Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart, Nathanael Hübbe

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

MPI Kollektive Operationen

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

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

Dazu gehören:

MPI_BROADCAST Eine Info an alle Prozesse versenden

MPI_REDUCE „aggregierende“ Operationen (Summe; Prod) auf Matrix ausführen

MPI_SCATTER Teilarrays an Prozesse übertragen

MPI_GATHER Teilarrays zusammenführen

MPI-SCATTERV variable Teilarrays an Prozesse übertragen

MPI-GATHERV variable Teilarrays von Prozessen zusammenführen

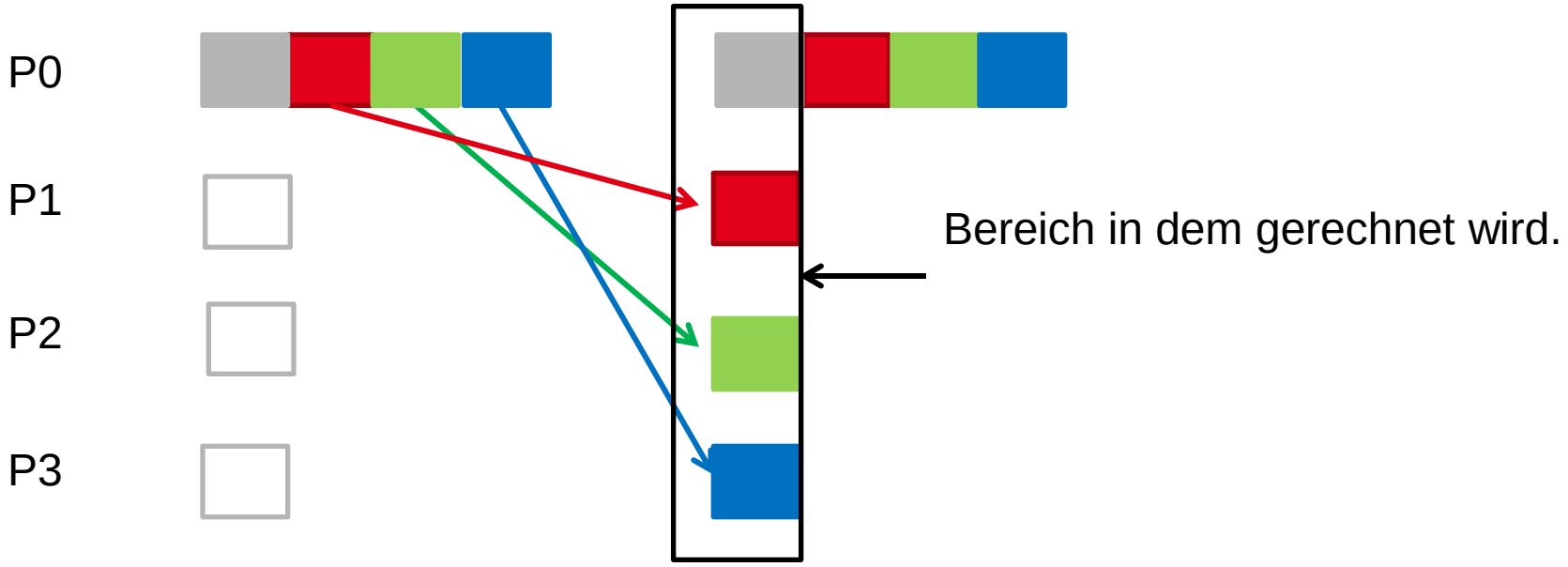
MPI Scatter I

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

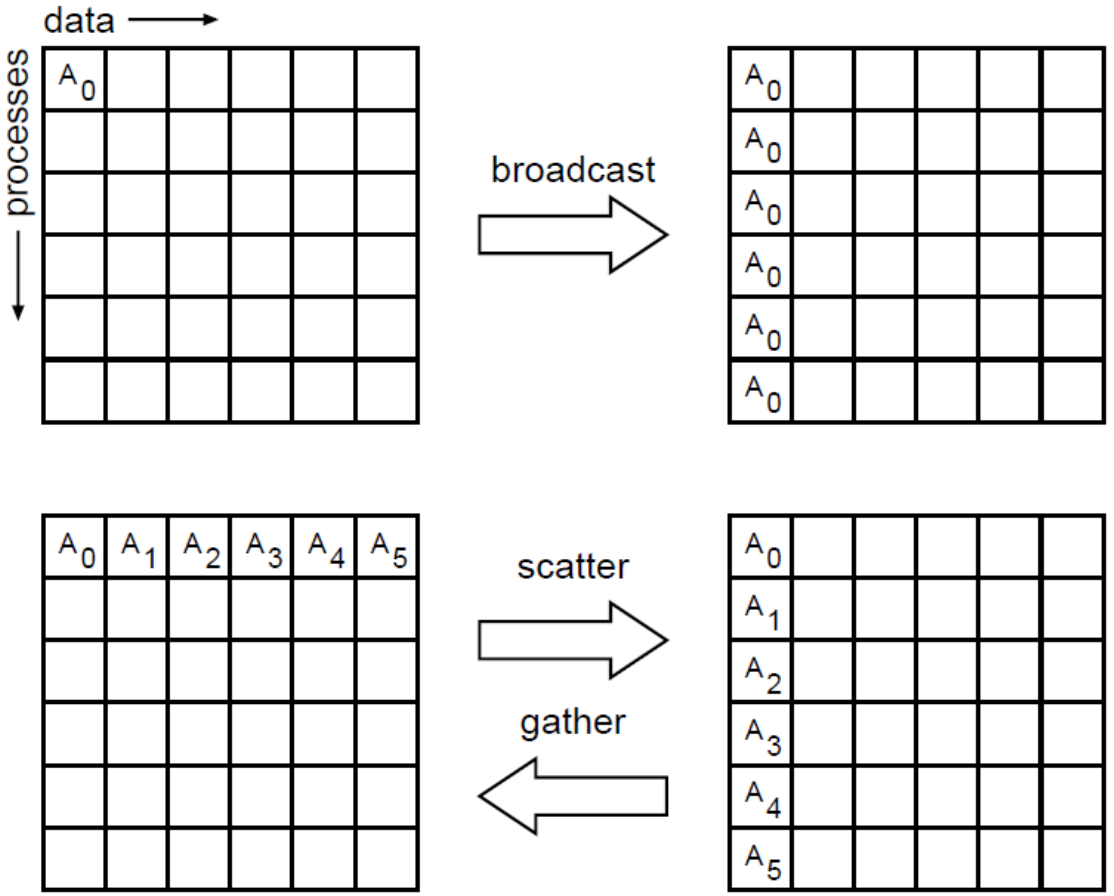
Syntax: MPI_Scatter(Sendmessage, Sendcount, Sendtype,
Recvmessage, Recvcount, Recvtype,
Root, Comm, Ierror)

MPI Scatter II

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



MPI Scatter-Gather



(William Gropp ANL, MPI Tutorial)

MPI Gather I

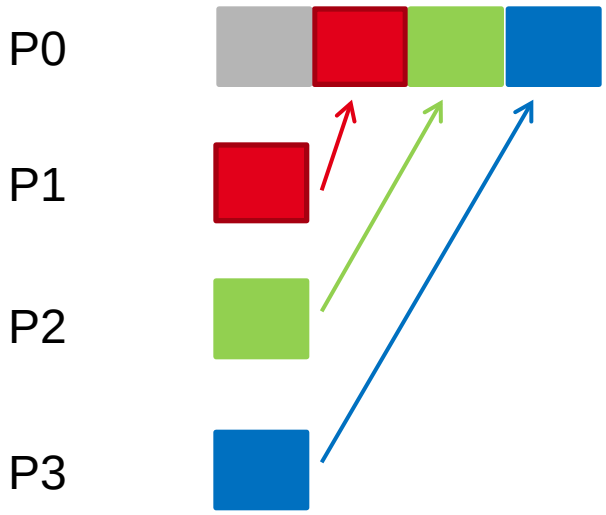
Eine weitere Möglichkeit Teilarrays der Prozesse (z.B. für I/O Zwecke) zusammenzuführen, bietet MPI_GATHER:

Syntax: MPI_Gather(Send_Message, Send_Count, Sendt_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 „Kommunikations Hirarchie I“

Call MPI_SEND (Message, Count, Datatype, Dest, Tag, *Comm*, Ierror)

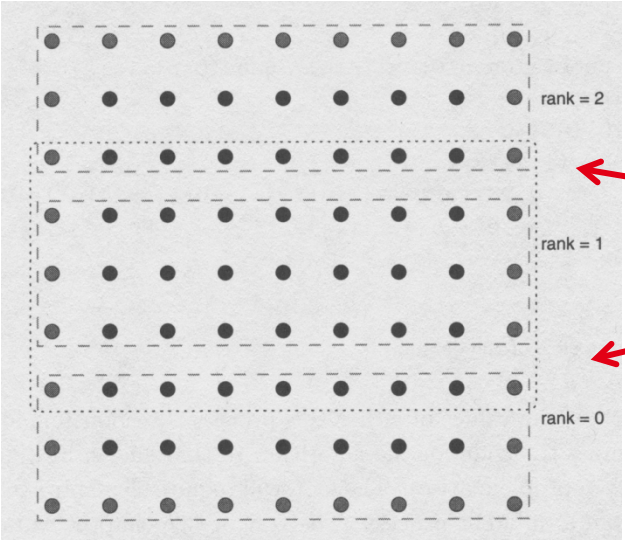
MPI_BCAST (Message, Count, Datatype, Root,*Comm*, Ierror)

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

Type : (Send_/Recv_) Message (*)

Integer : (Send_/Recv_) Count, (Send_/Recv_) Datatype, Tag, Dest/Root, Ierror, *Comm*

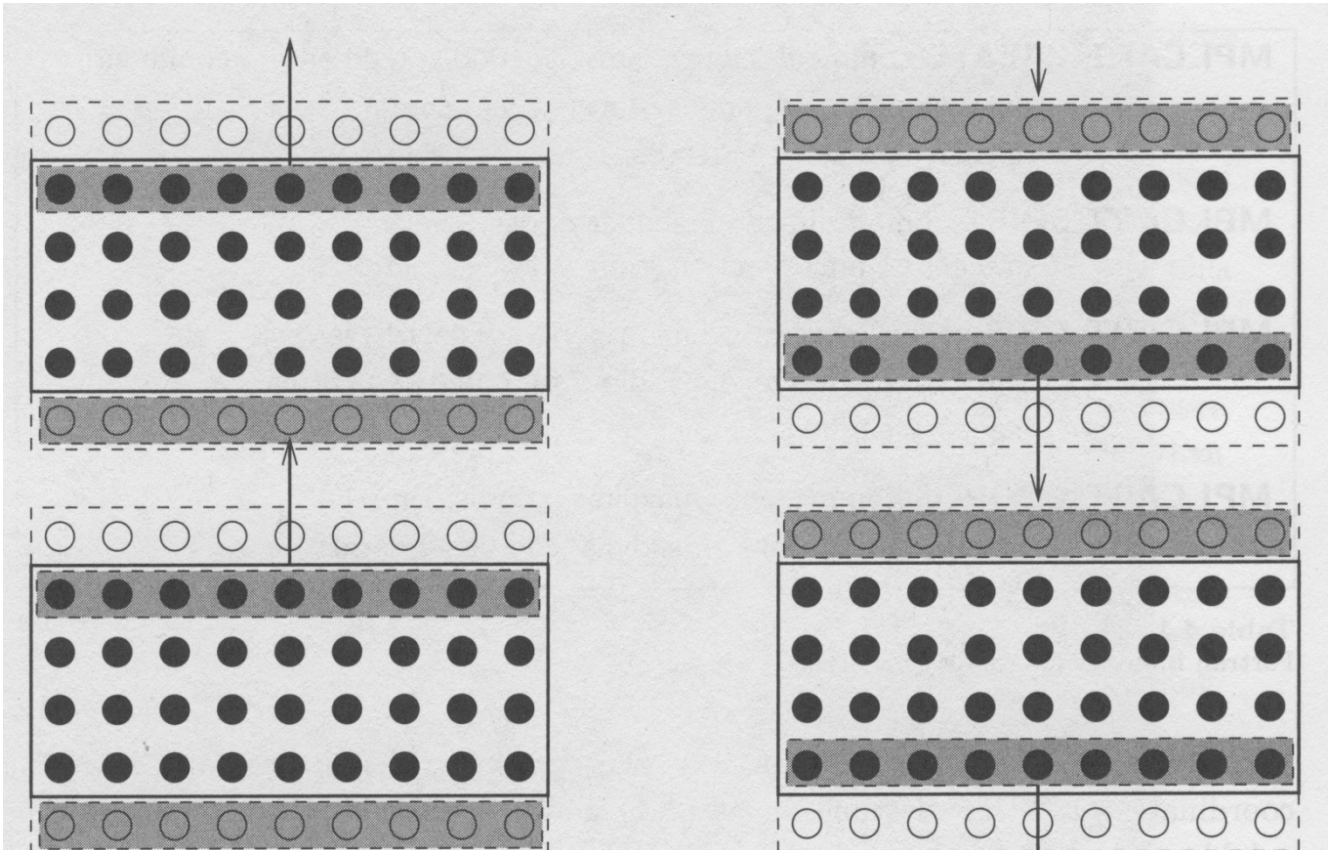
Parallele Bearbeitung einer Matrix I



Überlappung der Teilmatrizen der einzelnen Prozesse

Quelle:
Gropp, Lusk & Skjellum
Using MPI

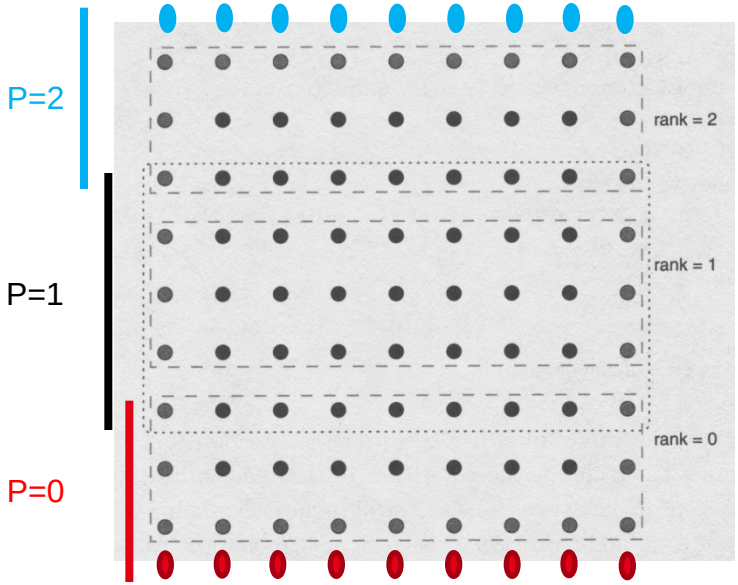
Parallele Bearbeitung einer Matrix II



Austausch
der Randstreifen
im Programmlauf

Quelle:
Gropp, Lusk & Skjellum
Using MPI

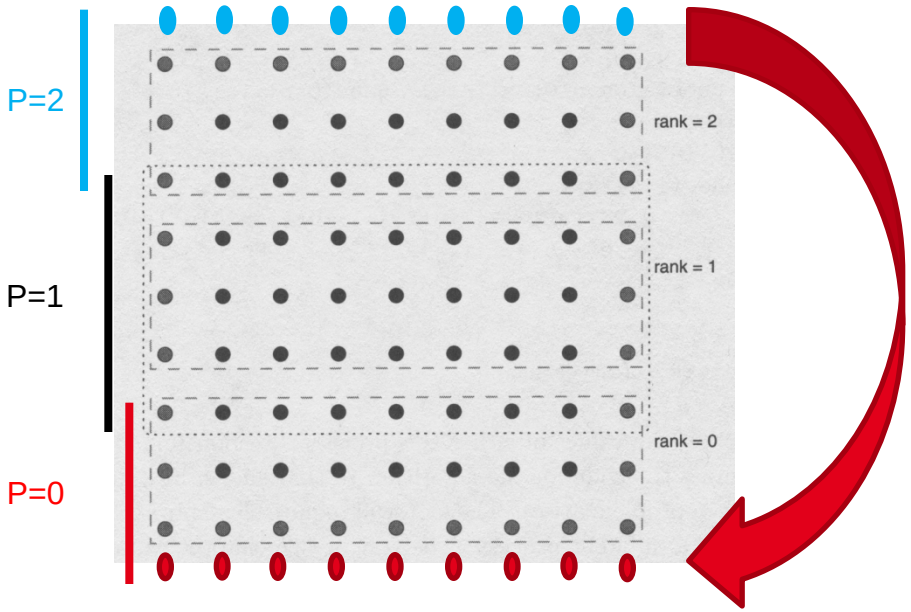
Parallele Bearbeitung einer Matrix III



Geänderte Matrix
für unsere Applikationen

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

MPI „Kommunikations Hirarchie II“

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)

Type (Send_/Recv_)Message (*)

Integer (Send_/Recv_) Count, (Send_/Recv_) Datatype, **Displacement**,
Root, Ierror, *Comm*

Beispiel Verwendung von Scatterv:

```
real*8, allocatable, dimension(:,:) :: data, partData
integer :: intervals, ierr, rank, numThreads
integer, dimension(:), allocatable :: sendCounts, recvCounts, displs, partRows


intervals=99
allocate(data(intervals+1,intervals+1)

Vorraussetzung:    mod(intervals+1,numThreads) == 0
partRows(:) = (intervals+1)/numThreads
sendCounts(1:numThreads) = (intervals+1)*partRows(1)
do i=1,numThreads
displs(i) = (i-1)*partRows(1)*(intervals+1)
end do

recvCounts=(intervals+1)*partRows(rank+1)
allocate(partData(intervals+1,partRows(rank+1)+2))

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

Beispiel Verwendung von Scatterv:

```
real*8, allocatable, dimension(:,:) :: data, partData
integer :: intervals, ierr, rank, numThreads
integer, dimension(:), allocatable :: sendCounts, recvCounts, displs, partRows

intervals=99
allocate(data(intervals+1,intervals+1)
```

Matrix data(100X100)

Vorraussetzung: mod(intervals+1,numThreads) == 0

```
partRows(:) = (intervals+1)/numThreads
sendCounts(1:numThreads) = (intervals+1)*partRows(1)
```

partRows(:) = =100/4 = 25

sendCounts(1:4)=100*25=2500

```
do i=1,numThreads
displs(i) = (i-1)*partRows(1)*(intervals+1)
end do
```

displ = /0, 2500,5000,7500/

```
recvCounts=(intervals+1)*partRows(rank+1)
allocate(partData(intervals+1,partRows(rank+1)+2))

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

Beispiel Verwendung von Scatterv:

```
real*8, allocatable, dimension(:,:) :: data, partData
integer :: intervals, ierr, rank, numThreads
integer, dimension(:), allocatable :: sendCounts, recvCounts, displs, partRows

intervals=99
allocate(data(intervals+1,intervals+1)                    Matrix data(100X100)

Vorraussetzung:   mod(intervals+1,numThreads) == 0
partRows(:) = (intervals+1)/numThreads                    partRows(:) =   =100/4 = 25
sendCounts(1:numThreads) = (intervals+1)*partRows(1)

                                                          sendCounts(1:4)=100*25=2500
do i=1,numThreads
displs(i) = (i-1)*partRows(1)*(intervals+1)               displ = /0, 2500,5000,7500/
end do

allocate(partData(intervals+1,partRows(rank+1)+2))        Matrix partData(100,27)
recvCounts=(intervals+1)*partRows(rank+1)                 recvCounts=100*25=2500

call MPI_SCATTERV(data,sendCounts,displs,MPI_DOUBLE_PRECISION,&
&partData(1,2), recvCounts,MPI_DOUBLE_PRECISION,
&master,MPI_COMM_WORLD,ierr)                              master=0
```

MPI Scatterv / Gaterv

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

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

Type (Send_/Recv_)Message (*)

Integer (Send_/Recv_) Count, (Send_/Recv_) Datatype, **Displacement**,
Root, Ierror, *Comm*

Danke das wars!