Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart, Ulrich Körner, Nathanael Hübbe

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

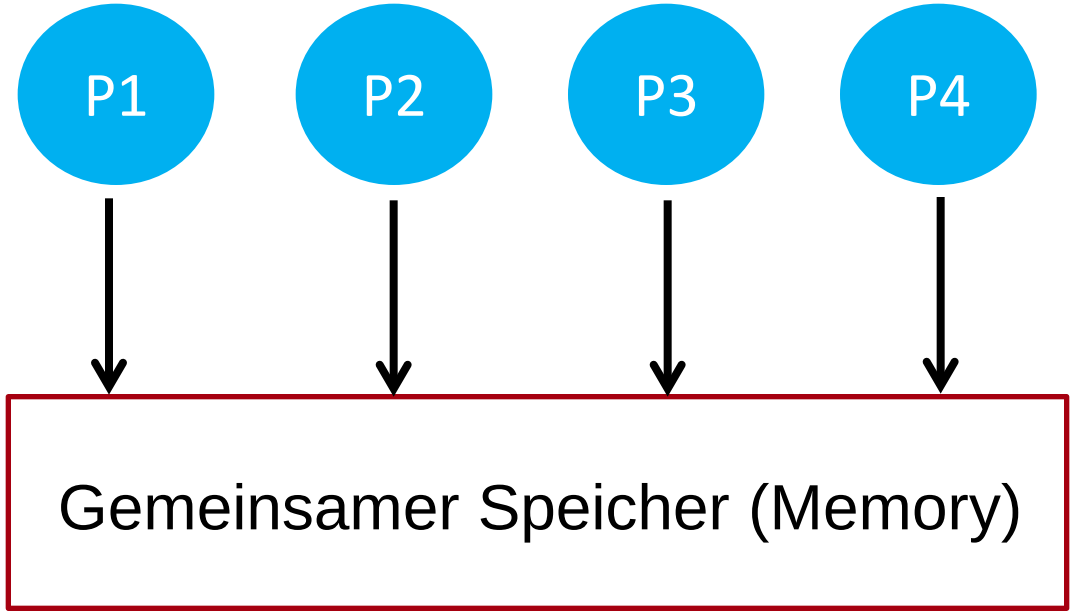
MPI Einführung I:

- Hardware Voraussetzung
- Nachrichtenaustausch
- Send/Receive Syntax
- MPI Umgebungsvariablen
- Broadcast
- Reduce Operation
- Gather Operation

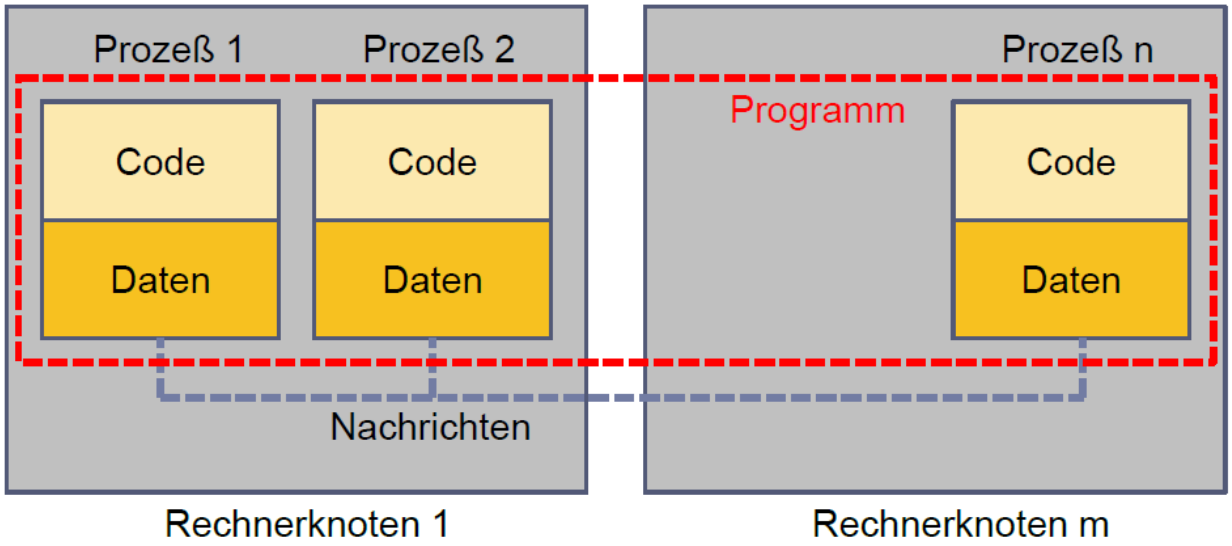
Hardware Voraussetzung

OpenMP Merkmal gemeinsamer Speicher:

SMP:
Symmetrisches Multiprozessersystem
(symmetric multiprocessing)



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!

MPI Nachrichtenaustausch

MPI Nachrichten sind Datenpakete die zwischen Prozessen ausgetauscht werden.

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 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 III

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

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

temp	Adresse des Sendepuffers; Real :: Vector
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 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

Alle Prozesse starten gleichzeitg!

```
Program hello
use mpi
INTEGER :: ierr, rank, size
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
CALL MPI_COMM_SIZE (MPI_COMM_WORLD, size,ierr)
Print*, ' I am ',rank,' of  ',size                    !    If (rank.eq. 0) Then ….
CALL MPI_FINALIZE(ierr)                               !       Ausführung für Thread 0
END                                                   !    Endif
```

MPI Umgebungsvariablen III

Bei einem MPI-Fehler bricht das MPI Programm als default alle Prozesse ab, ohne dass Fehlercodes ausgewertet werden müssen.

Beispiel für gezielt eingesetztes error handling:

```
CALL MPI_INIT(ierr)
if (ierr .ne. MPI_SUCCESS) then
      print *,'Error starting MPI program. Terminating.'
      call MPI_ABORT(MPI_COMM_WORLD, rc, ierr)
end if
```

MPI Programmausführung

Die Ausführung von Program hello erfolgt

```
mpif90  -o hello hello.f90
mpiexec   –n  4   ./hallo
```

Startet mit 4 Threads

Hinweis: $ module load mpich2

$ man mpiexec

Das Kommando module load stellt die MPI Umgebung ein.

Im Programm müssen vorhanden sein

```
INCLUDE 'mpif.h,
CALL MPI_INIT(ierr)
……
CALL MPI_FINALIZE(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 Kompiler, der bei der Kompilierung der MPI Bibliothek festgelegt wurde. Der Aufruf ersetzt daher im Makefile das Kommando f90 oder gfortran.

MPI Broadcast

Neben dem Versenden von Nachrichten zwischen einzelnen Threads mittels

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

gibt es auch die Möglichkeit **eine Nachricht an alle anderen Threads** 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.

MPI Reduce I

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

MPI_REDUCE(Operand, Result, Count,Datatype, Operation, Root, Comm, Ierror)

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

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

MPI Reduce II

Übersicht der möglichen MPI_Reduce Opertionen:

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 Gather I

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

Syntax: MPI_Gather(Sendmessage, Sendcount, Sendtype,
Recvmessage, Recvcount, Recvtype,
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)