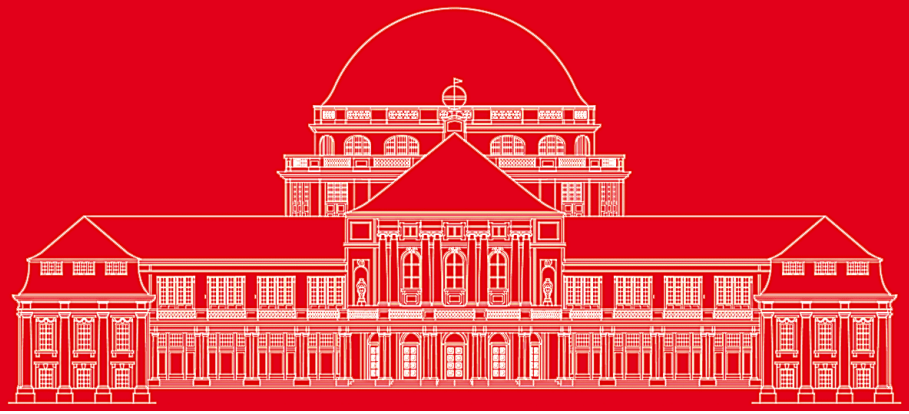
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart & Tim Jammer



Dr. Hermann-J. Lenhart
hermann.lenhart@zmaw.de

MPI Einführung I:

- Einführung Nachrichtenaustausch mit MPI
- MPI point-to-point communication: Send & Receive
- MPI Barrier
- MPI Arbeitsumgebung definieren

MPI Umgebung - Start

```
Program hello
    use mpi
    INTEGER :: ierr
    CALL MPI_INIT(ierr)

    CALL MPI_FINALIZE(ierr)
End
```

Paralleler Bereich Beginn

Paralleler Bereich Ende

ABER: Alle Prozesse starten gleichzeitg!

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

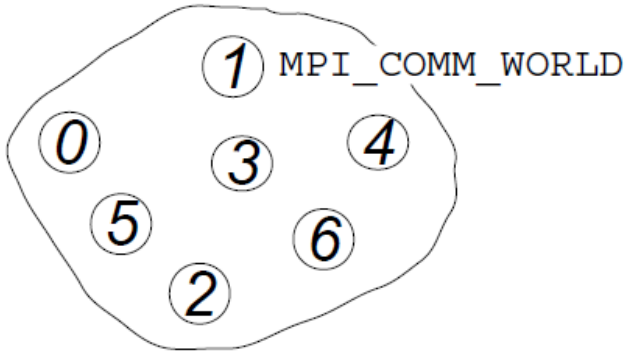
MPI Umgebungsvariablen II

```
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 IV

Nutzung der Umgebungsvariablen zur Programmsteuerung:

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

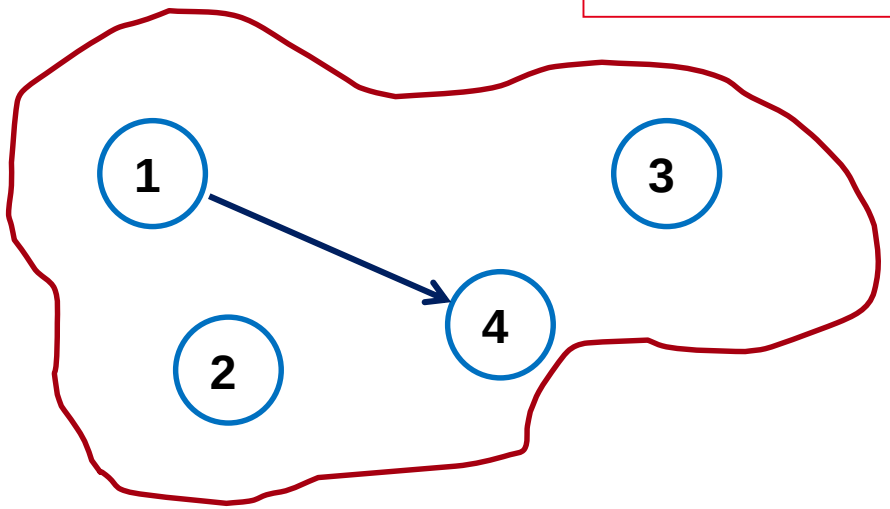
MPI Umgebungsvariablen III

Nutzung der Umgebungsvariablen zur Programmsteuerung:

```
integer :: myid, numproc, ierr, master
master=0                                   Nummerierung der Porzesse 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 Point to Point Communication:

Kommunikator: MPI_COMM_WORLD



Send -> Receive

MPI Nachrichtenaustausch

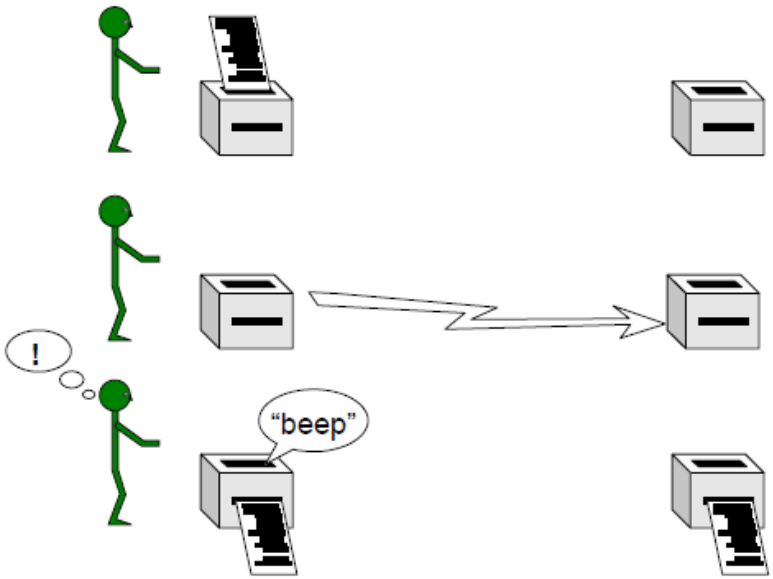
Der MPI Nachrichtenaustausch im Vergleich zum FAX

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

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

Ein Prozess sendet eine Nachricht zu einem Anderen,

und erhält Information über die ordnungsgemäße Zustellung (!)



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, Ierror)

z.B:

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

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)
Ierror	Fehlerstatus; integer :: Ierror

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, Ierror)

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

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	Nachrichtenkennung (Reihenfolge); integer :: tag
MPI_COMM_World	Kommunikator (Gruppe, Kontext)
status	Empfangsstaus der Nachricht (angekommen?); integer status(MPI_STATUS_SIZE)
Ierror	Fehlerstatus; integer :: Ierror

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'
                                                          => senden an 1
             call MPI_SEND(message, 11, MPI_CHARACTER, 1, 2017, MPI_COMM_WORLD, ierr)

              endif
                                                          ⇕
             call MPI_RECV(message, 11, MPI_CHARACTER, 0, 2017, MPI_COMM_WORLD, status, ierr)
                                                            => empfangen von 0
      call mpi_finalize(ierr)

 end program main
```

MPI Barrier I

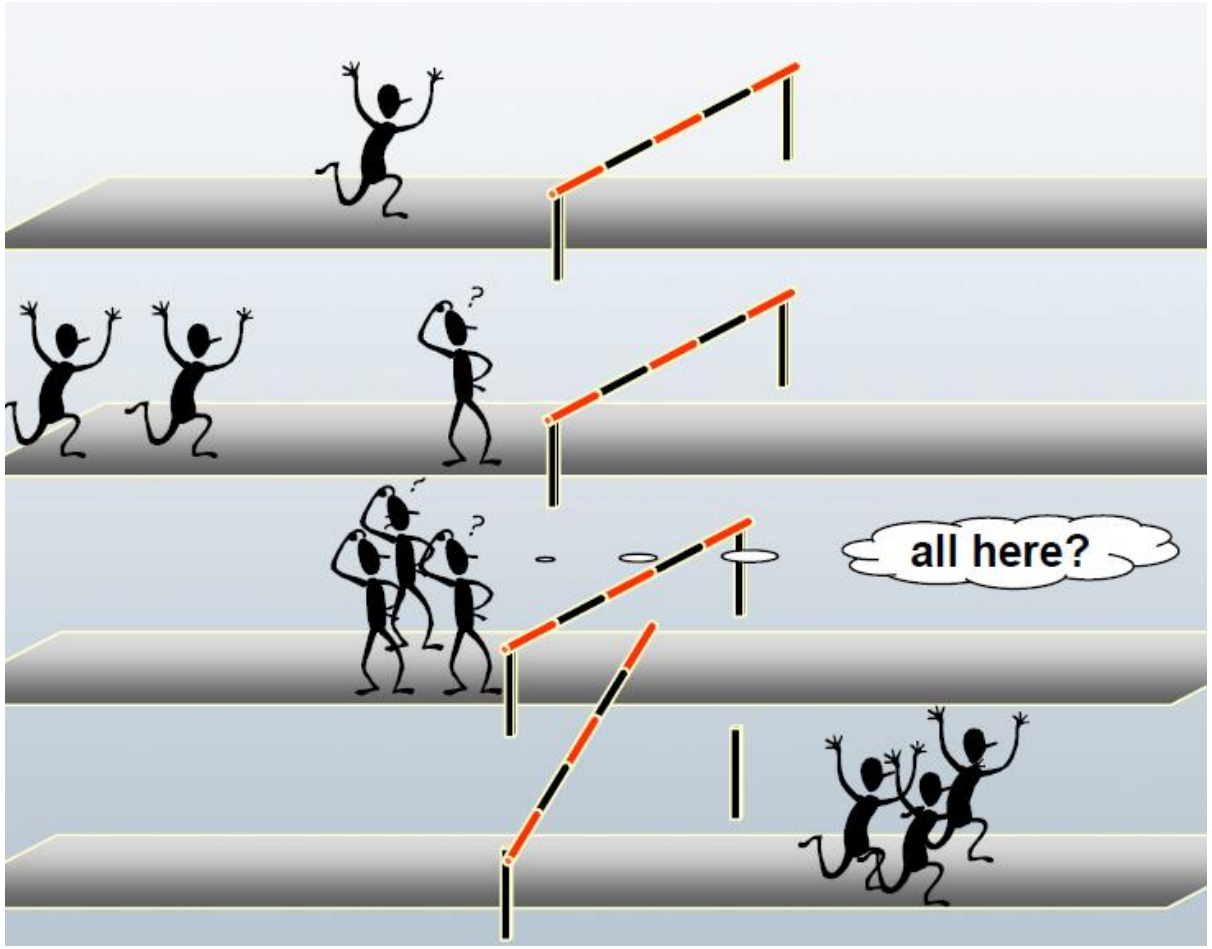
Der MPI_BARRIER Befehl wird zur Programmsteuerung eingesetzt.

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 vorranging 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
```

MPI Programmausführung I

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
```

MPI Programmausführung II

Auf dem Cluster müssen am Anfang folgende Kommandos

```
$ export MPICH_NEMESIS_NETMOD=tcp
$ spack load -r mvapich2
$
```

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.

Danke,
gibt es noch Fragen?