MPI Topologien

Effiziente Programmierung

Alena Pils

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2019-12-03

Gliederung (Agenda)

Motivation

- **Beispiel: Globalmodell ICON**
- Größenordung des Zeitschritts: wenige Minuten
- Maschenweite: 13 km
- Vertikale Schichten: 90

  $\rightarrow$ 265 Millionen Gitterpunkte

  $\Rightarrow$ Hochleistungsrechner mit Aufteilung des Gitters nötig

Abbildung: ICON-Modellgitter [DWD]

Initialisierung

1) Initialisierung: Aufteilung der Rechengebiete und Initialisierung

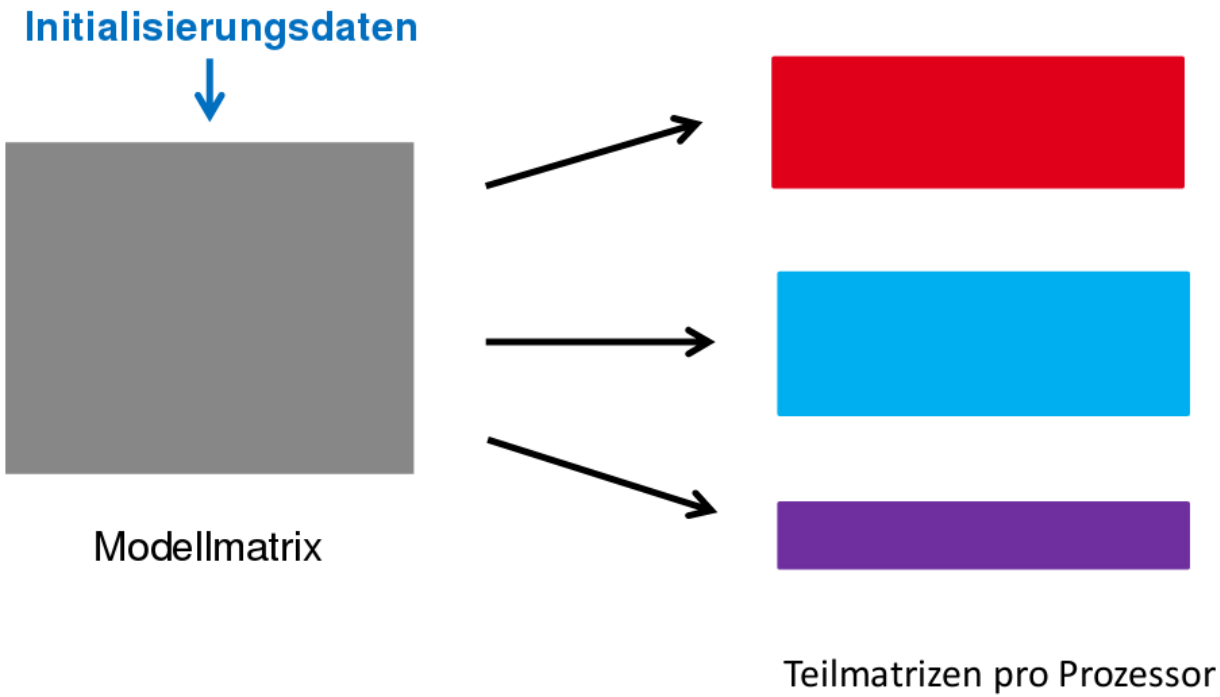

Abbildung: Aufteilung der Rechengebiete und Initialisierung [LLB19]

Berechnung des Modells auf Teilmatrizen

2) Berechnung des Modells auf Teilmatrizen: Austausch zwischen Teilmatrizen

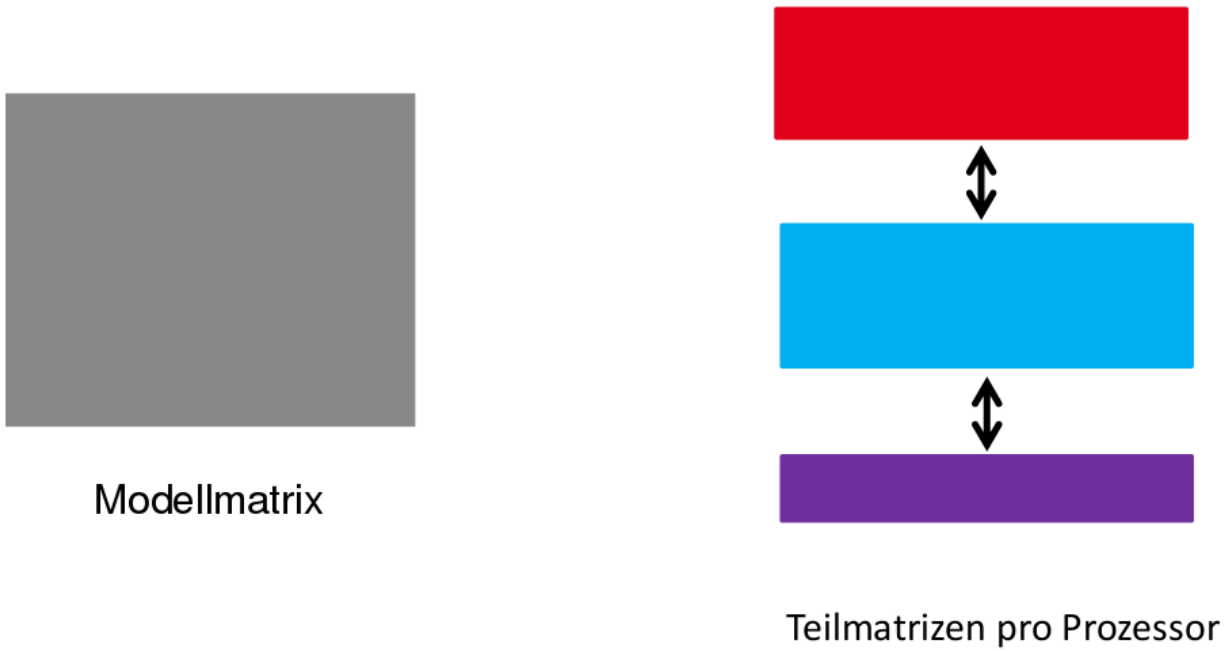

Abbildung: Austausch zwischen Teilmatrizen [LLB19]

Finalisierung

3) Finalisierung: Zusammenfügen der Rechenergebnisse der Teilmatrizen

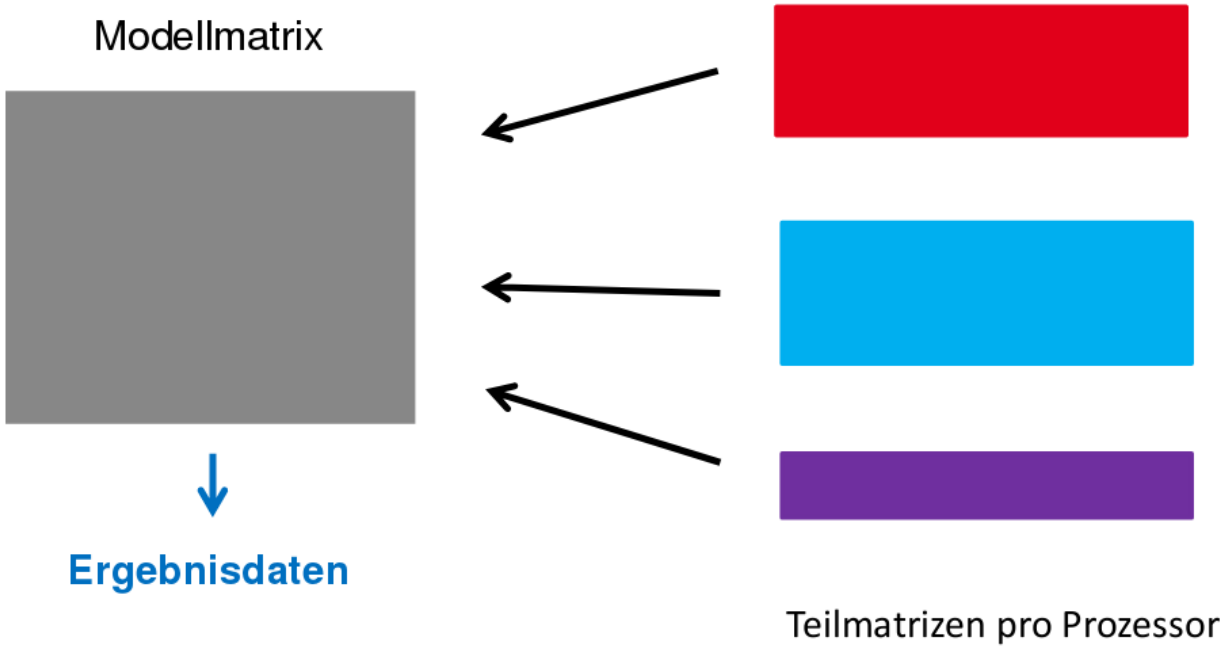

Abbildung: Zusammenfügen der Rechenergebnisse der Teilmatrizen

[LLB19]

Was ist MPI?

- MPI = Message-Passing Interface
- Ziel: keine Kompromisse zwischen Effizienz, Portabilität und Funktionalität
- „MPI bedient das Message-Passing-Modell.“ [GLS07]
- „MPI ist eine Bibliothek und keine Sprache.“ [GLS07]
- „MPI ist eine Spezifikation, keine spezielle Implementierung.“ [GLS07]

Start der MPI-Umgebung

```
1  program start
2     use mpi
3
4     implicit none
5     integer :: ierr, irank, wsize
6
7     call MPI_INIT(ierr)
8     call MPI_COMM_RANK(MPI_COMM_WORLD, irank, ierr)
9     call MPI_COMM_SIZE(MPI_COMM_WORLD, wsize, ierr)
10
11    write(*,'("Ich bin Rang",i2," von",i2," Prozessen.")') irank,
       ↪ wsize
12
13    call MPI_FINALIZE(ierr)
14
15 end program start
```

Listing 1: Die MPI-Umgebung

Ausführung des Programms

```
pils@west1:~/EP/start$ mpiexec -n 4 ./start.x
Ich bin Rang 0 von 4 Prozessen.
Ich bin Rang 1 von 4 Prozessen.
Ich bin Rang 2 von 4 Prozessen.
Ich bin Rang 3 von 4 Prozessen.
```

Abbildung: Ausgabe des Programmcodes

Konzept des Nachrichtenaustauschs

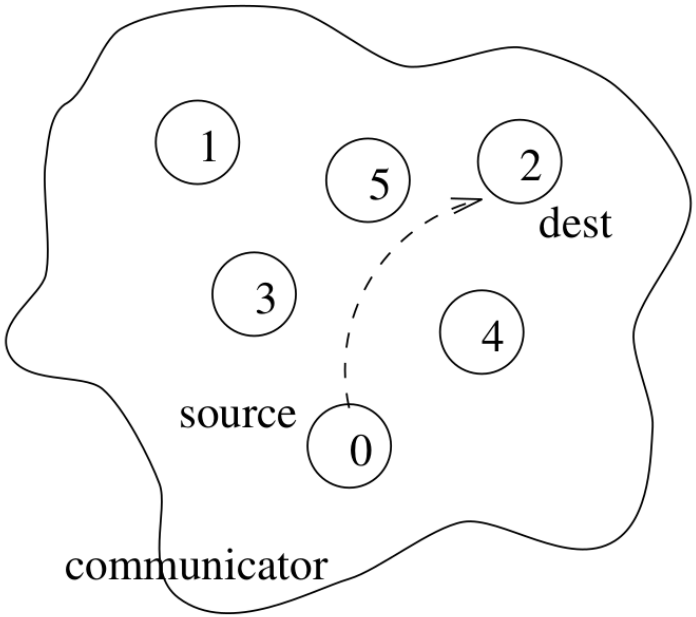

Abbildung: Point-to-Point Kommunikation. Ein Prozess sendet eine Nachricht an einen konkreten anderen Prozess. [BMM+]

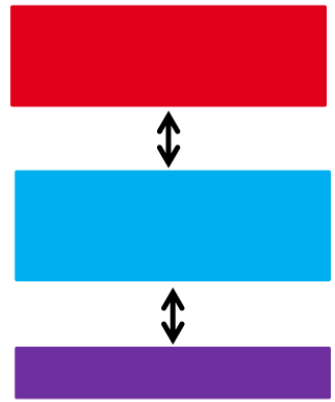

Abbildung: Austausch zwischen Teilmatrizen [LLB19]

Send/Receive Syntax

MPI_SEND(message, count, datatype, dest, tag, comm, ierror)
MPI_RECV(message, count, datatype, source, tag, comm, status, ierror)

message	Nachricht
count	Anzahl der Elemente
datatype	Datentyp des gesendeten Elements
dest	Rang des Zielprozesses
source	Rang des Sendeprozesses
tag	Nachrichtenkennung
comm	Kommunikator
status	Empfangsstatus der Nachricht
ierror	Fehlerstatus

Send/Receive Beispiel

```
1  ...
2  integer           :: master = 0, i, status(MPI_STATUS_SIZE)
3  character(len=5) :: nachricht
4  ...
5  nachricht = "Moin"
6  if (irank == master) then
7     nachricht = "Hallo"
8     call MPI_SEND(nachricht, 5, MPI_CHARACTER, 2, 2019,
       ↪ MPI_COMM_WORLD, ierr)
9  else if (irank == 2) then
10    call MPI_RECV(nachricht, 5, MPI_CHARACTER, master, 2019,
       ↪ MPI_COMM_WORLD, status, ierr)
11 end if
12
13 write(*,'(i1,a6)') irank, nachricht
14 ...
```

Listing 2: Senden und Empfangen einer Nachricht

Ausführung des Programms

```
pils@west1:~/EP/start$ mpiexec -n 4 ./start2.x
1 Moin
3 Moin
0 Hallo
2 Hallo
```

Abbildung: Ausgabe des Programmcodes

Kollektive Kommunikation

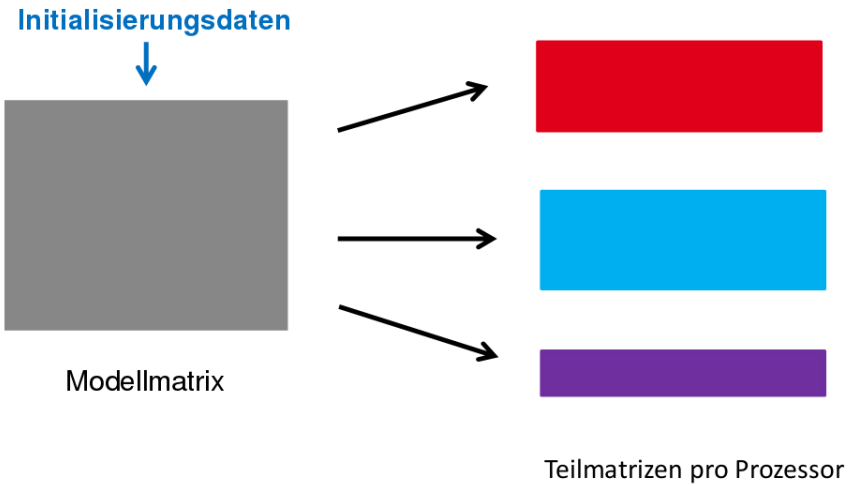

Abbildung: Initialisierung: **Broadcast**, Aufteilung der Matrix auf Teilmatrizen: **Scatter** [LLB19]

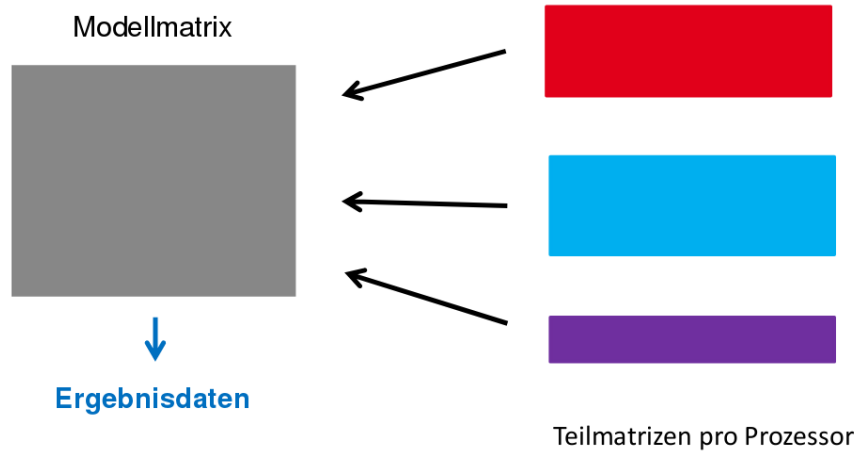

Abbildung:
Finalisierung/Zusammenfügen der Teilmatrizen: **Gather** [LLB19]

Kollektive Kommunikation

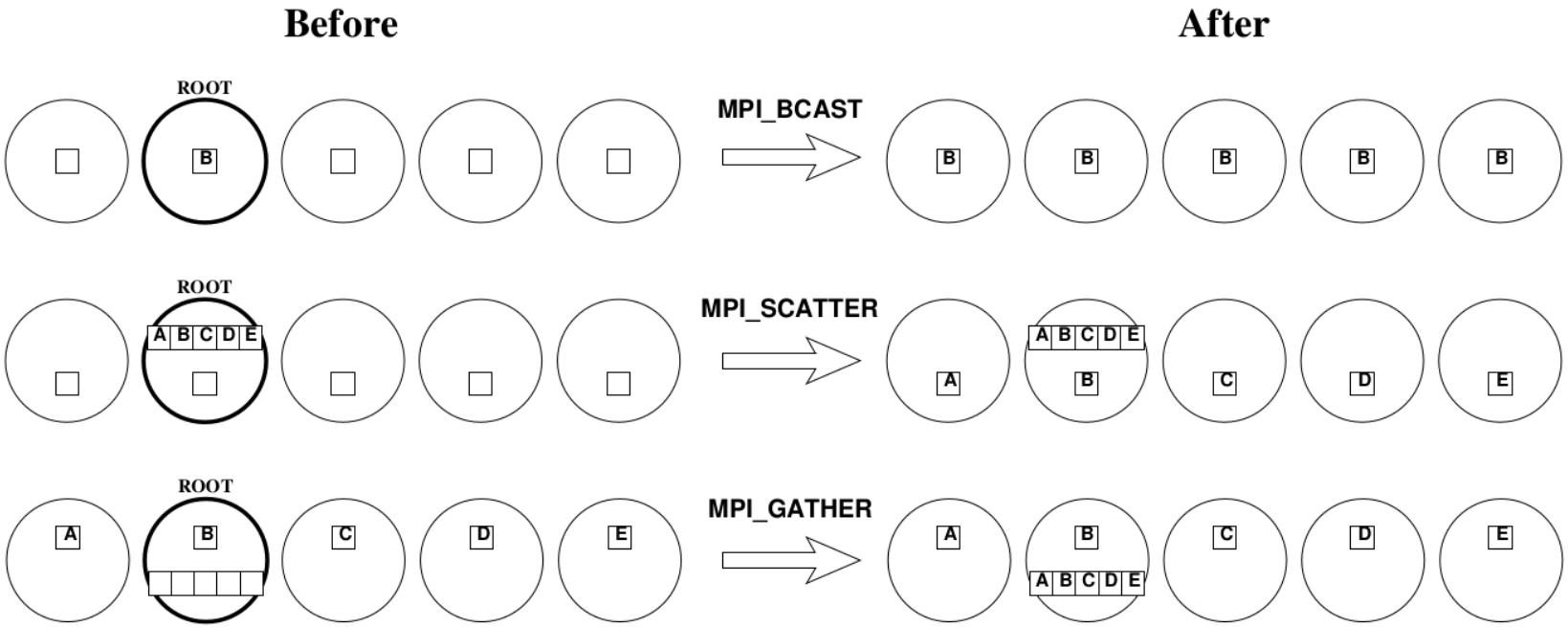

Abbildung: Schematische Illustration von Broadcast, Scatter und Gather [BMM+]

Definition Topologie

- Es muss zwischen zwei verschiedenen Konzepten der Topologie unterschieden werden:
  - **Topologie der Hardware:** beschreibt, wie die Prozesse eines Parallelrechners mit anderen Prozessen verbunden sind
  - **Virtuelle Topologie:** Kommunikationsmuster

- MPI unterscheidet zwischen drei Topologietypen: Kartesisch, Graph und verteilter Graph

Konstruktor Syntax

MPI_CART_CREATE(comm_old, ndims, dims, periods, reorder, comm_cart, ierr)

comm_old	in	Kommunikator
ndims	in	Anzahl an Dimensionen im kartesischen Gitter
dims	in	Array der Größe ndims, der die Anzahl an Prozessen in jeder Dimension spezifiziert
periods	in	Array der Größe ndims, der beschreibt ob das Gitter periodisch ist oder nicht
reorder	in	Rangfolge kann sich ändern (true) oder nicht (false)
comm_cart	out	Kommunikator mit kartesischer Topologie
ierr	out	Fehlerstatus

Kartesische Topologie

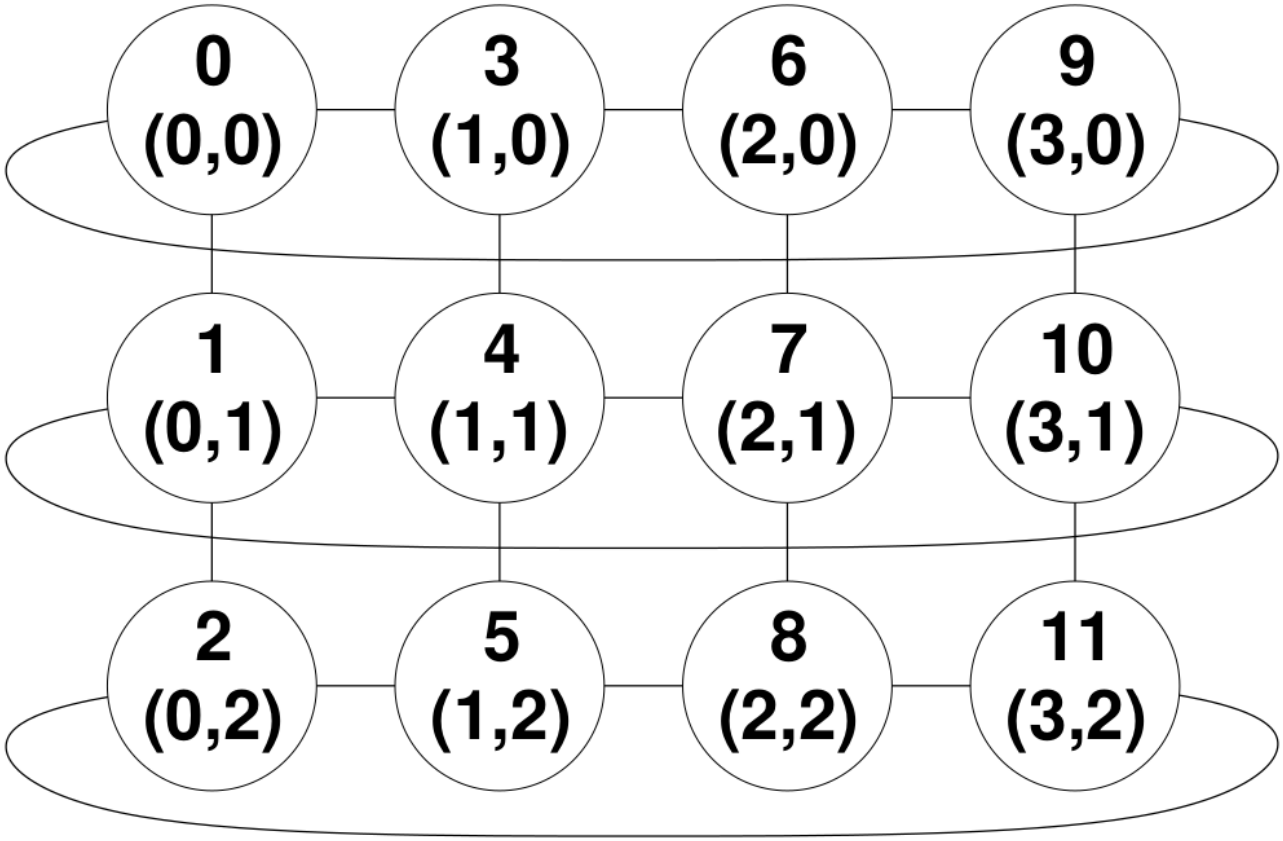

Abbildung: Virtuelle Topologie bestehend aus zwölf Prozessen [BMM+]

MPI_CART_COORDS Syntax

MPI_CART_COORDS(comm, rank, maxdims, coords)

comm	in	Kommunikator mit kartesischer Struktur
rank	in	Rang des Prozesses
maxdims	in	Länge des Vektors coords
coords	out	Integer Array, der die kartesischen Koordinaten des jeweiligen Prozesses enthält

MPI_CART_SHIFT Syntax

MPI_CART_SHIFT(comm, direction, disp, rank_source, rank_dest, ierr)

comm	in	Kommunikator mit kartesischer Struktur
direction	in	Koordinatendimension der Verschiebung
disp	in	Verschiebung (>0 nach oben, <0: nach unten)
rank_source	out	Rang des Quellprozesses
rank_dest	out	Rang des Zielprozesses
ierr	out	Fehlerstatus

MPI_CART_SHIFT

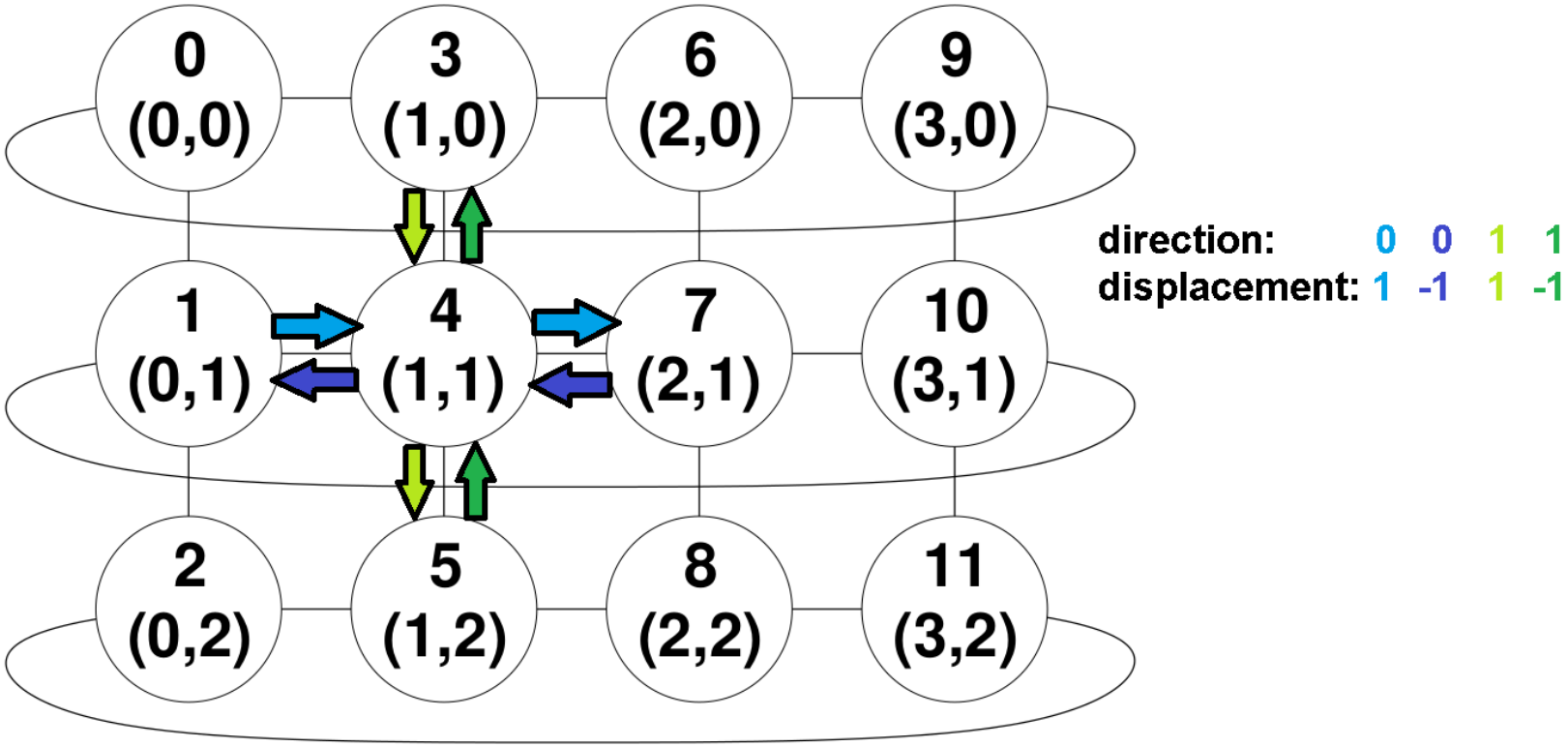

Abbildung: Virtuelle Topologie bestehend aus zwölf Prozessen. Verschiebungen nach MPI_CART_SHIFT [BMM+]

Beispielproblem

- vier Prozesse mit je einer Teilmatrix der Größe 5x5
- Austausch der jeweils letzten Spalte jeder Teilmatrix mit der ersten Spalte der rechts danebenliegenden Teilmatrix
- zyklische Randbedingung

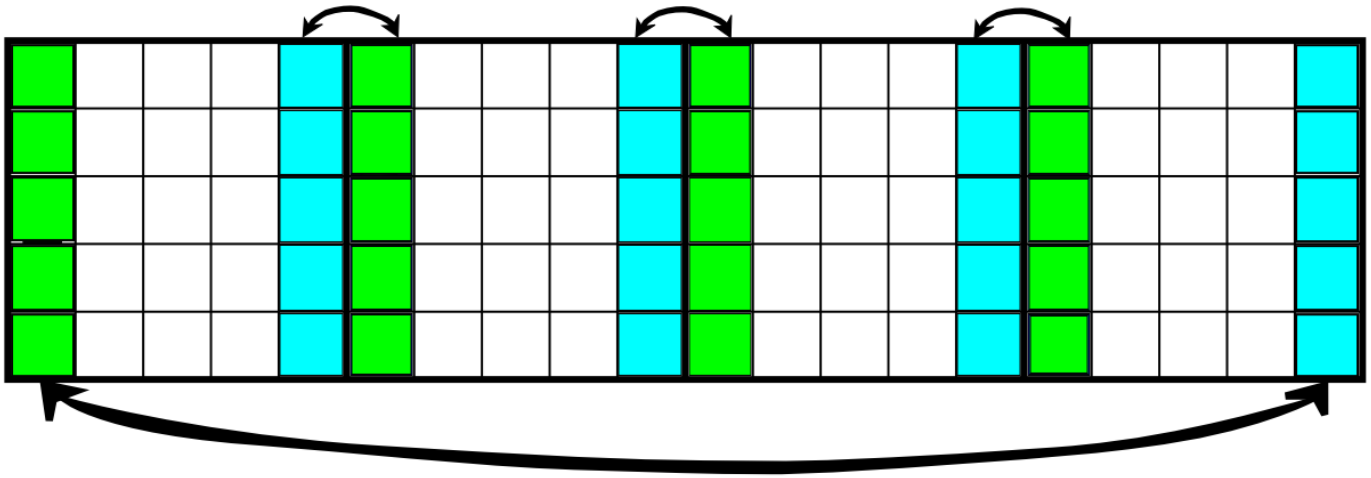

Abbildung: Datenaustausch zwischen den Matrizen

Programmbeispiel

```
1  program topobsp
2     use mpi
3     implicit none
4     integer :: ierr, irank, wsize
5
6     call MPI_INIT(ierr)
7     call MPI_COMM_RANK(MPI_COMM_WORLD, irank, ierr)
8     call MPI_COMM_SIZE(MPI_COMM_WORLD, wsize, ierr)
9     ...
10    call MPI_FINALIZE(ierr)
11
12 end program topobsp
```

Listing 3: Start der MPI-Umgebung

```
1  ...
2  integer, parameter :: ndims = 1
3  integer            :: dims = 4
4  logical            :: periods = .true., reorder = .true.
5  integer            :: comm_cart, coords
6  integer            :: direction = 0, disp = 1, rank_source,
       ↪ rank_dest
7  ...
8  ! Bestimmung der Nachbarn mithilfe der kartesischen Topologie
9  call MPI_CART_CREATE(MPI_COMM_WORLD, ndims, dims, periods,
       ↪ reorder, comm_cart, ierr)
10 call MPI_CART_COORDS(comm_cart, irank, ndims, coords, ierr)
11 call MPI_CART_SHIFT(comm_cart, direction, disp, rank_source,
       ↪ rank_dest, ierr)
12 write(*,'(a17,i2,a16,i2,a13,i2,a4,i2)') "Prozess des Rangs",
       ↪ irank, &
13 " mit Koordinaten", coords, " hat Nachbarn", rank_source, "
       ↪ und", rank_dest
14 ...
```

Listing 4: Erzeugen der kartesischen Topologie

Ausführung des Programmteils

```
pils@west1:~/EP/progbsp$ mpiexec -n 4 ./topobsp2.x
Prozess des Rangs 0 mit Koordinaten 0 hat Nachbarn 3 und 1
Prozess des Rangs 1 mit Koordinaten 1 hat Nachbarn 0 und 2
Prozess des Rangs 2 mit Koordinaten 2 hat Nachbarn 1 und 3
Prozess des Rangs 3 mit Koordinaten 3 hat Nachbarn 2 und 0
```

Abbildung: Ausgabe des Programmcodes

```
1  ...
2    integer                              :: i, j
3    integer, parameter                   :: msize = 5
4    integer, dimension(msize,msize) :: matrix
5  ...
6    ! Erstellen einer Beispielmatrix
7    do i = 1, msize
8       do j = 1, msize
9          matrix(j,i) = irank
10      end do
11   end do
12 ...
```

Listing 5: Erstellen einer Beispielmatrix

```
1  ...
2  integer     :: status(MPI_STATUS_SIZE)
3    ...
4  ! Austausch der ersten und letzten Spalte durch Senden und
       ↪ Empfangen
5    do i = 0, wsize - 1
6       call MPI_SEND(matrix(1,:), msize, MPI_INTEGER, rank_source,
       ↪ 2020, MPI_COMM_WORLD, ierr)
7       call MPI_SEND(matrix(msize,:), msize, MPI_INTEGER,
       ↪ rank_dest, 2021, MPI_COMM_WORLD, ierr)
8    end do
9    do i = 0, wsize - 1
10      call MPI_RECV(matrix(msize,:), msize, MPI_INTEGER,
       ↪ rank_dest, 2020, MPI_COMM_WORLD, status, ierr)
11      call MPI_RECV(matrix(1,:), msize, MPI_INTEGER, rank_source,
       ↪ 2021, MPI_COMM_WORLD, status, ierr)
12   end do
13 ...
```

Listing 6: Austausch der jeweils ersten und letzten Spalte

```
1  ...
2    ! Ausgabe der veraenderten Beispielmatrix
3    if (irank == 0) then
4       write(*,*) "Ausgabematrix des Prozesses von Rang 0"
5       do j = 1, msize
6          write(*,'(i2," *",i2," *",i2," *",i2," *",i2)')
       ↪ matrix(:,j)
7       end do
8    end if
9    ...
```

Listing 7: Ausgabe der Teilmatrix des Masterprozesses

Ausführung des Programms

```
pils@west1:~/EP/progbsp$ mpiexec -n 4 ./topobsp.x
 3 * 0 * 0 * 0 * 1
 3 * 0 * 0 * 0 * 1
 3 * 0 * 0 * 0 * 1
 3 * 0 * 0 * 0 * 1
 3 * 0 * 0 * 0 * 1
```

Abbildung: Ausgabe des Programmcodes

Zusammenfassung

- in beispielsweise der Meteorologie sind sehr große Gitter vorhanden
  $\rightarrow$ paralleles Rechnen erforderlich
- MPI ist ein Standard für den Nachrichtenaustausch bei parallelen Rechnungen
- zwischen MPI_INIT und MPI_FINALIZE befindet sich der parallele Bereich
- virtuelle Topologien vereinfachen den nachfolgenden Nachrichtenaustausch
- in der Meteorologie wird insbesondere die kartesische Topologie benötigt, die mithilfe von MPI_CART_CREATE erstellt werden kann

Literatur I

[BMM+] Wolfgang Baumann, Neil MaxDonald, Espeth Minty, Tim Harding, and Simon Brown. Writing message-passing parallel programs with mpi. course notes. *Edinburgh Parallel Computing Centre*.

[DWD] DWD. Globalmodell icon. https://www.dwd.de/DE/forschung/wettervorhersage/num_modellierung/01_num_vorhersagemodelle/icon_beschreibung.html.

[DWD17] DWD. Wettermodelle, 2017. https://www.dwd.de/DE/fachnutzer/luftfahrt/download/produkte/wettermodelle/wettemodelle_download.pdf?_blob=publicationFile.

Literatur II

[For15] Message Passing Interface Forum. Mpi: A message-passing interface standard. version 3.1. Technical report, Knoxville, TN, USA, 06 2015. https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[GLS07] William Gropp, Ewing Lusk, and Anthony Skjellum. *MPI - Eine Einführung. Portable parallele Programmierung mit dem Message-Passing Interface*. Oldenbourg Wissenschaftsverlag GmbH, 2007.

Literatur III

[LLB19] Prof. Thomas Ludwig, Hermann Lenhart, and Michael Blesel. Praktikum: Paralleles programmieren für geowissenschaftler. Universität Hamburg, 2019. https://wr.informatik.uni-hamburg.de/teaching/sommersemester_2019/einfuehrung_in_parallele_programmierung_fuer_geowissenschaftler.