

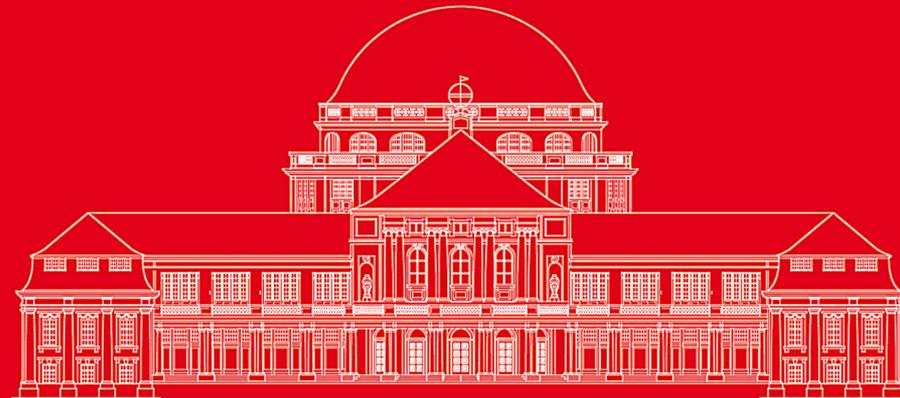


Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart & Michael Bleasel



Dr. Hermann-J. Lenhart

hermann.lenhart@informatik.uni-hamburg.de



MPI Einführung I:

- Einführung Nachrichtenaustausch mit MPI
- MPI point-to-point communication: Send & Receive
- MPI Barrier
- MPI kollektive Operationen: Broadcast & Reduce
- MPI Arbeitsumgebung definieren

MPI Umgebung - Start

Program hello

```
use mpi
```

```
INTEGER :: ierr
```

```
CALL MPI_INIT(ierr)
```

Spezielle Syntax bei FORTRAN

```
CALL MPI_FINALIZE(ierr)
```

Paralleler Bereich Beginn



Paralleler Bereich Ende

End

ABER: Alle Prozesse starten gleichzeitig!



MPI Umgebungsvariablen I

Für allgemeine Infos stehen folgende Befehle zur Verfügung um die MPI Umgebung zu erfragen.

MPI_Comm_size Wieviele Prozesse sind aktiv

MPI_Comm_rank Welchen Rang hat der aktuelle Prozess

MPI Umgebungsvariablen II

Program hello

```
use mpi
```

```
INTEGER :: ierr, rank, size
```

```
CALL MPI_INIT(ierr)
```

Paralleler Bereich Beginn

```
CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
```

```
CALL MPI_COMM_SIZE (MPI_COMM_WORLD, size,ierr)
```

```
Print*, ' I am ',rank,' of ',size
```

```
CALL MPI_FINALIZE(ierr)
```

Paralleler Bereich Ende



End

Die Nutzung von MPI Befehlen haben die Struktur von Funktionsaufrufen



MPI Steuerung mit Umgebungsvariablen I

Nutzung der Umgebungsvariablen zur Programmsteuerung:

```
integer :: myid, numproc, ierr
```

```
call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
```

```
call MPI_COMM_SIZE(MPI_COMM_WORLD, numproc, ierr)
```

Dazu wichtige Vorüberlegung, wie will ich die Anteile der Berechnung:

- Teilbereiche einer DO-Schleife
- Teilbereiche eines Vektors bzw. einer Matrix
- u.s.w.

=> auf die Prozesse aufteilen die mir zur Verfügung stehen?



MPI Steuerung mit Umgebungsvariablen II

Nutzung der Umgebungsvariablen zur Programmsteuerung:

```
integer :: myid, numproc, ierr, master
```

```
master=0
```

Nummerierung der Prozesse startet mit NULL! (Master)

```
call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
```

```
call MPI_COMM_SIZE(MPI_COMM_WORLD, numproc, ierr)
```

```
if (myid .ne. master) then
```

```
    call MPI_SEND(temp,1,MPI_DOUBLE_PRECISION,master,.....)
```

```
else
```

Prozesse senden an Master

```
do i=1,numproc-1
```

```
    call MPI_RECV(temp,1,MPI_DOUBLE_PRECISION,i,.....)
```

```
enddo
```

Master empfängt gesendete Nachricht von Prozessen

```
endif
```

MPI Umgebungsvariable III

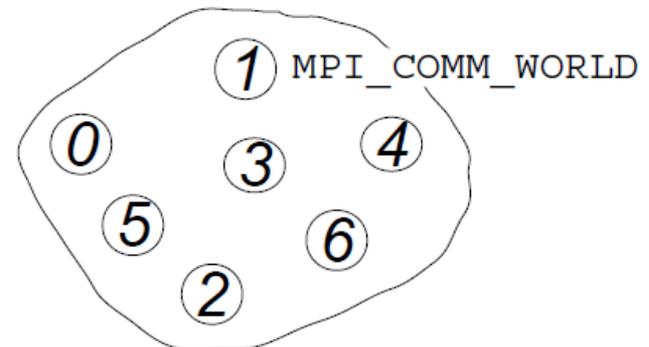
MPI_COMM_World Kommunikator (Gruppe, Kontext)

Der Kommunikator ist eine Variable welche eine Gruppe von Prozessen definiert die miteinander kommunizieren dürfen.

Es gibt einen default Kommunikator

MPI_COMM_WORLD

welcher die Gruppe aller vorhandenen Prozesse (hier 6) automatisch definiert.

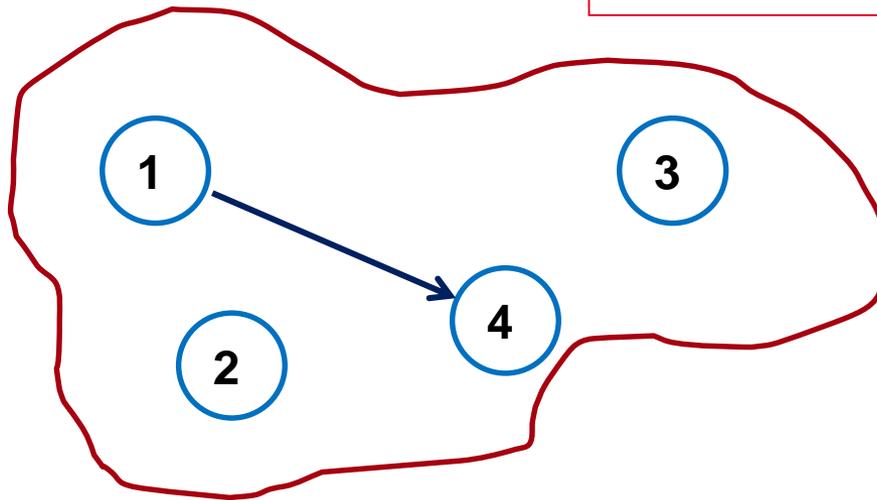


In einem Programm können mehrere Kommunikatoren gleichzeitig definiert werden.



MPI Point to Point Communication:

Kommunikator: `MPI_COMM_WORLD`



Send -> Receive



MPI Nachrichtenaustausch

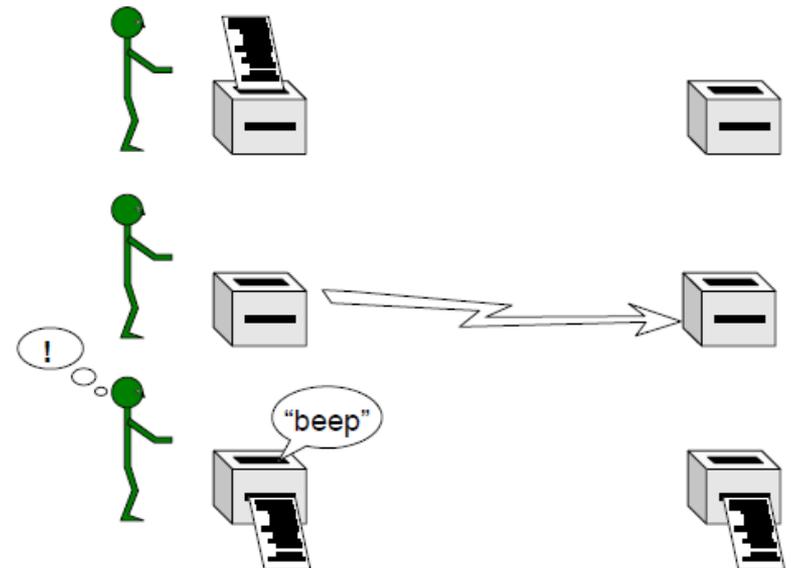
Der MPI Nachrichtenaustausch im Vergleich zum FAX

Für einfachste Art der MPI Kommunikation -
Point to Point Communication:

Ein Prozess sendet eine Nachricht
zu einem Anderen,

und erhält Rückmeldung über die
ordnungsgemäße Zustellung (!)

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





MPI Nachrichtenaustausch

Der Nachrichtenaustausch bedarf folgender Informationen:

- Sendender Prozess (Quelle)
- Empfangender Prozess (Ziel)
- Datentyp
- Datenlänge
- Status der Nachricht
- Nachrichtenumgebung (z.B. wieviele Prozesse sind vorhanden?)



MPI Send/Receive Syntax I

MPI_SEND(Message, Count, Datatype, Dest, Tag, Comm, lerror)

z.B:

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

temp	Adresse des Sendepuffers; Real :: temp
1	Count – Anzahl der Elemente im Puffer
MPI_Real	Datentyp des gesendeten Elementes
dest	Angabe des Ranges des Zielprozesses; integer :: dest
tag	Nachrichtenkennung - frei wählbar; integer :: tag
MPI_COMM_World	Kommunikator (Gruppe, Kontext)
lerror	Fehlerstatus; integer :: lerror



MPI Send/Receive Syntax II

MPI Datentypen in Anlehnung an Fortran

MPI Datentyp

FORTRAN Datentyp

MPI_INTEGER

INTEGER

MPI_REAL

REAL

MPI_DOUBLE_PRECISION

DOUBLE PRECISION

MPI_LOGICAL

LOGICAL

MPI_CHARACTER

CHARACTER(1)

Aber auch frei definierbar!



MPI Send/Receive Syntax II

Match zwischen Send und Receive:

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

MPI_RECV(Message, Count, Datatype, Source, Tag, Comm, status, Ierror)

Bzw:

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

Call MPI_RECV(temp, 1, MPI_Real, source, tag, MPI_COMM_World, status, Ierror)



MPI Send/Receive Syntax III

MPI_RECV(Message, Count, Datatype, Source, Tag, Comm, status, lerror)

Call MPI_RECV(temp, 1, MPI_Real, source, tag, MPI_COMM_World, status, lerror)

temp	Adresse des Sendepuffers; Real :: temp
1	Count – Anzahl der Elemente im Puffer
MPI_Real	Datentyp des gesendeten Elementes
source	Angabe des Ranges des Sendeprozesses; integer :: source
tag	Nachrichtenennung (Reihenfolge); integer :: tag
MPI_COMM_World	Kommunikator (Gruppe, Kontext)
status	Empfangsstatus der Nachricht (angekommen?); integer status(MPI_STATUS_SIZE)
lerror	Fehlerstatus; integer :: lerror



MPI Send/Receive Programmbeispiel

program main

use mpi

integer rank, ierr, **status(MPI_STATUS_SIZE)**

character(len=11) :: message

call MPI_INIT(ierr)

call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)

if (rank.eq.0) then

message = 'Hello from Master'

call MPI_SEND(message, 11, MPI_CHARACTER, **1**, 2017, MPI_COMM_WORLD, ierr)

=> senden an 1

endif



call MPI_RECV(message, 11, MPI_CHARACTER, **0**, 2017, MPI_COMM_WORLD, status, ierr)

=> empfangen nur auf 1 von 0

call mpi_finalize(ierr)

end program main



MPI Barrier I

Der MPI_BARRIER Befehl wird zur Programmsteuerung eingesetzt.

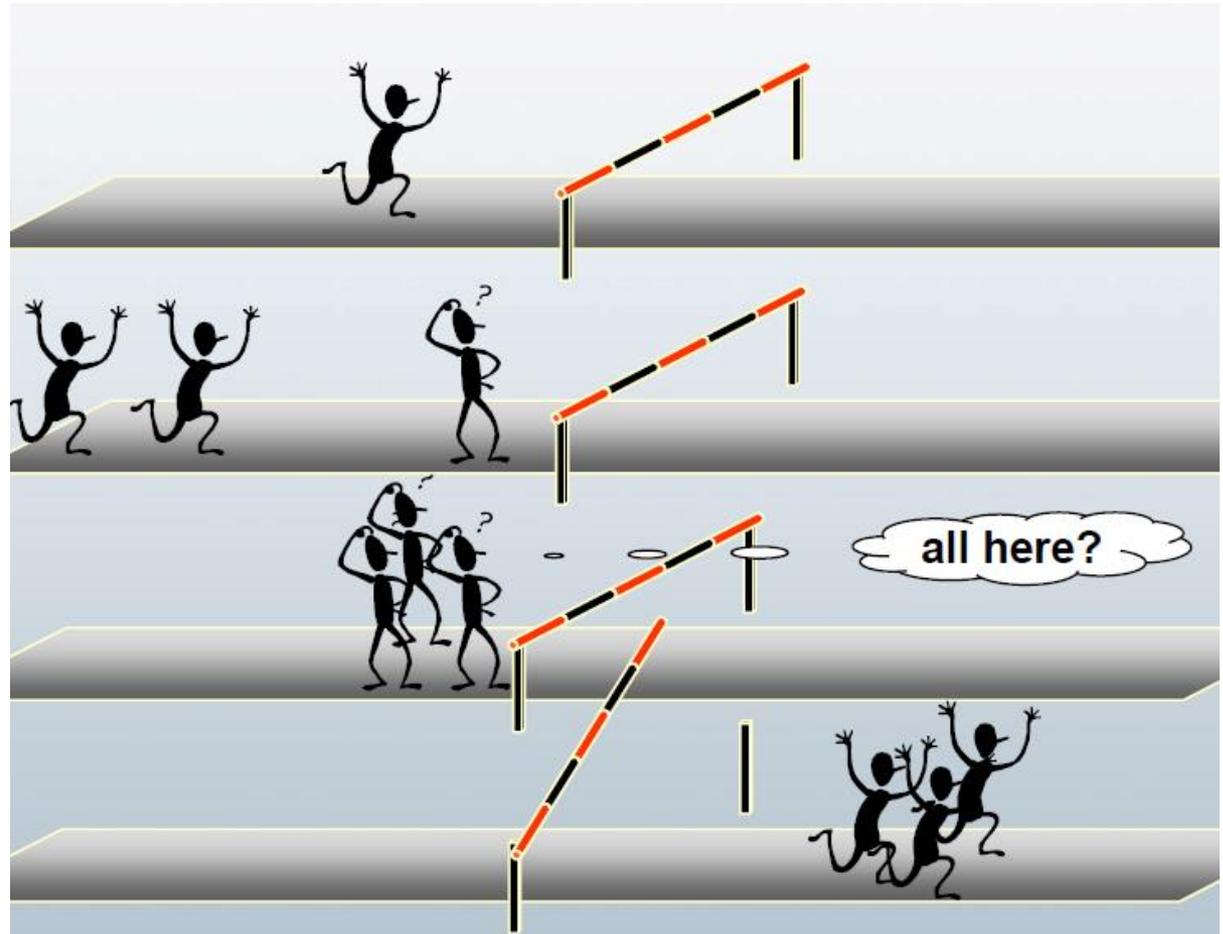
Call MPI_BARRIER(MPI_COMM_World, Ierror)

Der MPI_Barrier Befehl erzwingt dass alle Prozesse den gleichen Punkt im Code erreicht haben bevor das Programm weiterläuft.



MPI Barrier II

DKRZ MPI Einführungs Kurs





MPI Barrier III

Der MPI_BARRIER Befehl wird vorrangig zur Zeitmessung eingesetzt, z.B.

....

```
Double Precision :: t1, total_time
```

....

```
Call MPI_BARRIER(MPI_COMM_World, ierror)
```

```
t1 = MPI_WTIME()
```

....

```
Call MPI_BARRIER(MPI_COMM_World, ierror)
```

```
total_time = MPI_WTIME() - t1
```



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, lerror)`

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

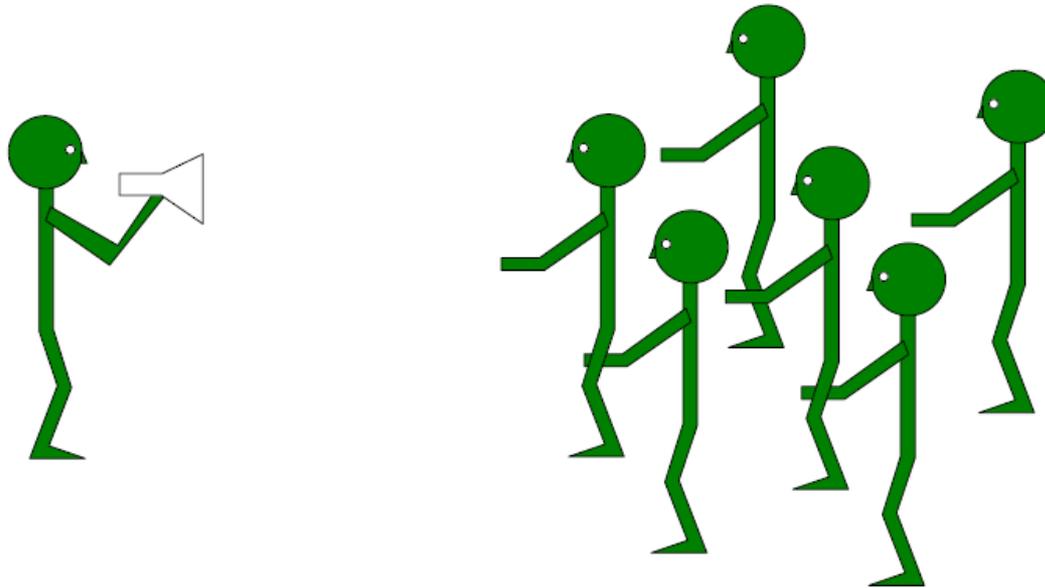
`MPI_BCAST(Message, Count, Datatype, Root, Comm, lerror)`

Call `MPI_BCAST(temp, 1, MPI_Real, source, MPI_COMM_World, lerror)`

Oft für Initialisierung oder zum Programmabbruch vom Master genutzt.



MPI Broadcast II

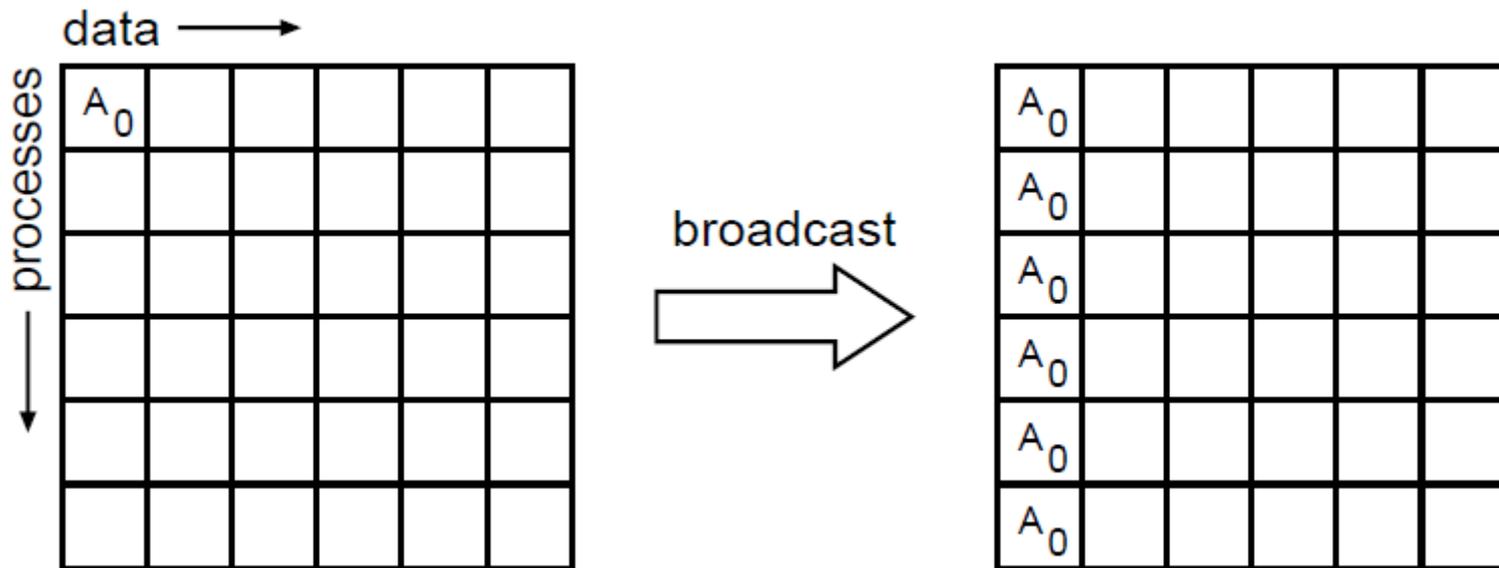


(Wolfgang Baumann ZIB, 2009;
Parallel Programming with MPI)



MPI Broadcast III

Verwendung zur Initialisierung



Es gibt eine interne Hierarchie der Zuteilung!

(William Gropp ANL, MPI Tutorial)



MPI Reduce I

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

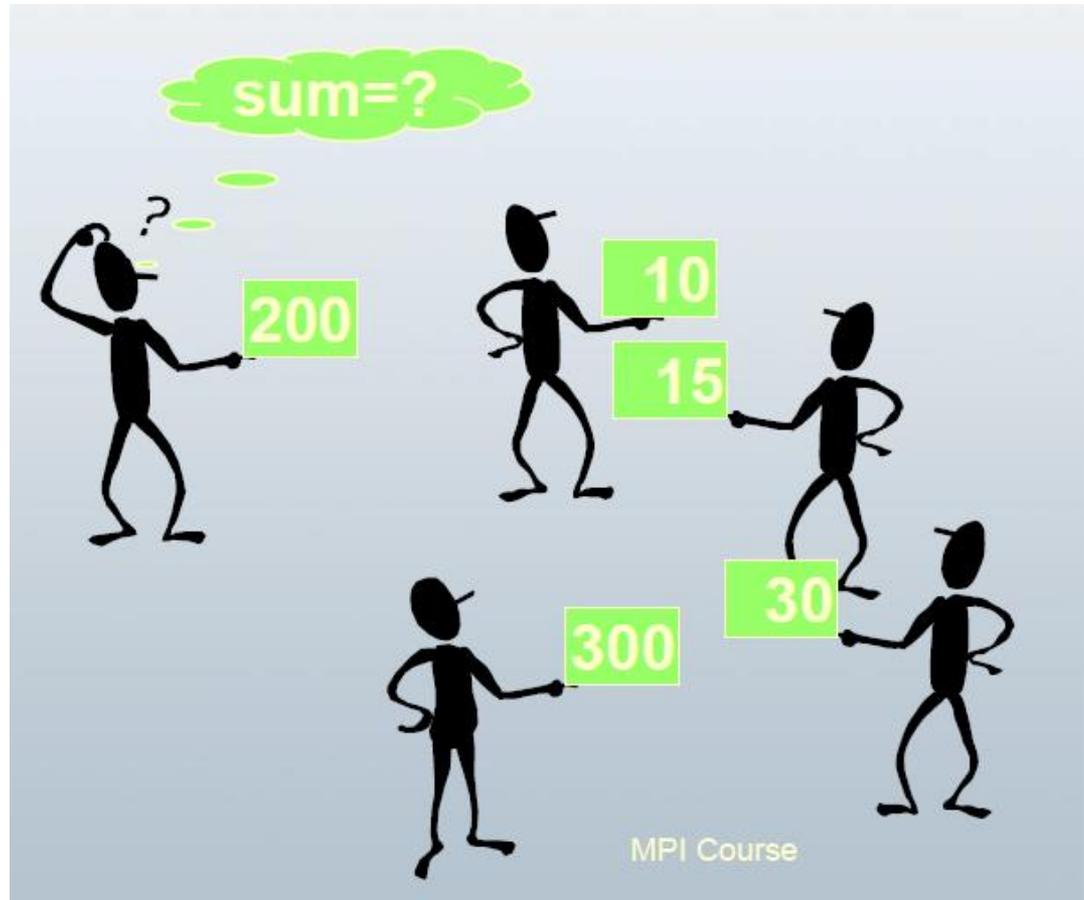
`MPI_REDUCE(Operand, Result, Count, Datatype, Operation, Root, Comm, lerror)`

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

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



MPI Reduce II

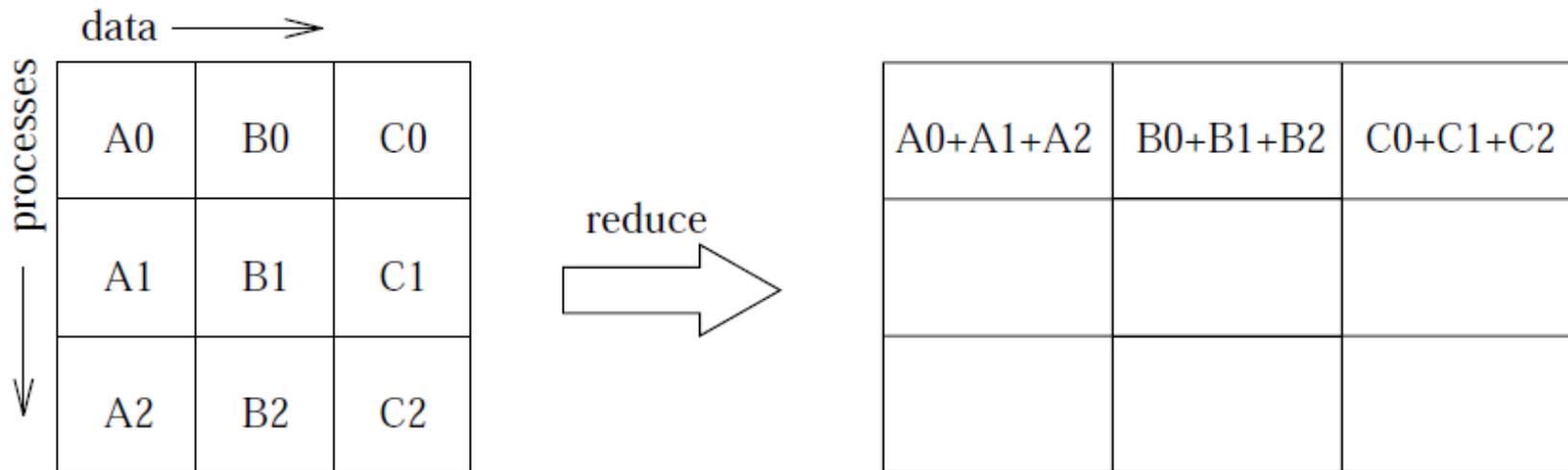


DKRZ MPI Einführungs Kurs



MPI Reduce III

Operator SUM



(William Gropp ANL,
MPI Tutorial)



MPI Reduce IV

Übersicht der möglichen MPI_Reduce Operationen:

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 Programmausführung

Makefile für die Ausführung von Program main.f90

Parallel auf 4 Prozessoren:

```
main.x:main.f90
    mpif90 -o main.x main.f90
```

```
run: main.x
    mpiexec -n 4 ./main.x
```



Danke,
gibt es noch Fragen?

Weitere Links auf Webseite !