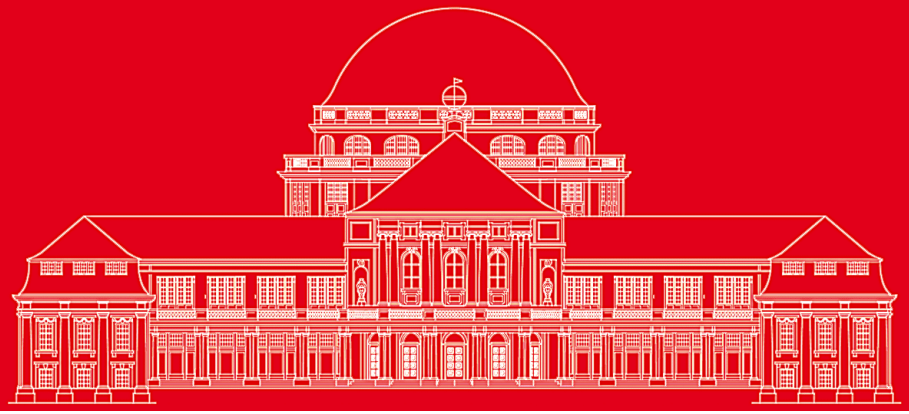
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart



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

MPI Einführung I:

- Hardware Voraussetzung zur Parallelen Programmierung
- MPI Nachrichtenaustausch
- Send/Receive Syntax
- MPI Umgebungsvariablen & praktischer Einsatz

Möglichkeiten der Parallelen Programmierung :

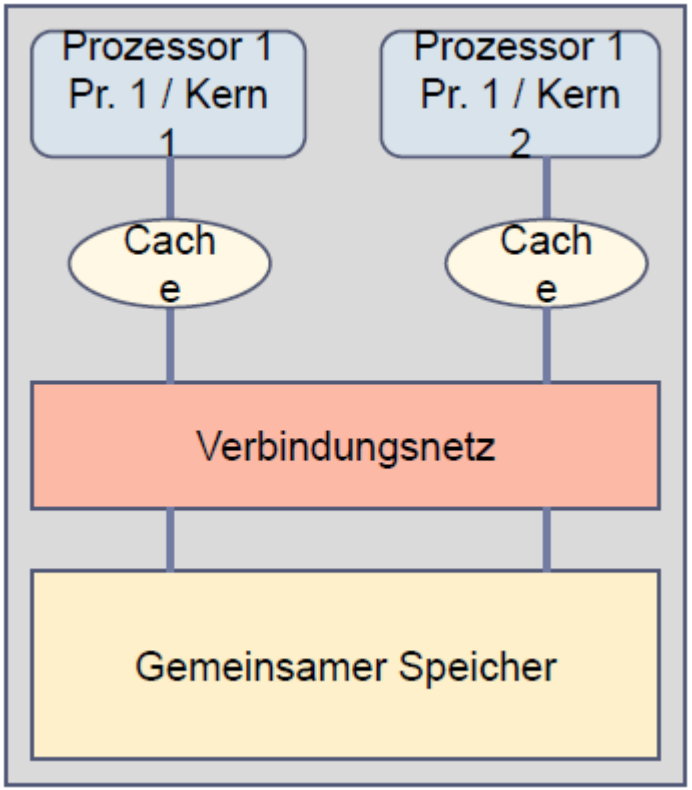
Generell abhängig von der Hardware:

- **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

OpenMP - Hardware Voraussetzung ist gemeinsamer Speicher



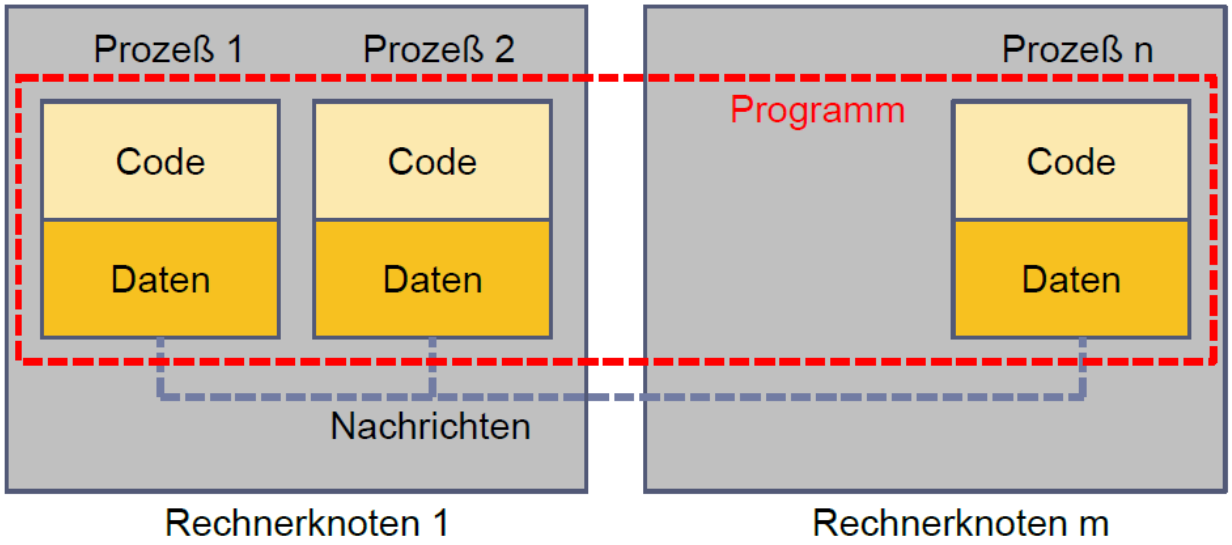
SMP:
Symmetrisches Multiprozessersystem
(symmetric multiprocessing)

Auf die Daten im gemeinsamem Speicher kann jederzeit von jedem Prozess aus zugegriffen werden.

(nach Ludwig WS12/13)

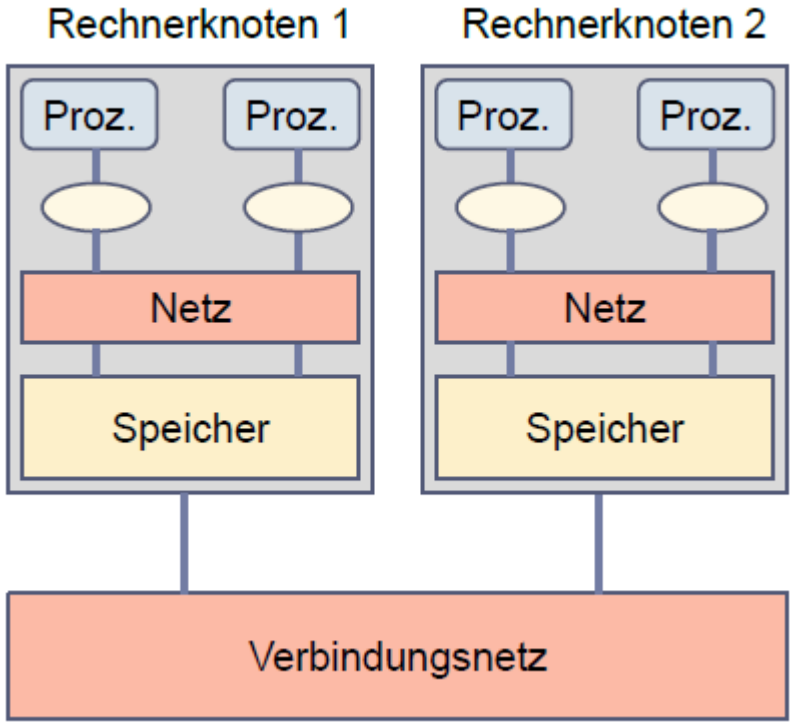
MPI – Hardware Vorraussetzung

(nach Ludwig WS12/13)



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

Hybride Programmierung



Existierende HLR sind heute meist
eine Kombination aus
Rechnerknoten mit gemeinsamem Speicher
von den man viele verwendet
und **über ein Verbindungsnetz** verbindet.

Ermöglicht die Kombination von
MPI und OpenMP

(nach Ludwig WS12/13)

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

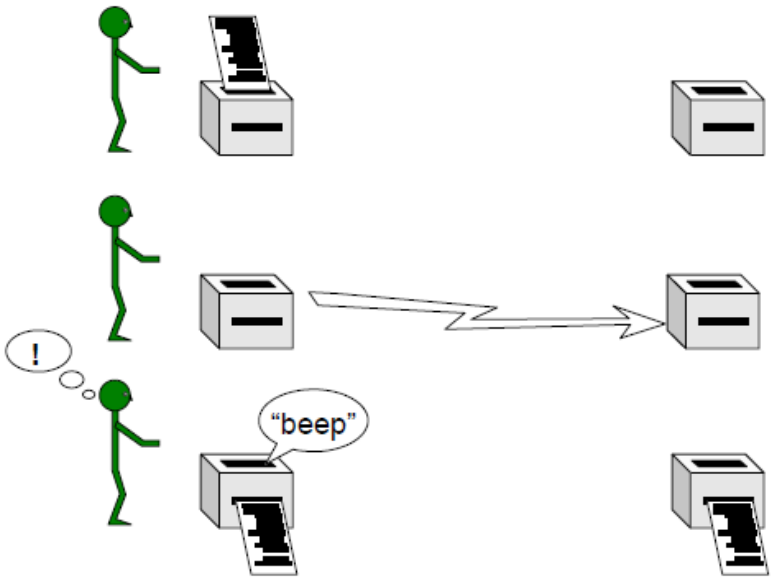
Der MPI Nachrichtenaustausch im Vergleich zum FAX

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
- Datentyp
- Datenlänge
- Empfangender Prozess
- 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:

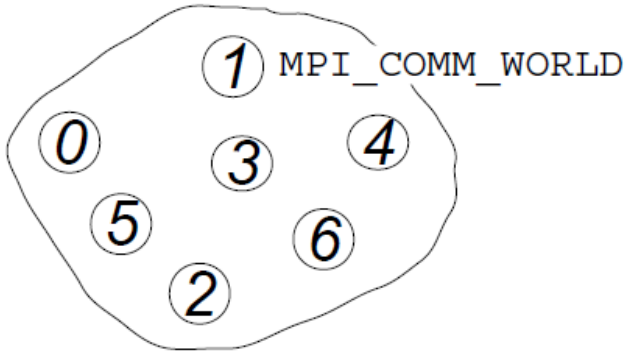
Call MPI_SEND(temp, 1, MPI_Real, dest, tag, MPI_COMM_World, Ierror)

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_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 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 BROADCAST

Neben dem Versenden von Nachrichten zwischen einzelnen Prozesse mittels

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

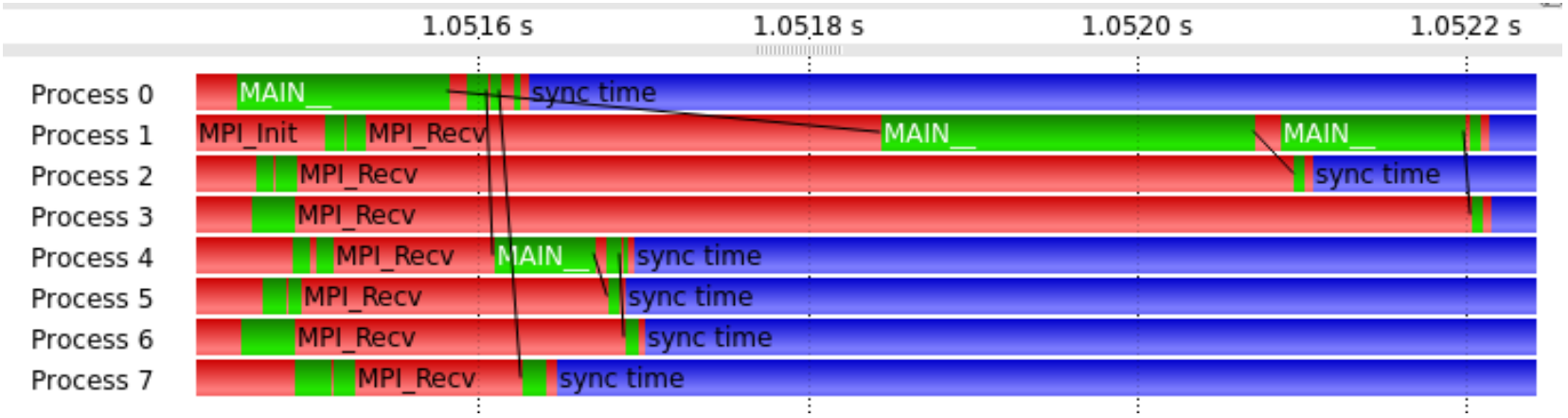
gibt es auch die Möglichkeit **eine Nachricht an alle anderen Prozesse** zu senden:

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

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

Für Initialisierung oder zum Programmabbruch genutzt.

Visualisierung des Programmablaufes mit Vampire:



MPI Umgebungsvariablen I

Für die „Bestückung“ der Send/Receive Aufrufe sowie 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
```

ABER: Alle Prozesse starten gleichzeitg!

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
        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
```

Nummerierung der Porzesse startet mit NULL! (Master)

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 Programmausführung

Die Ausführung von Program hello erfolgt

mpif90 -o hello hello.f90

mpiexec –n 4 ./hallo

Startet mit 4 Prozessen

Im Programm müssen vorhanden sein

```
use mpi
      CALL MPI_INIT(ierr)
          ..............
      CALL MPI_FINALIZE(ierr)
```

ierr gibt den Fehlerstatus an

integer :: ierr

MPI Programmausführung

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.

Danke,
gibt es noch Fragen?