



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart



Dr. Hermann-J. Lenhart

hermann.lenhart@zmaw.de



MPI Einführung I:

- Einführung Nachrichtenaustausch mit MPI
- MPI point-to-point communication
- MPI erste kollektive Befehle



Möglichkeiten der Parallelen Programmierung :

Generell abhängig von der Hardware:

- **OpenMP** - Möglich bei der Nutzung von gemeinsamem Speicher
(shared memory directives)

- **MPI** (Message-Passing Interface)
 - bei Rechnerarchitektur mit verteiltem Speicher
 - derzeit einziger Standard mit Portabilität auf allen Plattformen

- **Hybride** Programmierung: Kombination von MPI und OpenMP



MPI (Message Passing Interface) - Nachrichtenaustausch

MPI Nachrichten sind Datenpakete die zwischen Prozessen ausgetauscht werden.

Hardware Rahmenbedingungen:

- Keinen direkten Zugriff auf Memory (Daten) von anderen Prozessen.
- Datenverfügbarkeit **nur** über expliziten Datenaustausch (Senden/Empfangen) mit anderen Prozessen!

!! Vorteil: MPI Prozesse lassen sich skalieren !!



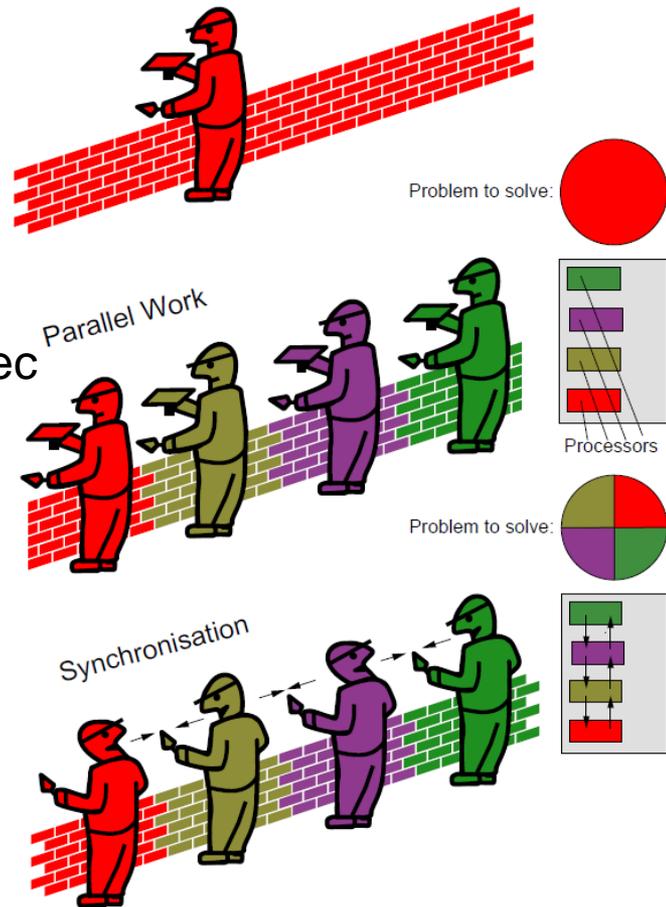
MPI Nachrichtenaustausch zu Bedenken !!

Als Info für das Verhältnis
Rechnen / Nachrichtenaustausch
 soll folgende Abschätzung dienen:

Ein moderner Parallelrechner schafft
 ca. 3 Mrd. floating point operationen / Sec

Der Nachrichtenaustausch
 aber nur 10 Mio. Wörter / Sec

Faktor 300 !!



MPI Umgebung - Start

Program hello

```
use mpi
```

```
INTEGER :: ierr
```

```
CALL MPI_INIT(ierr)
```

```
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

MPI Umgebungsvariable III

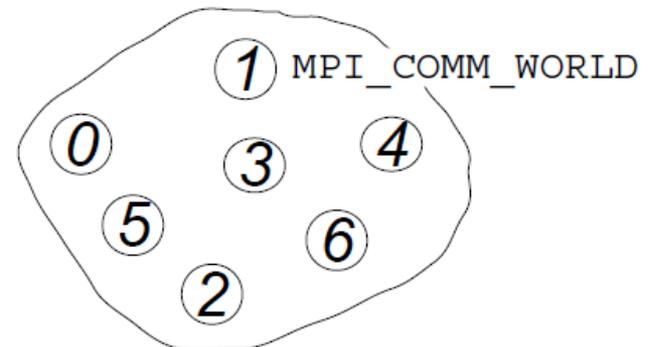
MPI_COMM_World Kommunikator (Gruppe, Kontext)

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 Umgebungsvariablen IIII

Nutzung der Umgebungsvariablen zur Programmsteuerung:

```
integer :: myid, proc, ierr, master
```

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

```
call MPI_COMM_SIZE(MPI_COMM_WORLD, proc, 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 Umgebungsvariablen III

Nutzung der Umgebungsvariablen zur Programmsteuerung:

```
integer :: myid, proc, ierr, master
```

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

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

```
if (myid .ne. 0) then
```

Nummerierung der Prozesse startet mit NULL! (Master)

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

```
else
```

```
    do i=1,proc-1
```

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

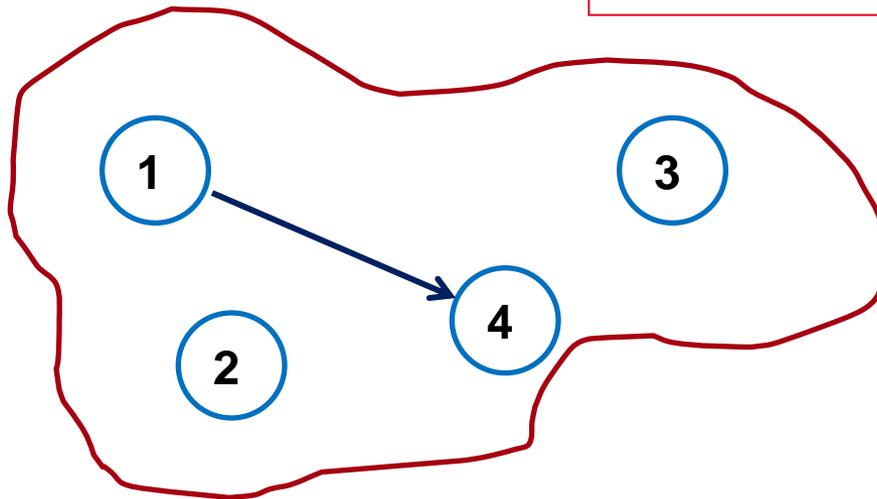
```
    enddo
```

```
endif
```



MPI Point to Point Communication:

Kommunikator: MPI_COMM_WORLD



Send -> Receive



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; 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)



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 Again'
```

```
endif
```

```
call MPI_SEND(message, 11, MPI_CHARACTER, 1, 2016, MPI_COMM_WORLD, ierr)
```



```
call MPI_RECV(message, 11, MPI_CHARACTER, 0, 2016, MPI_COMM_WORLD, status, ierr)
```

```
call mpi_finalize(ierr)
```

```
end program main
```



MPI Programmausführung I

Hinweis:

Auf dem cluster muss am Anfang das Kommando

`module load mpich2`

ausgeführt werden. Dann stehen auch die manpages für MPI zur Verfügung.

`man mpiexec`

`man mpif90`

Das Kommando `mpif90` verwendet intern einen Fortran Compiler, der bei der Kompilierung der MPI Bibliothek festgelegt wurde.

Der Aufruf ersetzt daher im Makefile das Kommando `f90` oder `gfortran`.



MPI Programmausführung II

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

clean: |
    rm *.x
```



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; Prod) 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:

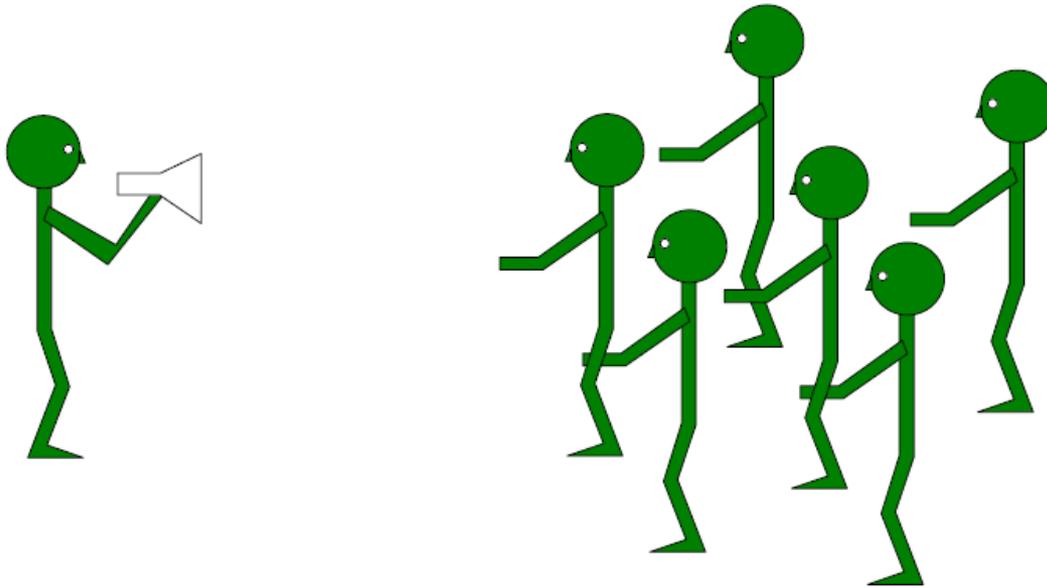
`MPI_BCAST(Message, Count, Datatype, Root, Comm, lerror)`

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

Für Initialisierung oder zum Programmabbruch genutzt.



MPI Broadcast II

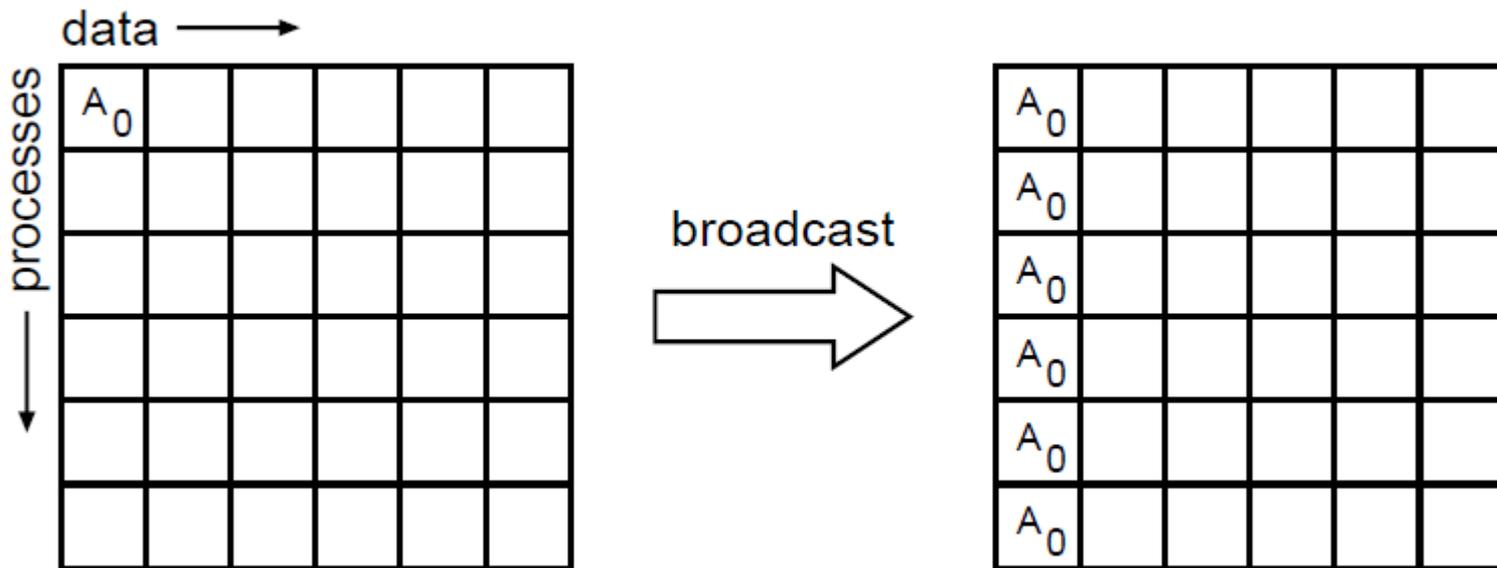


(Wolfgang Baumann ZIB, 2009;
Parallel Programming with MPI)



MPI Broadcast III

Verwendung zur Initialisierung



Es gibt eine interne Hierarchie der Zuteilung!

(William Gropp ANL, MPI Tutorial)



MPI Barrier I

Der MPI_BARRIER Befehl wird zur Programmsteuerung eingesetzt.

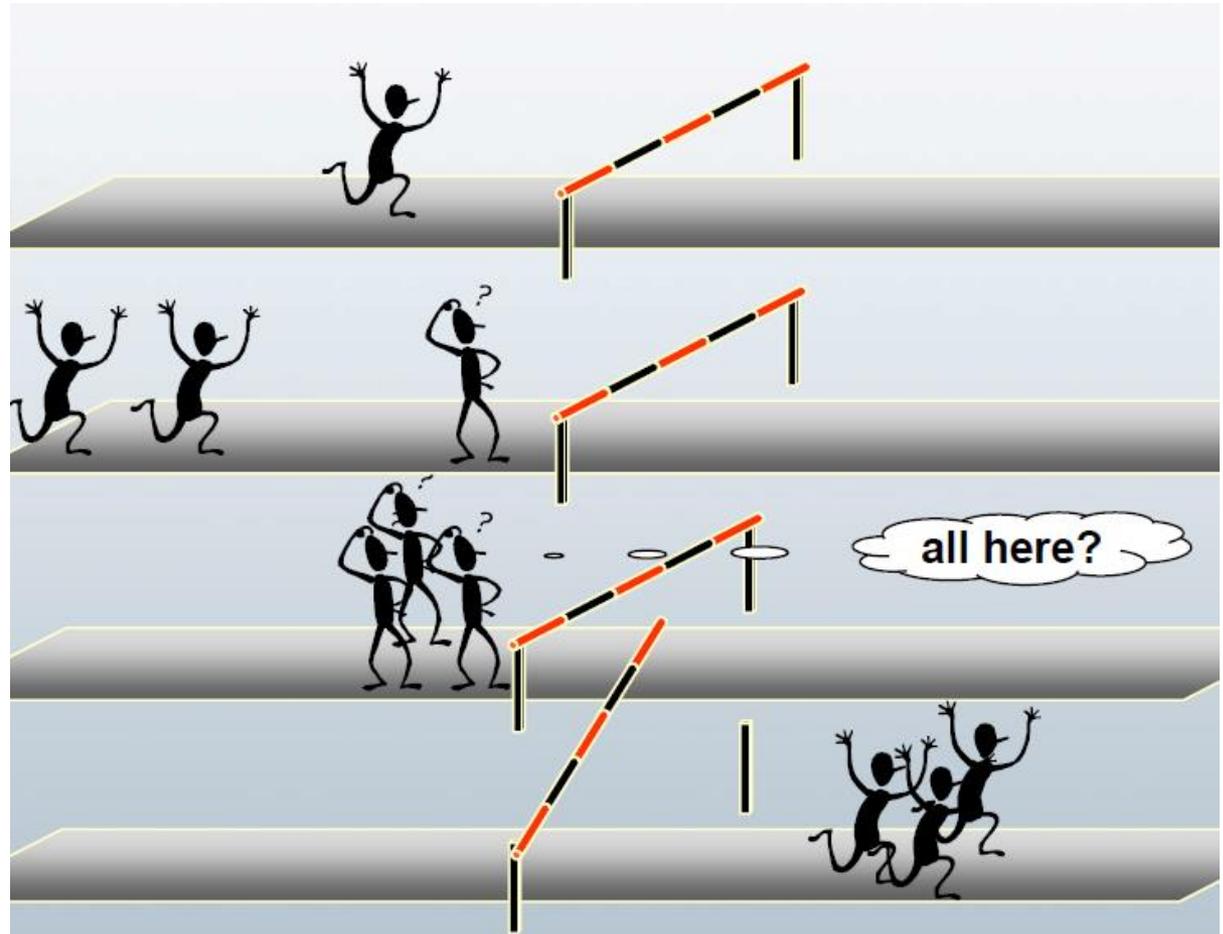
Call MPI_BARRIER(MPI_COMM_World, Ierror)

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.

....

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

```
t1 = MPI_WTIME()
```

....

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

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



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG



Danke,
gibt es noch Fragen?