_interconnect

Beachte: Topologien sind rein virtuell

- dienen zum besseren Verständnis für die Entwickler
- bilden meist nicht die reale Hardware ab
- Performanzgewinn nicht immer möglich
- aber: [7] zeigt ein Beispiel für bessere Performance (siehe Bild)

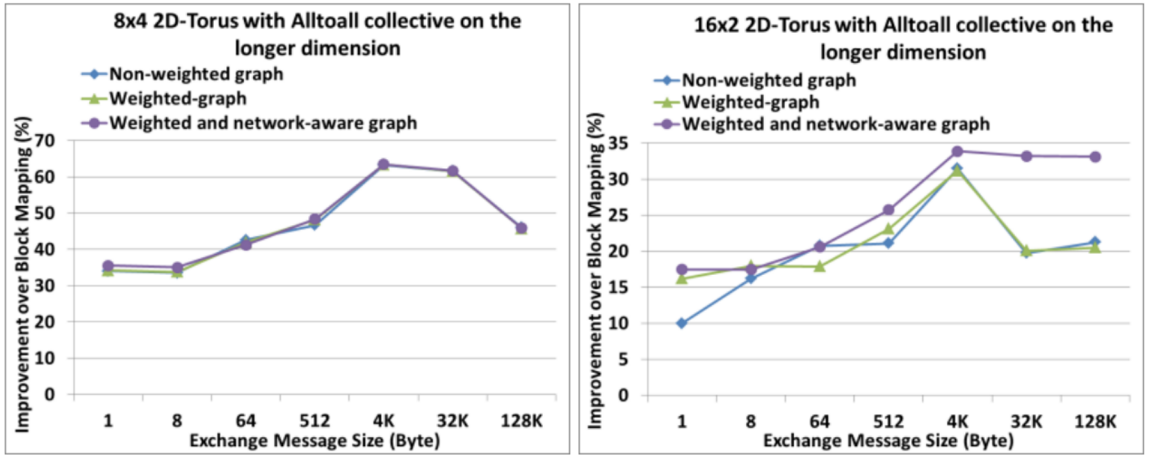


Fig. 3. Runtime improvement of topology-aware mapping over block mapping for 2D-torus in the dimensional collective (Alltoal) micro-benchmark

Varianten von Topologien in MPI

1. Kartesisch
- kartesische Koordinaten
- in rechteckigen Rastern angeordnet
- Prozesse kommunizieren mit den Nachbarn

2. Graphen
- alle sonstigen Topologien
- dargestellt als allgemeine Graphen oder verteilte Graphen

Kartesische Topologie: MPI_Cart_create

int MPI_Cart_create(MPI_Comm comm_old, int ndims, const int dims[], const int periods[], int reorder, MPI_Comm * comm_cart);

(in) comm_old: input communicator (handle)

(in) ndims: number of dimensions of cartesian grid (integer)

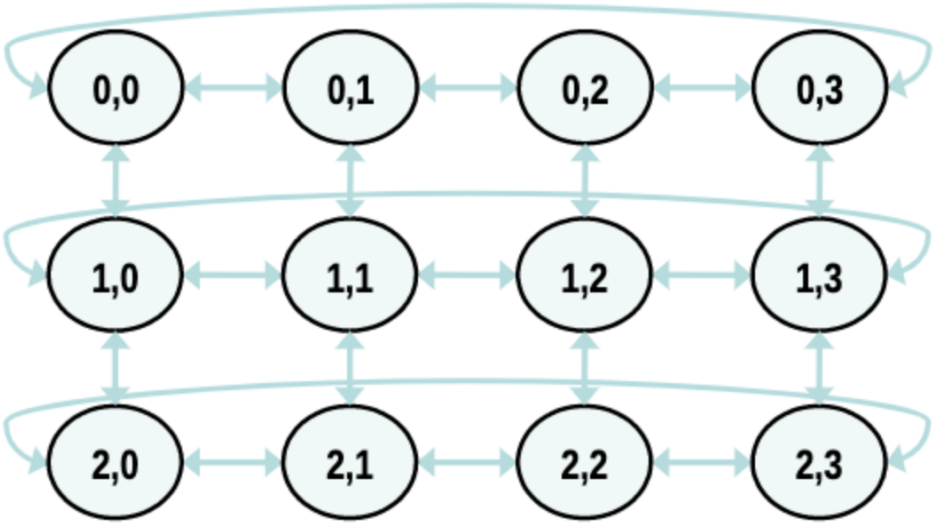
(in) dims: integer array of size ndims specifying the number of processes in each dimension

(in) periods: logical array of size ndims specifying whether the grid is periodic (true) or not (false) in each dimension

(in) reorder: ranking may be reordered (true) or not (false) (logical)

(out) comm_cart: communicator with new cartesian topology (handle)

MPI_Cart_create: Beispiel



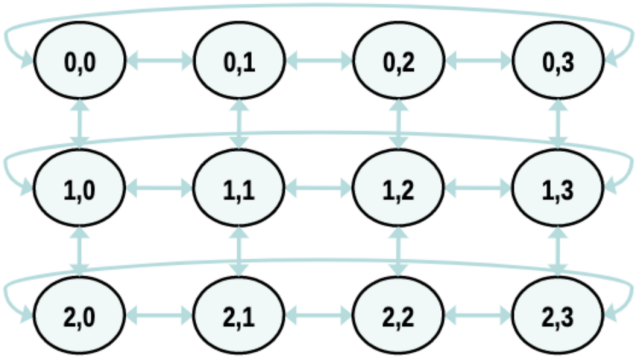
4x3 Grid, nur Zeilen periodisch. Quelle: [5]

MPI_Cart_create: Beispiel

```c
#include<mpi.h>
#include<stdio.h>
 /* A two-dimensional torus of 12 processes in a 4x3 grid */
int main(int argc, char *argv[])
{
    int rank, size;
    MPI_Comm comm;
    int dim[2], period[2], reorder;
    int coord[2], id;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    dim[0]=4; dim[1]=3;
    period[0]=1; period[1]=0;
    reorder=0;
    MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, reorder, &comm);
    if (rank == 5)
    {
        MPI_Cart_coords(comm, rank, 2, coord);
        printf("Rank %d coordinates are %d %d\n", rank, coord[1], coord[0]);fflush(stdout);
    }
    if(rank==0)
    {
        coord[0]=3; coord[1]=1;
        MPI_Cart_rank(comm, coord, &id);
        printf("The processor at position (%d, %d) has rank %d\n", coord[1], coord[0], id);fflush(stdout);
    }
    MPI_Finalize();
    return 0;
}
```



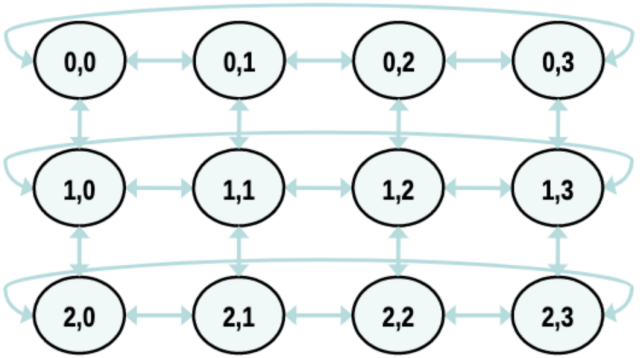
Quelle: [5]

MPI_Cart_create: Beispiel

```c
#include<mpi.h>
#include<stdio.h>
 /* A two-dimensional torus of 12 processes in a 4x3 grid */
int main(int argc, char *argv[])
{
    int rank, size;
    MPI_Comm comm;
    int dim[2], period[2], reorder;
    int coord[2], id;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    dim[0]=4; dim[1]=3;
    period[0]=1; period[1]=0;
    reorder=0;
    MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, reorder, &comm);
    if (rank == 5)
    {
        MPI_Cart_coords(comm, rank, 2, coord);
        printf("Rank %d coordinates are %d %d\n", rank, coord[1], coord[0]);fflush(stdout)
    }
    if(rank==0)
    {
        coord[0]=3; coord[1]=1;
        MPI_Cart_rank(comm, coord, &id);
        printf("The processor at position (%d, %d) has rank %d\n", coord[1], coord[0], id);fflush(stdout);
    }
    MPI_Finalize();
    return 0;
}
```

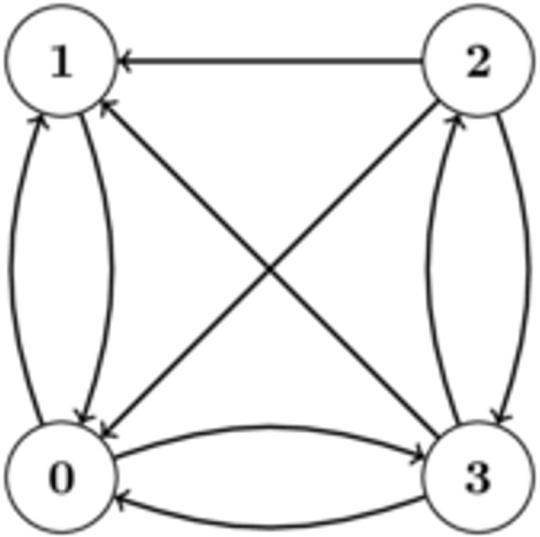


Quelle: [5]

```
Compiling
Compilation is OK
Execution ...
The processor at position (1, 3) has rank 10
Rank 5 coordinates are 2 1
Done.
```

Bildbeispiel für Graphtopologie

- manche Prozesse kommunizieren auch diagonal
- allgemeine Graphtopologie nötig



Quelle: [5]

Graphtopologie: MPI_Graph_create

int MPI_Graph_create(MPI_Comm comm_old, int nnodes, int *index, int *edges, int reorder, MPI_Comm *comm_graph);

(in) comm_old: input communicator without topology (handle)

(in) nnodes: number of nodes in graph (integer)

(in) index: array of integers describing node degrees

(in) edges: array of integers describing graph edges

(in) reorder: ranking may be reordered (true) or not (false) (logical)

(out) comm_graph: communicator with graph topology added (handle)

MPI_Dist_Graph_create

int MPI_Dist_graph_create (MPI_Comm comm_old, int n, const int sources[], const int degrees[], const int destinations[], const int weights[], MPI_Info info, int reorder, MPI_Comm *comm_dist_graph)

(in) comm_old: input communicator (handle)
(in) n: number of source nodes for which this process specifies edges (non-negative integer)
(in) sources: array containing the n source nodes for which this process specifies edges (array of non-negative integers)
(n) degrees: array specifying the number of destinations for each source node in the source node array (array of non-negative integers)

MPI_Dist_graph_create (cont.)

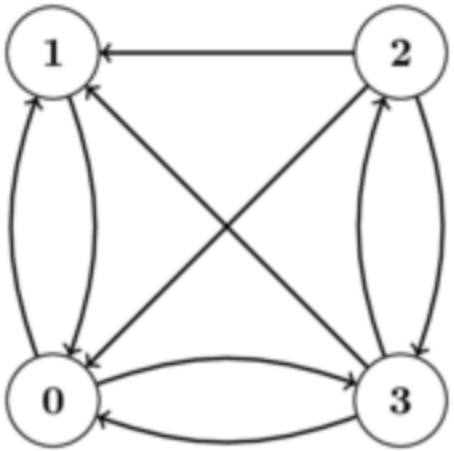
int MPI_Dist_graph_create (MPI_Comm comm_old, int n, const int sources[], const int degrees[], const int destinations[], const int weights[], MPI_Info info, int reorder, MPI_Comm *comm_dist_graph)

(in) destinations : destination nodes for the source nodes in the source node array (array of non-negative integers)
(in) weights : weights for source to destination edges (array of non-negative integers or MPI_UNWEIGHTED)
(in) info : hints on optimization and interpretation of weights (handle)
(in) reorder : the process may be reordered (1) or not (0) (logical)
(out) comm_dist_graph : communicator with distributed graph topology added (handle)

MPI_Dist_graph_create: Beispiel

```
// set dest and degree.
if (rank == 0)
{
    dest[0] = 1;
    dest[1] = 3;
    degree = 2;
}
else if(rank == 1)
{
    dest[0] = 0;
    degree = 1;
}
else if(rank == 2)
{
    dest[0] = 3;
    dest[1] = 0;
    dest[2] = 1;
    degree = 3;
}
else if(rank == 3)
{
    dest[0] = 0;
    dest[1] = 2;
    dest[2] = 1;
    degree = 3;
}

// create graph.
MPI_Comm graph;
MPI_Dist_graph_create(MPI_COMM_WORLD, 1, &source, &degree, dest, weight, MPI_INFO_NULL, 1, &graph);
```



Source: [5]

4. Zusammenfassung

–Communicator erlauben die Aufteilung von Prozessen in Gruppen

–Vorteile:

- bessere Lesbarkeit
- Modularität
- Insulation

–MPI Topologien erlauben die Abbildung von Kommunikationsstrukturen auf MPI-Prozesse

–Vorteile:

- bessere Lesbarkeit
- Performanz kann optimiert werden, z.B. im Netzwerk

Literaturliste

1. https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf (Zugriff am 13.01.21)
MPI-Referenz zur Version 3.1. Topologien: Kapitel 6
2. https://www.ia.pw.edu.pl/~ens/epnm/mpi_course.pdf (Zugriff am 13.01.21)
MPI-Kurs des Edinburgh Parallel Computing Center. Topologien: Kapitel 7
3. https://wr.informatik.uni-hamburg.de/_media/teaching/sommersemester_2019/ppg-19-parallel-prog-intro.pdf (Zugriff am 13.01.21)
Folien zum Praktikum Paralleles Programmieren für Geowissenschaftler am DKRZ
4. https://wr.informatik.uni-hamburg.de/teaching/wintersemester_2020_2021/hochleistungsrechnen (Zugriff am 13.01.21)
Folien zur Vorlesung Hochleistungsrechnen am DKRZ

Literaturliste

5. https://www.codingame.com/playgrounds/47058/have-fun-with-mpi-in-c/mpi-process-topologies (Zugriff am 13.01.21)
Interaktive Beispiele für MPI-Topologien.
6. https://www.youtube.com/watch?v=vtT3S-47Zig (Zugriff am 13.01.21)
Videotutorial von SHARCNET, Topologien ab Minute 20
7. https://www.queensu.ca/academia/afsahi/pprl/papers/EuroMPI-2011.pdf (Zugriff am 13.01.21)
Multi-core and Network Aware MPI Topology Functions, Rashti et al. Paper, das auf Performanz von MPI_Cart_create
und MPI_Graph_create eingeht.

Literaturliste: Tutorials

8. https://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/
MPI Tutorial zum Thema Communicator. (Zugriff am 13.01.21)
9. https://pages.tacc.utexas.edu/~eijkhout/pcse/html/mpi-topo.html
MPI Tutorial der University of Texas. (Zugriff am 13.01.21)
10.https://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiexmpl/src/jacobi/C/main.html
Tutorial der Argonne National Laboratory zum Jacobi-Verfahren mit MPI. (Zugriff am 13.01.21)