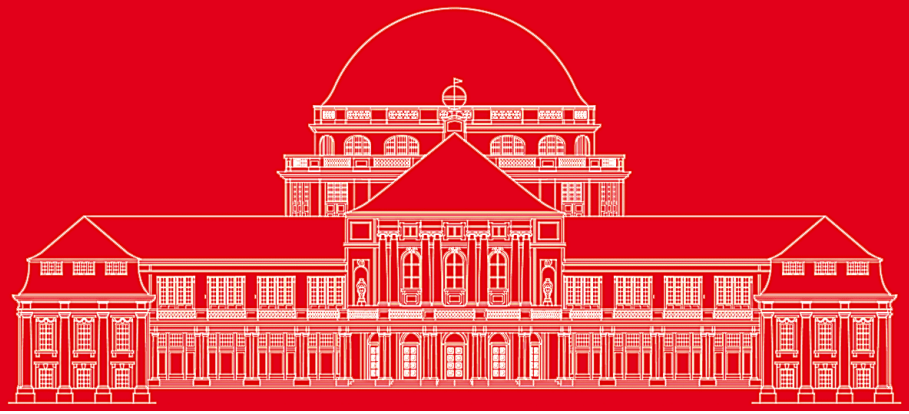
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart & Tim Jammer



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

MPI Einführung II: Kollektive Operationen

- Broadcast
- Reduce Operation
- Scatter / Gather => gleichmäßige Matrixaufteilung

MPI Kollektive Operationen

Neben **Point-to-Point Kommunikation** mittels Send & Recv

verfügt MPI über umfangreiche Operationen zum **kollektiven Bewegen von Daten**.

MPI_BROADCAST Eine Info an alle Prozesse versenden

MPI_REDUCE „aggregierende“ Operationen (Summe) auf Matrix ausführen

MPI_SCATTER Teilarrays an Prozesse übertragen

MPI_GATHER Teilarrays zusammenführen

MPI BROADCAST I

Neben dem Versenden von Nachrichten zwischen einzelnen Prozesse mittels

Call MPI_SEND(temp, 1, MPI_Real, dest, tag, MPI_COMM_World, Ierror)

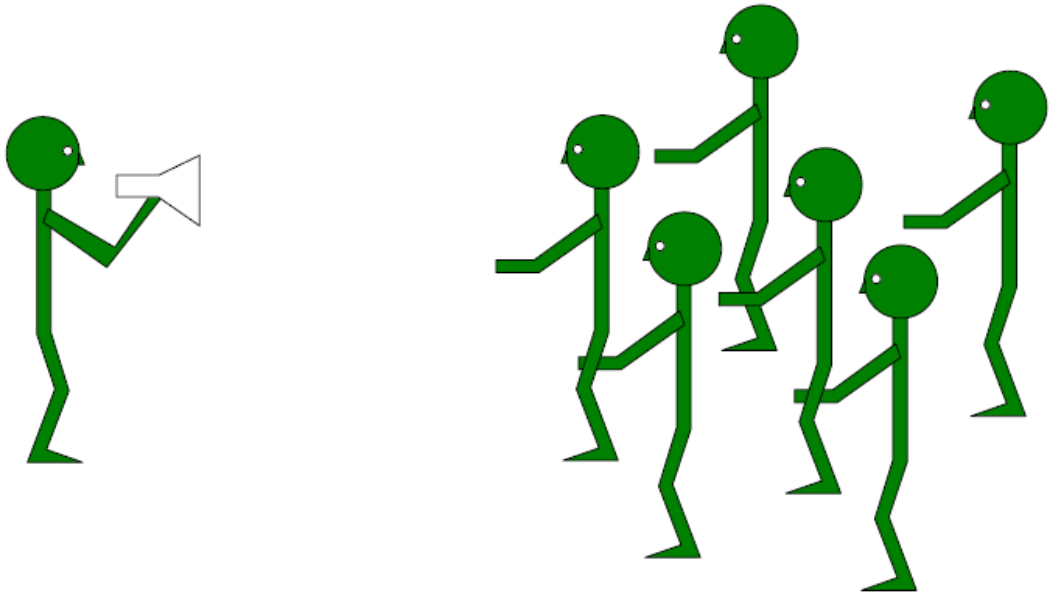
gibt es auch die Möglichkeit **eine Nachricht an alle anderen Prozesse** zu senden:

MPI_BCAST(Message, Count, Datatype, Root, Comm, Ierror)

Call MPI_BCAST(temp, 1, MPI_Real, source, MPI_COMM_World, Ierror)

Oft für Initialisierung oder zum Programmabbruch vom Master genutzt.

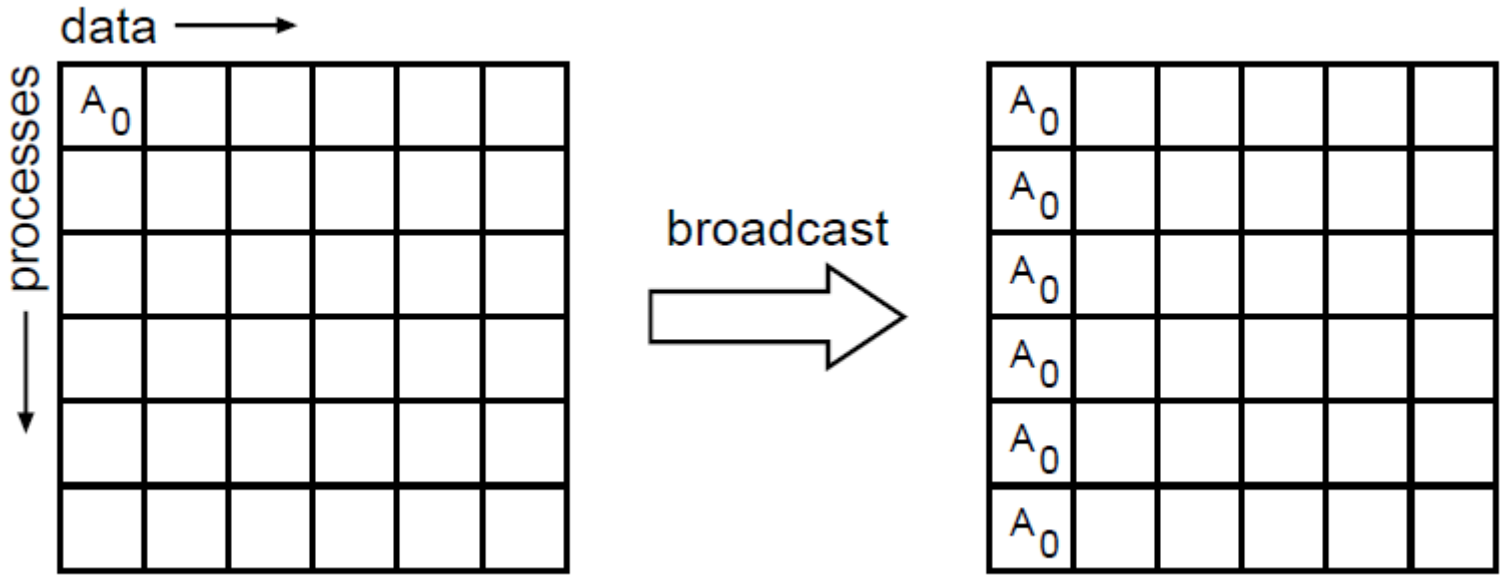
MPI Broadcast II



(Wolfgang Baumann ZIB, 2009;
Parallel Programming with MPI)

MPI Broadcast III

Verwendung zur Initialisierung



(William Gropp ANL, MPI Tutorial)

Es gibt eine interne Hirarchie der Zuteilung!

MPI Reduce I

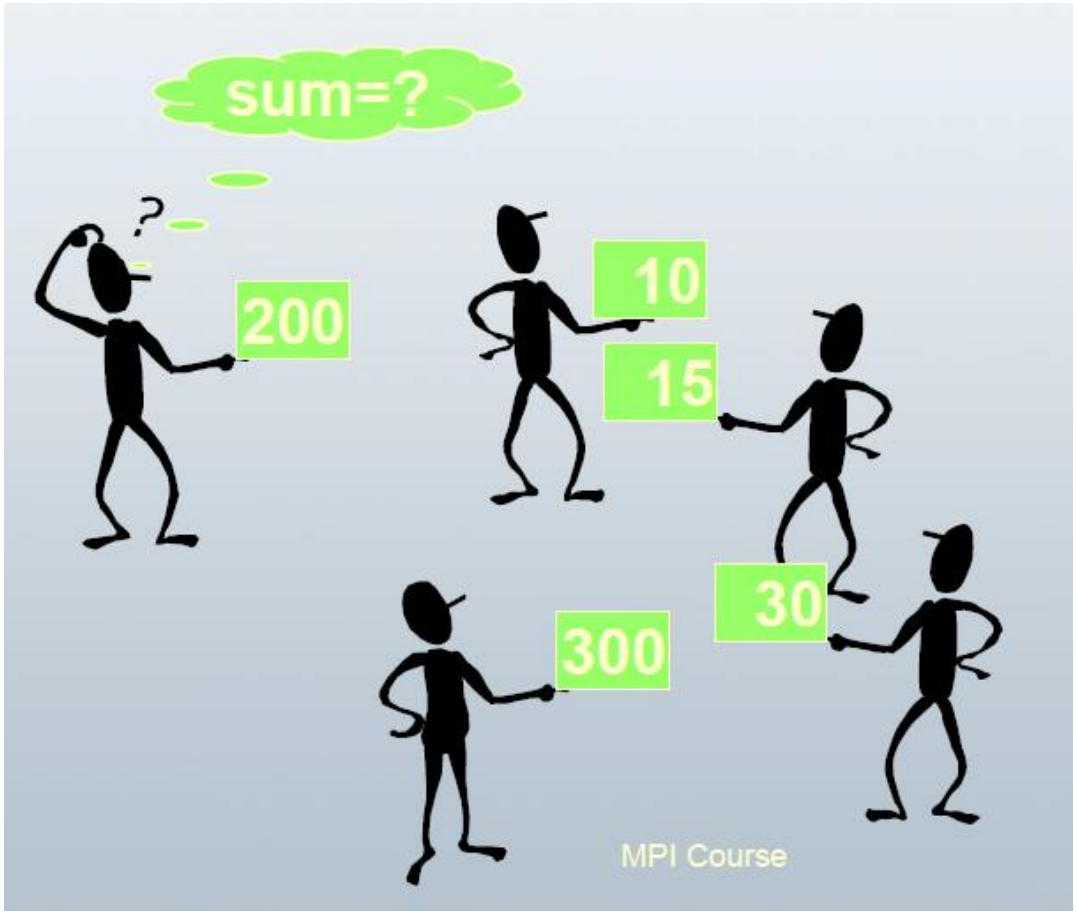
Um die Ergebnisse der einzelnen Prozesse zusammenzuführen gibt es eine Auswahl an „Reduce“ Operationen, z.B:

MPI_REDUCE(Operand, Result, Count, Datatype, Operation, Root, Comm, Ierror)

Call MPI_REDUCE(temp, sum,1, MPI_Real, MPI_SUM,0, MPI_COMM_World, Ierror)

Über die Operation MPI_SUM werden alle Resultate der Größe temp von allen Prozessen aufaddiert und in der Variable sum abgelegt.

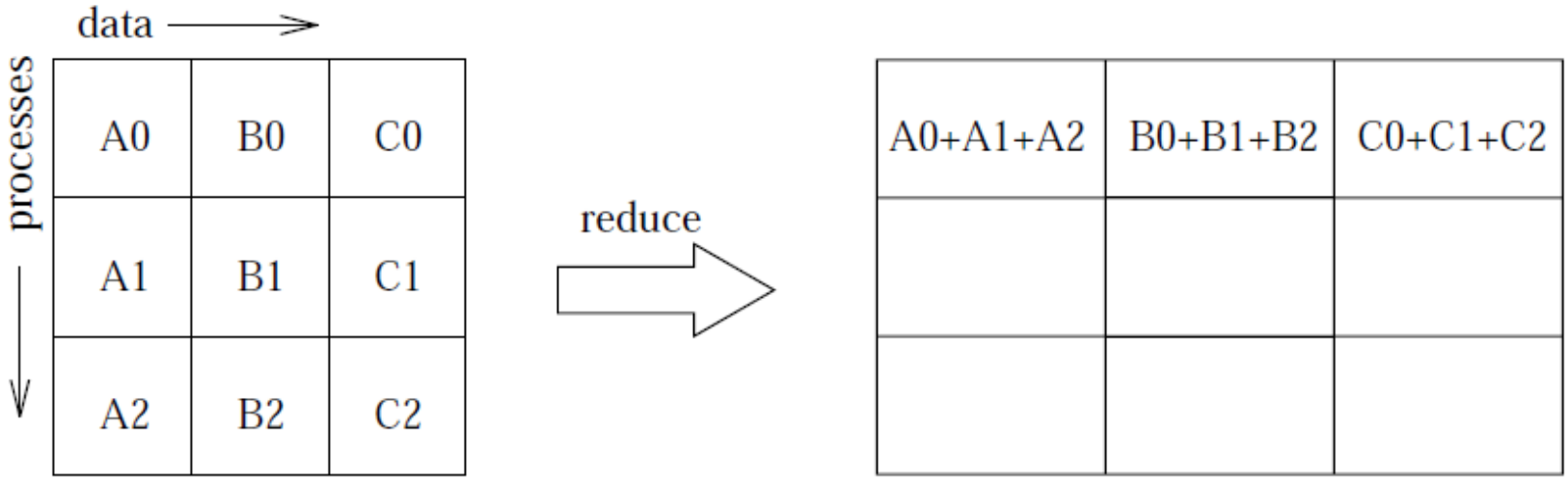
MPI Reduce II



DKRZ MPI Einführungs Kurs

MPI Reduce III

Operator SUM



(William Gropp ANL, MPI Tutorial)

MPI Reduce IV

Übersicht der möglichen MPI_Reduce Opertionen:

MPI_SUM	Summe
MPI_PROD	Produkt
MPI_MAX /MPI_MIN	Maximum/Minimum
MPI_MAXLOC	Maximum und Position des Maximums
MPI_LAND / MPI_LOR	Logical And / Logical Or

MPI Reduce V: Beispiel Pi-Berechnung

```
DOUBLE PRECISION   :: pi, pi_out, delta_x
CALL MPI_INIT(ierr)
x_start =  …  ,  x_end   =  ….. ,   delta_x = ….., pi = 0
 DO x=x_start,x_end
            pi = pi + 4 / (1 + (x/REAL(x_final))**2) * delta_x
END DO

CALL MPI_REDUCE(pi,pi_out,1,mpi_double_precision,mpi_sum,0,mpi_comm_world,ierr)
 IF (rank .EQ. 0) WRITE (*,*) 'Final result is: Pi = ', pi_out

CALL MPI_FINALIZE(ierr)
END PROGRAM pi_Bestimmung
```

MPI Scatter I

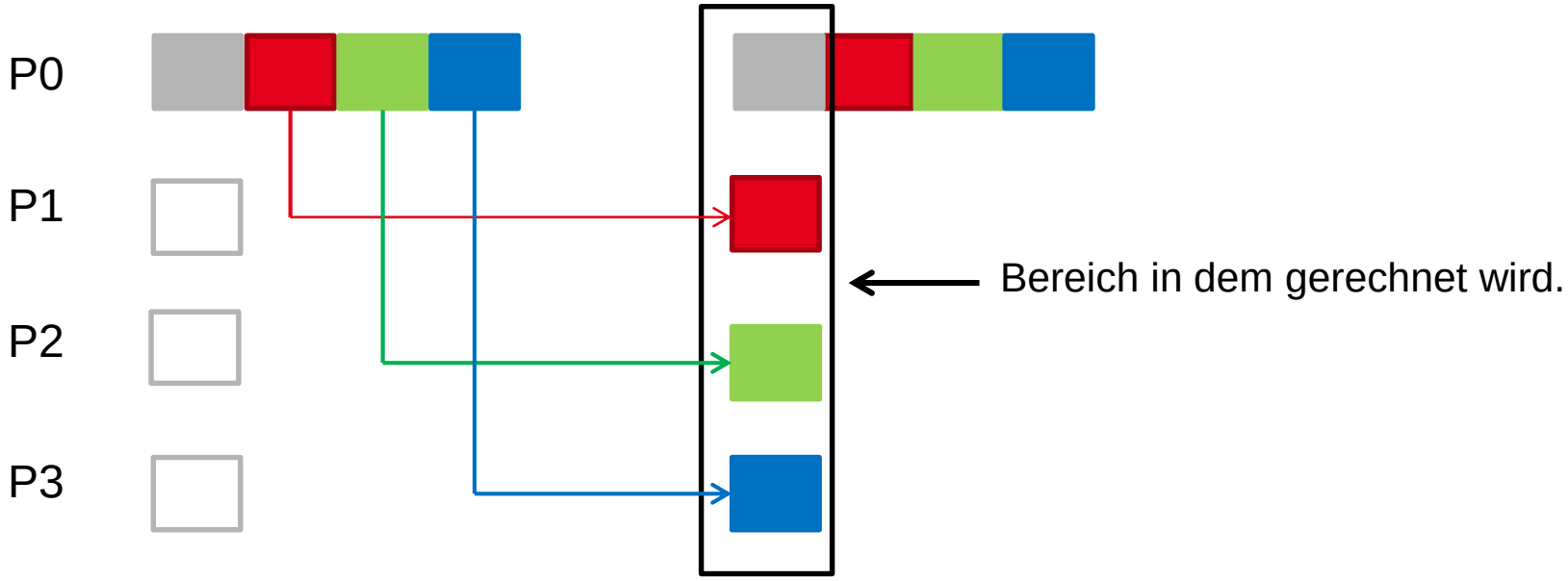
Eine Möglichkeit z.B. eine Anfangsbelegung auf die Teilarrays der Prozesse, zu übertragen bietet MPI_SCATTER:

Syntax: MPI_Scatter(Sendbuffer, Sendcount, Sendtype,
Recvbuffer, Recvcount, Recvtype,
Root, Comm, Ierror)

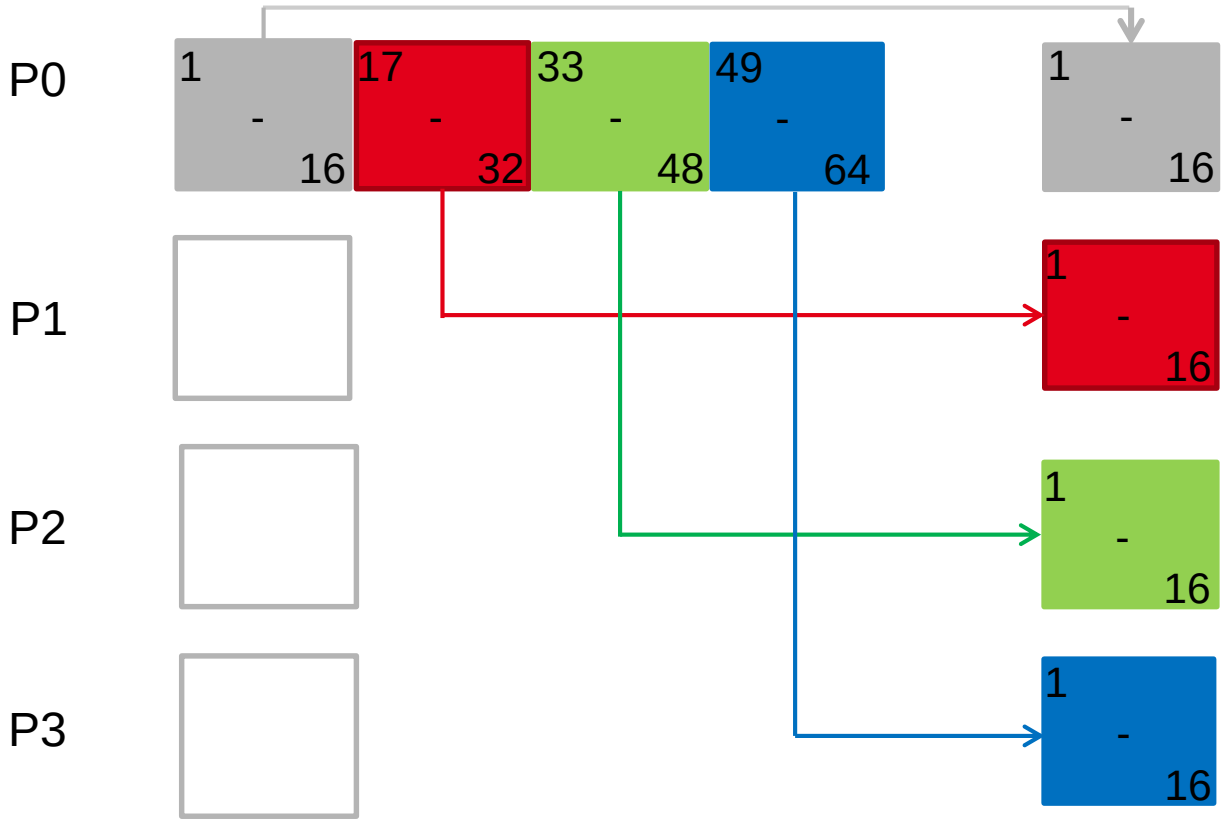
Call MPI_SCATTER(Send_Message, Send_Count, Send_Datatype,
Recv_Message, Recv_Count, Recv_Datatype,
0, MPI_COMM_World, Ierror)

MPI Scatter II

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



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



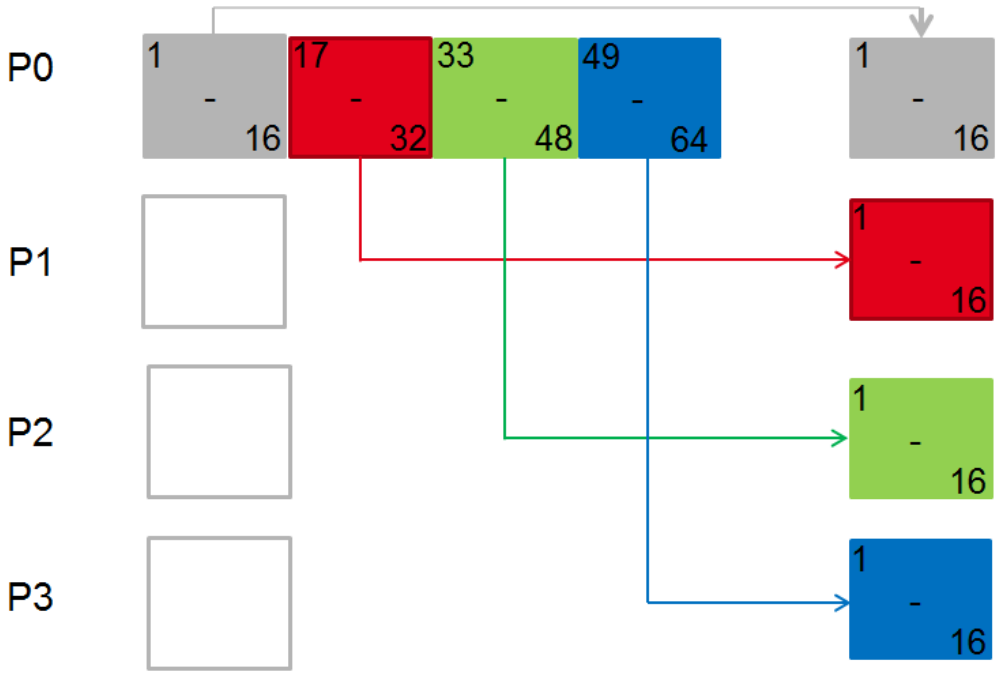
MPI Scatter:

integer, dimension(8, 8) :: array2D ! all2 2D Daten
integer, dimension(8, 2) :: chunk2D ! Teile der 2D Daten für jeden einzelne Prozess (4 Stück)

! 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:

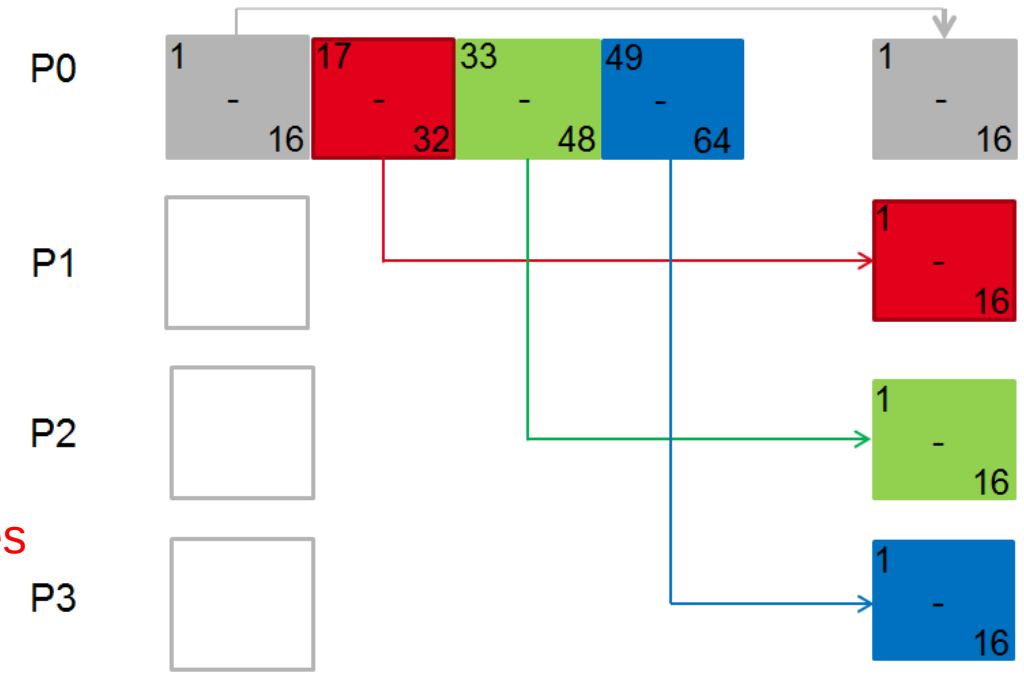
```
integer, dimension(8, 8) :: array2D !   all2 2D Daten
integer, dimension(8, 2) :: chunk2D ! Teile der 2D Daten für jeden einzelne Prozess (4 Stück)

 !  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

Die Anzahl der beteiligten
Prozesse wird intern angenommen;
aber nicht explizit ausgewiesen,
d.h. die Zuordnung der Matrix Indizes
muss intern abgestimmt werden!



MPI Gather I

Eine Möglichkeit Teilarrays der Prozesse (z.B. für I/O Zwecke) wieder zusammenzuführen, bietet MPI_GATHER:

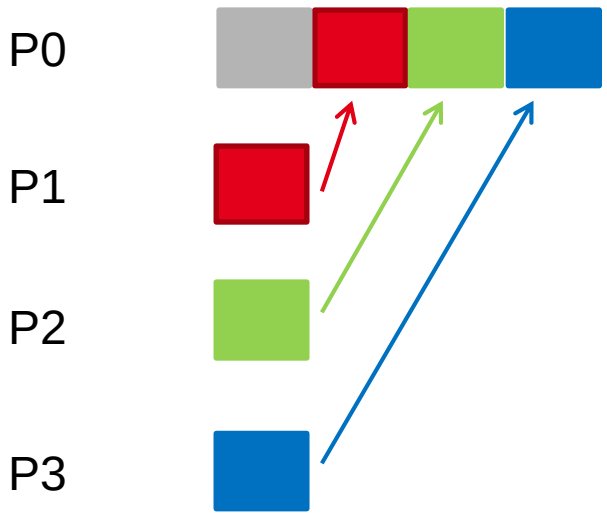
Syntax: MPI_Gather(Send_Message, Send_Count, Send_Datatype,
Recv_Message, Recv_Count, Recv_Datatype,
Root, Comm, Ierror)

Call MPI_GATHER (temp, 1, MPI_Real,
temps,1, MPI_Real,
0, MPI_COMM_World, Ierror)

MPI Gather II

Call MPI_GATHER(temp, 1, MPI_Real,

tempAll,1, MPI_Real, 0, MPI_COMM_World, Ierror)



MPI Gather III

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


    ! 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)

    CALL computation(chunk2D)  ! where the magic happens

    ! Chunks are gathered by process 0.
    CALL MPI_GATHER(chunk2D, 16, MPI_INTEGER, array2D, 16, MPI_INTEGER, 0,
MPI_COMM_WORLD, mpi_ierr)
```

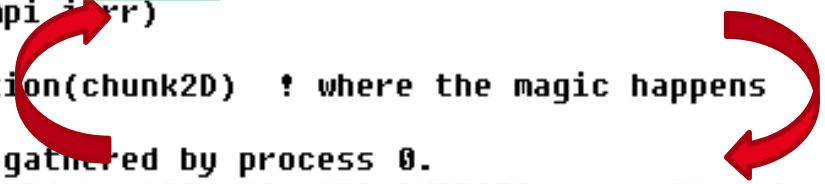
MPI Gather III

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


    ! 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)

    CALL computation(chunk2D)  ! where the magic happens

    ! Chunks are gathered by process 0.
    CALL MPI_GATHER(chunk2D, 16, MPI_INTEGER, array2D, 16, MPI_INTEGER, 0,
MPI_COMM_WORLD, mpi_ierr)
```



Danke das wars!