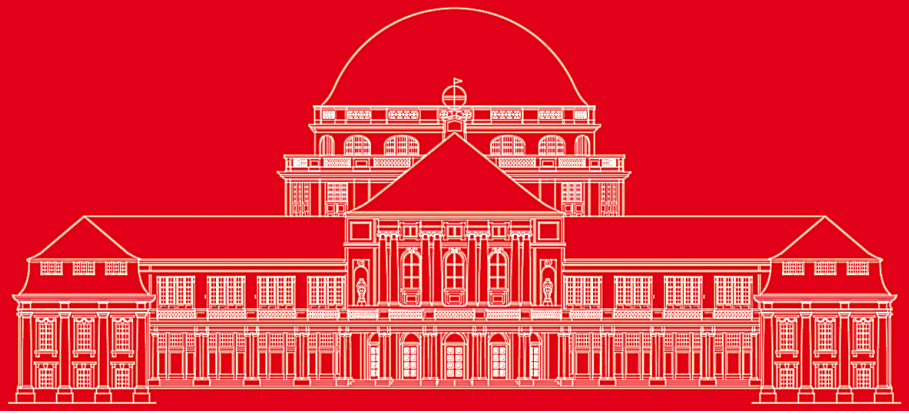
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart & Felix Hoffmann



Dr. Hermann-J. Lenhart
hermann.lenhart@informatik.uni-hamburg.de

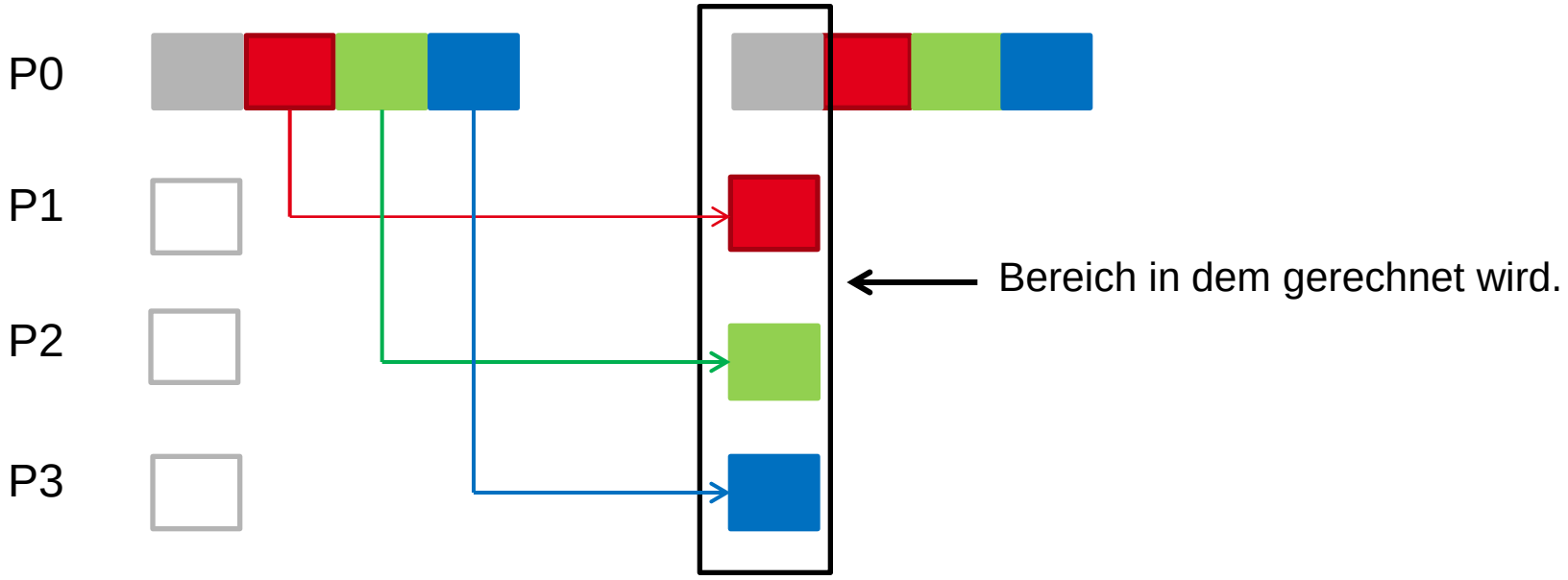
MPI Einführung III

- Scatterv / Gatherv
- Programmbeispiele
- Parellele Bearbeitung einer Matrix

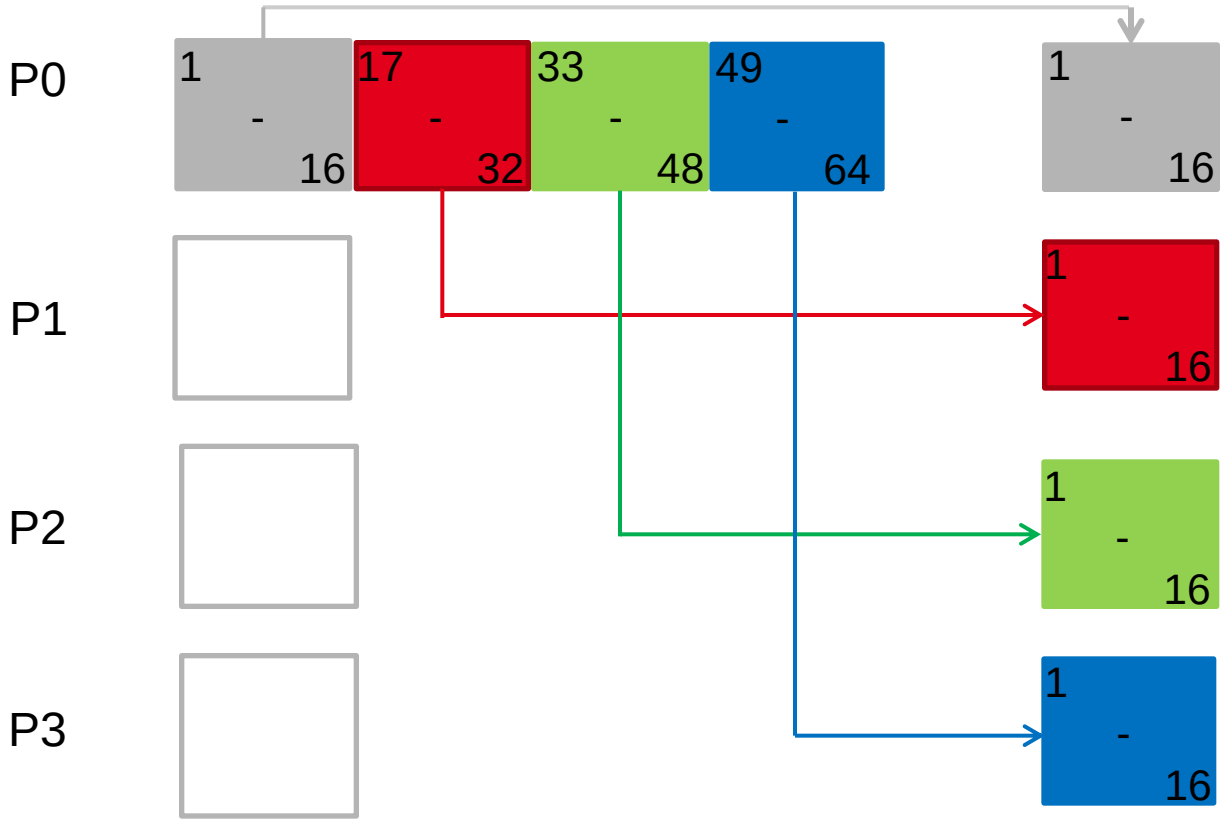
Dr. Hermann-J. Lenhart
hermann.lenhart@informatik.uni-hamburg.de

MPI Scatter I

Die Daten werden von P0 an die anderen Prozesse P1 – P3 gesendet.



MPI Scatter: Beispiel array2D(8X8) => 4 Teile chunk2D(4X2)

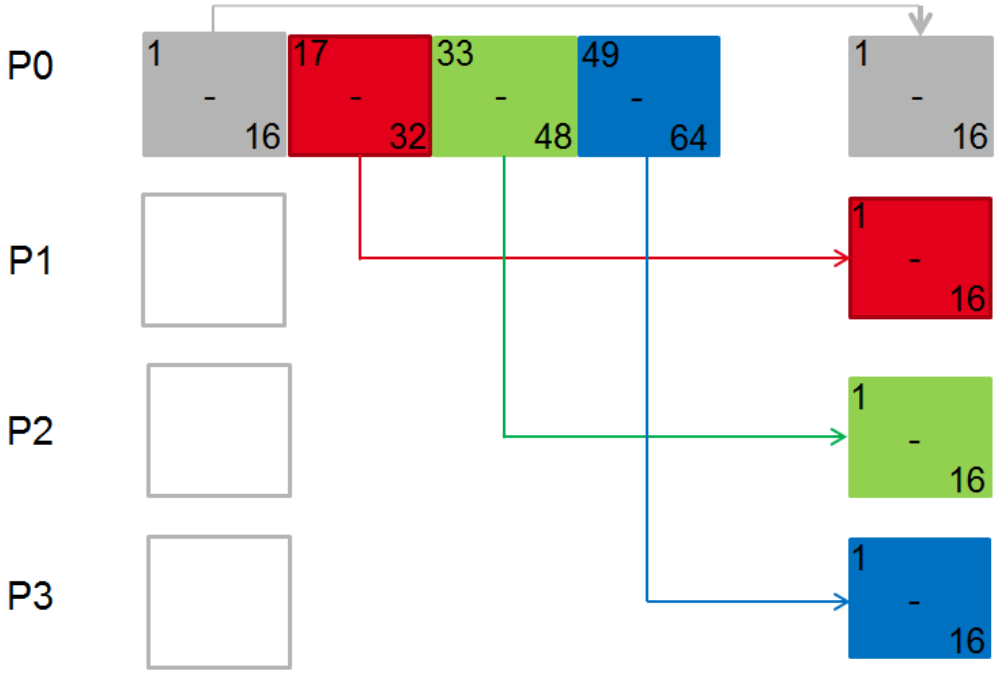


MPI Scatter:

```
integer, dimension(8, 8) :: array2D !   all the 2D data
integer, dimension(8, 2) :: chunk2D ! piece of 2D data each process works on

 ! int MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)
 ! 16 = 8*2 = size(chunk)
CALL MPI_SCATTER(array2D, 16, MPI_INTEGER, chunk2D, 16, MPI_INTEGER, &
0, MPI_COMM_WORLD, mpi_ierr)
```

Abbildung von Matrix
array2D auf 4 Teile chunk2D

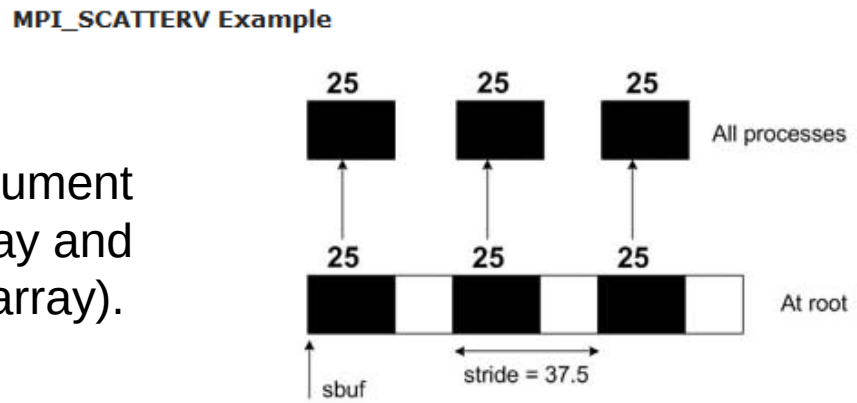


MPI Scatter-Gather: In der Literatur

MPI_Gatherv and MPI_Scatterv are the variable-message-size versions of MPI_Gather and MPI_Scatter.

MPI_Scatterv extends the functionality of MPI_Scatter to permit a varying count of data from each process.

It does this by changing the count argument from a single integer to an integer array and providing a new argument displs (an array).



(Quelle: https://www.cac.cornell.edu/vw/MPIcc/gathervscatterv.aspx)

MPI Scatter -> ScatterV

Call MPI_SCATTER (Send_Message, Send_Count, Send_Datatype,
Recv_Message, Recv_Count, Recv_Datatype,
Root, *Comm*, Ierror)

Call MPI_SCATTERV(Send_Mesage, Send_Count, **Displacement**, Send_Datatype,
Recv_Mesage, Recv_Count, Recv_Datatype,
Root, *Comm*, Ierror)

Änderung: ***Send_count*** ***von Integer - Zahl*** → ***Integer-Vektor***

Neu: ***Displacement-Vektor***

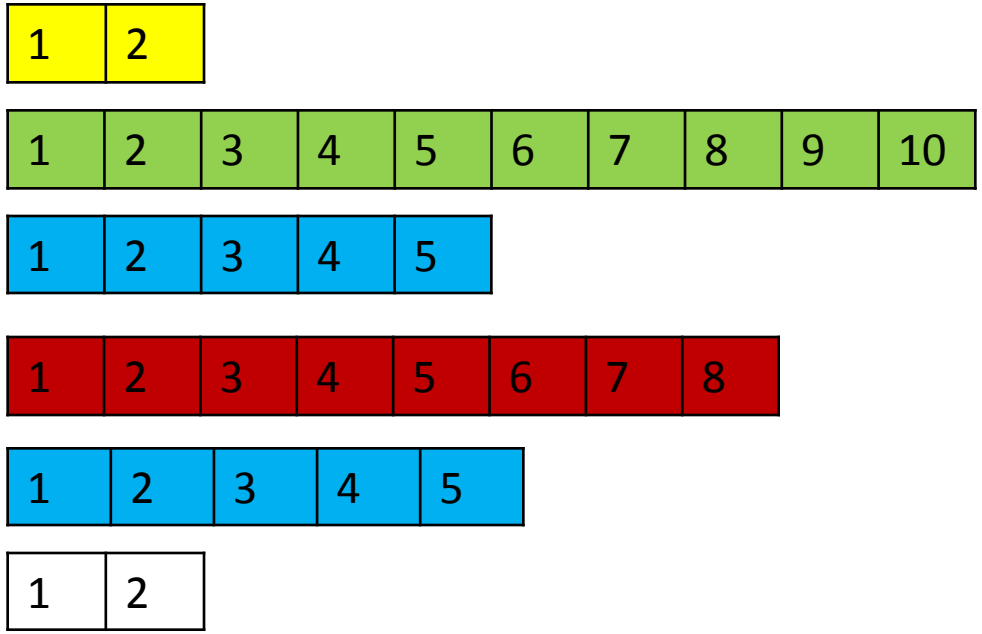
MPI Scatterv – Beispiel A

1	2								10
11	12					17			20
				25					30
	32		34		36		38		40

Aufteilung Matrix 10x4

auf einzelne Vektoren

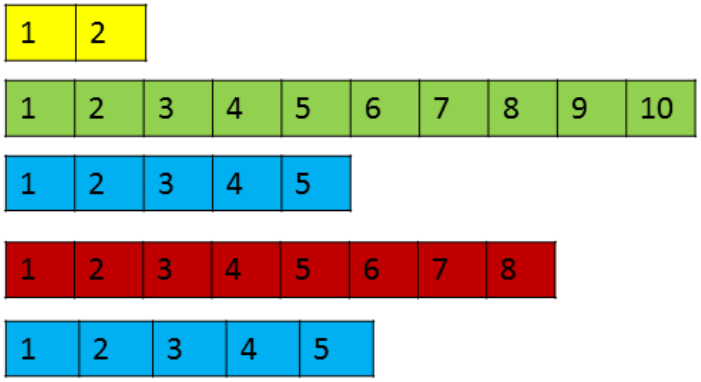
unterschiedlicher Länge



MPI Scatterv – Beispiel A - **Datenstruktur**

Für 10 Prozesse:

integer :: sendcount (10)
integer :: displacement (10)



sendcounts = (/2, 10, 5, 8, 5, 2, 2, 2, 2, 2/) 5 X 1 2

displacement = (/0, 2, 12, 17, 25, 30, 32, 34, 36, 38/)

Displacement:
Beginnt mit Null und bezieht sich jeweils auf das letzte Element des vorherigen Abschnitts!

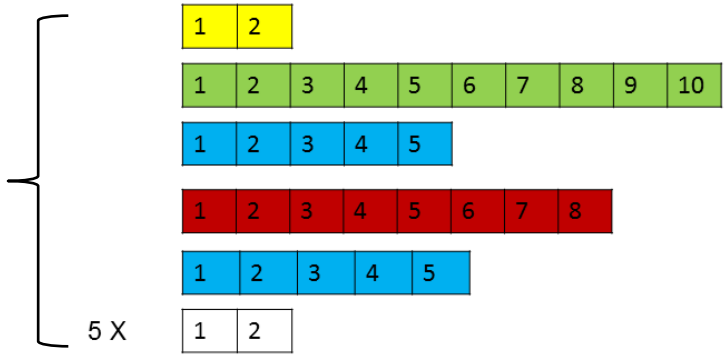
MPI Scatterv – Beispiel A – **Programmablauf**

10 Prozesse: max_rank = 10

```
integer, allocatable, dimension(:,:) :: array, chunk
integer, allocatable, dimension(:) :: displacement, sendcounts
```

MPI_init ……

```
allocate(sendcounts(max_rank))
sendcounts = (/2, 10, 5, 8, 5, 2, 2, 2, 2, 2/)
allocate (chunk(sendcounts(rank+1)))
```




```
if (rank == 0) then
allocate(array(max_rank, kWidth))
allocate(displacement(max_rank))
displacement = (/0, 2, 12, 17, 25, 30, 32, 34, 36, 38/)
endif
```


```
call MPI_SCATTERV(array, sendcounts, displacement, MPI_INTEGER,
                  chunk, sendcounts(rank+1),      MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
```

MPI Scatterv – Beispiel A – **Zuweisung I**

Die Matrix wir in Teilen verschickt!

sendcounts definiert die Größe der Datenpakete pro Prozess

sendcounts = (/2, 10, 5, 8, 5, 2, 2, 2, 2, 2/)

displacement = (/0, 2, 12, 17, 25, 30, 32, 34, 36, 38/)

displacement den Versatz in der Matrix

```
call MPI_SCATTERV(array, sendcounts, displacement, MPI_INTEGER,
                  chunk, sendcounts(rank+1),       MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
```

MPI Scatterv – Beispiel A – **Zuweisung II**

Die Matrix wir in Teilen verschickt!

sendcounts definiert die Größe der Datenpakete pro Prozess

sendcounts = (/2, 10, 5, 8, 5, 2, 2, 2, 2, 2/)

displacement = (/0, 2, 12, 17, 25, 30, 32, 34, 36, 38/)

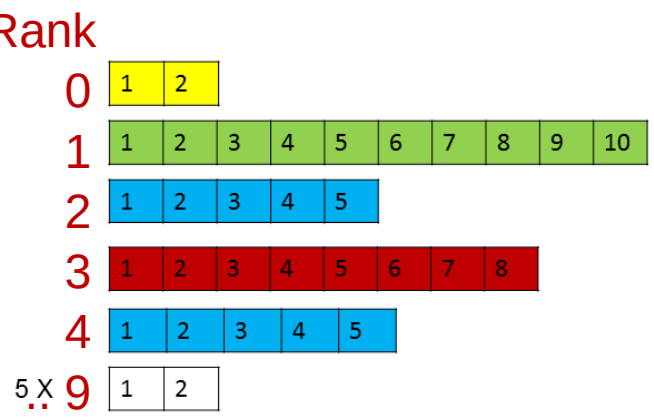
1	2								10
11	12					17			20
				25					30
	32		34		36		38		40

displacement den Versatz in der Matrix

```
call MPI_SCATTERV(array, sendcounts, displacement, MPI_INTEGER,
                  chunk, sendcounts(rank+1),       MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
```

Die Datenpakete werden den einzelne Vektoren (chunk) pro Prozess zugewiesen

Entsprechend der variablen Größe pro Prozess definiert durch sendcount(rank+1)



MPI Scatterv – Beispiel B

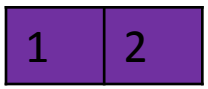
1	2								10
11	12					17			20
				25					30
	32		34		36		38		40

Aufteilung Matrix 10x4

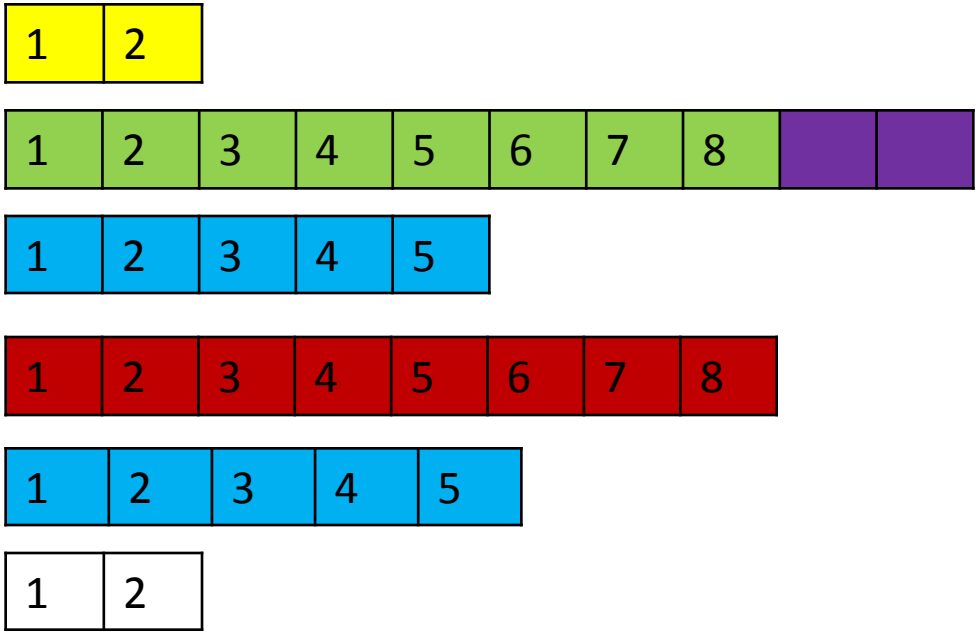
auf einzelne Vektoren

unterschiedlicher Länge

mit Lücke in den Daten



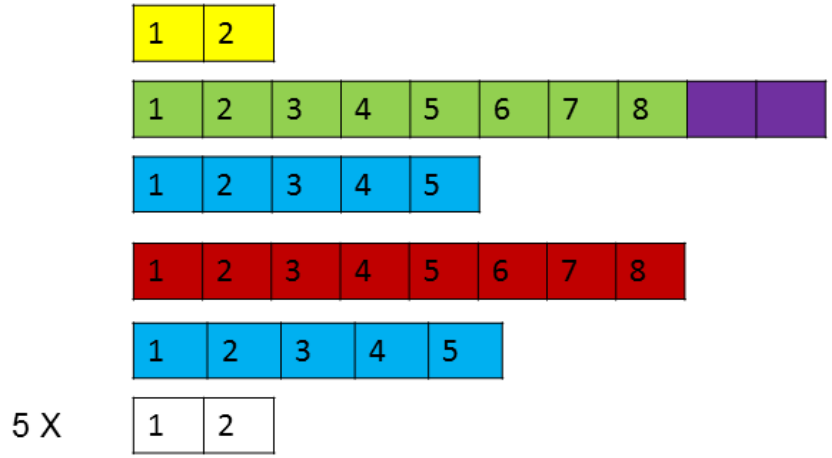
5 X



MPI Scatterv – Beispiel B - **Datenstruktur**

Für 10 Prozesse:

integer :: sendcount (10)
Integer :: displacement (10)



sendcounts = (/2, **8,** 5, 8, 5, 2, 2, 2, 2, 2/)

displacement = (/0, 2, 12, 17, 25, 30, 32, 34, 36, 38/)

sendcounts wird an neue Datenstruktur angepasst, Position 11 und 12 wird ausgelassen

displacement: hier ändert sich in diesem Beispiel nichts.

MPI Gatherv Vergleich zu Scatterv aus Beipiel B

10 Prozesse: max_rank = 10

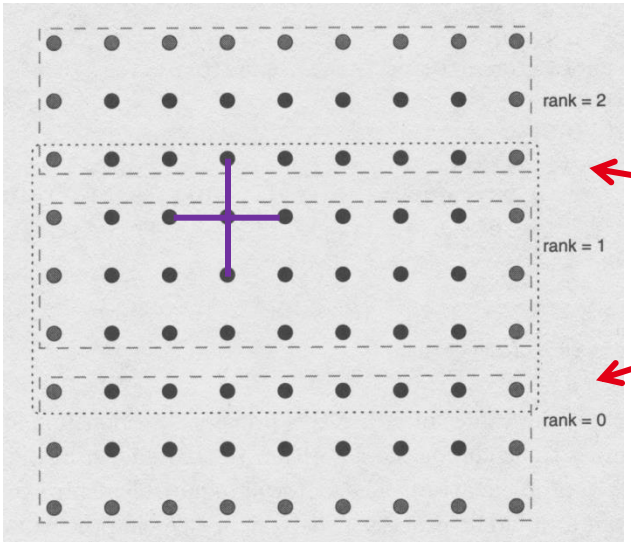
sendcounts = (/2, 8, 5, 8, 5, 2, 2, 2, 2, 2/)

displacement = (/0, 2, 12, 17, 25, 30, 32, 34, 36, 38/)

```
call MPI_SCATTERV(array, sendcounts, displacement, MPI_INTEGER,
     chunk, sendcounts(rank+1),      MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
```

```
call MPI_GATHERV(chunk, sendcounts(rank+1), MPI_INTEGER,
    array, sendcounts, displacement, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
```

Parallele Bearbeitung einer Matrix I



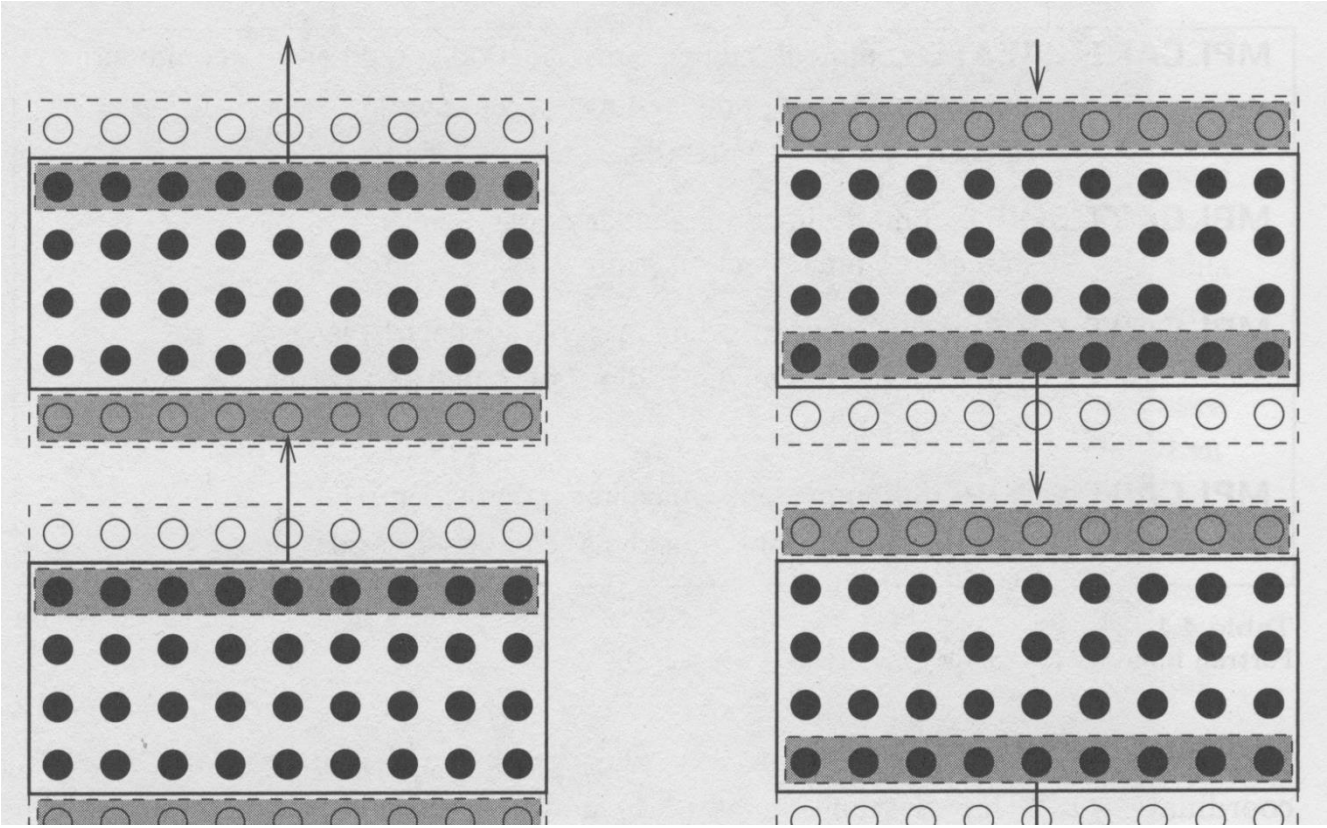
Überlappung der Teilmatrizen der einzelnen Prozesse

Berechnung der Poisson Gl.

auf *Star*

Quelle:
Gropp, Lusk & Skjellum
Using MPI

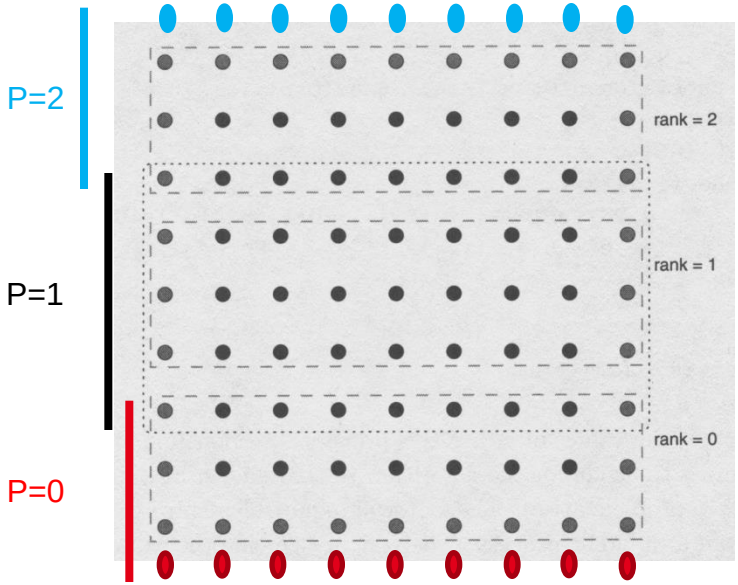
Parallele Bearbeitung einer Matrix II



Austausch
der Randstreifen
im Programmlauf

Quelle:
Gropp, Lusk & Skjellum
Using MPI

Parallele Bearbeitung einer Matrix III

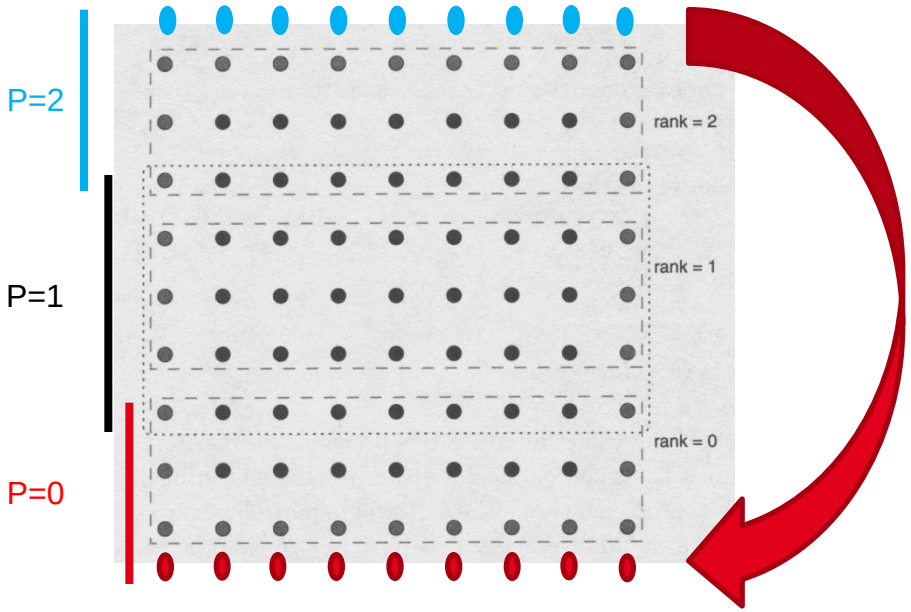


Geänderte Matrix
für unsere Applikationen

(ohne rechten und linken Rand)

Quelle:
Gropp, Lusk & Skjellum
Using MPI

Parallele Bearbeitung einer Matrix IV



Geänderte Matrix
für unsere Applikationen
mit zyklischen
Randbedingungen

Quelle:
Gropp, Lusk & Skjellum
Using MPI

Hilfestellung zur Matrixaufteilung: data(N,M) -> partData(:)

```
real*8, allocatable, dimension(:,:) :: data, partData
integer, dimension(:), allocatable :: sendCounts, recvCounts, displs

allocate(data(N,M)
allocate(partData(N,chunksize+2))
```

chunksize nach Aufteilung auf Prozessoren
+2 für Halo-Linien

```
call MPI_SCATTERV(data,sendCounts,displs,MPI_DOUBLE_PRECISION,&
     partData(1,2),recvCounts,MPI_DOUBLE_PRECISION,master,MPI_COMM_WORLD,ierr)
```

Definiert Aufsatzpunkt ab welcher Position Daten geschrieben werden,
(im Gegensatz zur displs-Logik „Punktgenau“!)

bzw:

```
call MPI_GATHERV(partData(1,2),recvCounts,MPI_DOUBLE_PRECISION, &
     data,sendCounts,displs,MPI_DOUBLE_PRECISION,master,MPI_COMM_WORLD,ierr)
```

Danke das wars!

Noch Fragen?