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 III:

- Kommunikation
- Standard = blockierende Kommunikation
- Nicht blockierende Kommunikation
- MPI Barrier
- Übersicht MPI Kommunikation

MPI Kommunikation:

Das wichtigste Kriterium für die Entwicklung paralleler Programme besteht darin die Kommunikation effektiv zu gestalten.

Zur Information:

Ein moderner Parallelrechner kann
bis zu **500 Millionen floating-point Operationen pro Sekunde** berechnen,

aber nur etwa **10 Millionen Wörter pro Sekunde**
zwischen den Prozessen verschicken!

(Using MPI; Gropp,Lusk,Skjellm, 1999)

MPI Standard Send / Receive = Blocking Communication

MPI_SEND Programm läuft erst weiter nachdem
Send-Buffer zur Wiederverwendung gelehrt wurde

MPI_RECV Programm läuft erst weiter nachdem
Receive-Buffer gefüllt ist, d.h. die Daten stehen zur Verfügung

Die Ausführung der Kommunikation ist abhängig von der Größe der Nachricht
und der Größe des Systembuffers .

Blockierende Kommunikaton ist einfach einzusetzen,
aber anfällig für Deadlocks!

MPI Standard Send / Receive - Deadlock

```
If( rank == 0 ) Then
        Call MPI_send( buffer1, 1, MPI_integer, 1, 1, MPI_comm_world, error )
        Call MPI_recv( buffer2, 1, MPI_integer, 1, 2, MPI_comm_world, status,error )
Else If( rank == 1 ) Then
        Call MPI_send( buffer2, 1, MPI_integer, 0, 2, MPI_comm_world, error )
        Call MPI_recv( buffer1, 1, MPI_integer, 0, 1, MPI_comm_world, status, error )
End If
```

Das Programm produziert einen Deadlock!

MPI Standard Send - Tag I

Beim Standard send ist zu beachten:

Der Standard Send Befehl ist beendet sobald die Message verschickt wurde;
unabhängig davon ob die Message schon beim Empfänger angekommen ist,
sie kann immer noch für einige Zeit im Netzwerk liegen.

Tags erlauben dem Programmierer die Handhabung der einkommenden Messages in einer geordneten Weise zu vollziehen. MPI-Tags haben Rage von 0 bis 32767.

MPI_ANY_TAG kann als „Wildcard“ eingesetzt werden.

MPI Standard Send – Tag II

```
Integer::  recvd_tag, recvd_from

call MPI_RECV(..., MPI_ANY_SOURCE, MPI_ANY_TAG, .. status, ierr)

tag_recvd = status(MPI_TAG)
recvd_from = status(MPI_SOURCE)
```

MPI nicht blockierende Kommunikation

Eine weitere Option bietet die nicht-blockierende Kommunikation
(non-blocking communication)

Zwischen dem Senden (nach Füllen des Buffers) kann dann weiter gerechnet werden
(Umgehen der Latenz in der Kommunikation)

Unabhängig vom Zeitpunkt des Füllens des Buffers entscheidet der Programmier
wann die Nachricht mit dem receive Befehl empfangen wird.

Nachteil: Programmierer muss Abfragen einfügen um die Abwicklung zu checken.

MPI non-blocking Send

MPI_ISEND(Message, Count, Datatype, Dest, Tag, Comm, request, Ierror)

z.B:

Call MPI_ISEND(temp, 1, MPI_Real, dest, tag, MPI_COMM_World, req, 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)
request	Handle; integer :: req , Ierror

MPI non-blocking Receive

MPI_IRECV(Message, Count, Datatype, Source, Tag, Comm, request, Ierror)

z.B:

Call MPI_IRECV(temp, 1, MPI_Real, source, tag, MPI_COMM_World, req, 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 Sourceprozesses; integer :: source
tag	Nachrichtenkennung; integer :: tag
MPI_COMM_World	Kommunikator (Gruppe, Kontext)
request	Handle; integer :: req **!!! Kein Status** , ierror

MPI non-blocking Kommunikation Statusabfrage

Zur Abfrage des **Status der Isend Nachricht** , gibt es folgende Option:

MPI_WAIT(request,status, Ierror)

Integer :: request, status(MPI_STATUS_SIZE), ierror

CALL MPI_ISEND(buffer, count, datatype, dest, tag, comm, request, ierr)

….. Der Prozess rechnet weiter

Call WAIT (request, status, ierr)

↓ Geht weiter wenn die ISEND-Nachricht angekommen ist
und der Buffer wieder frei ist und neu belegt werden kann.

MPI Barrier I

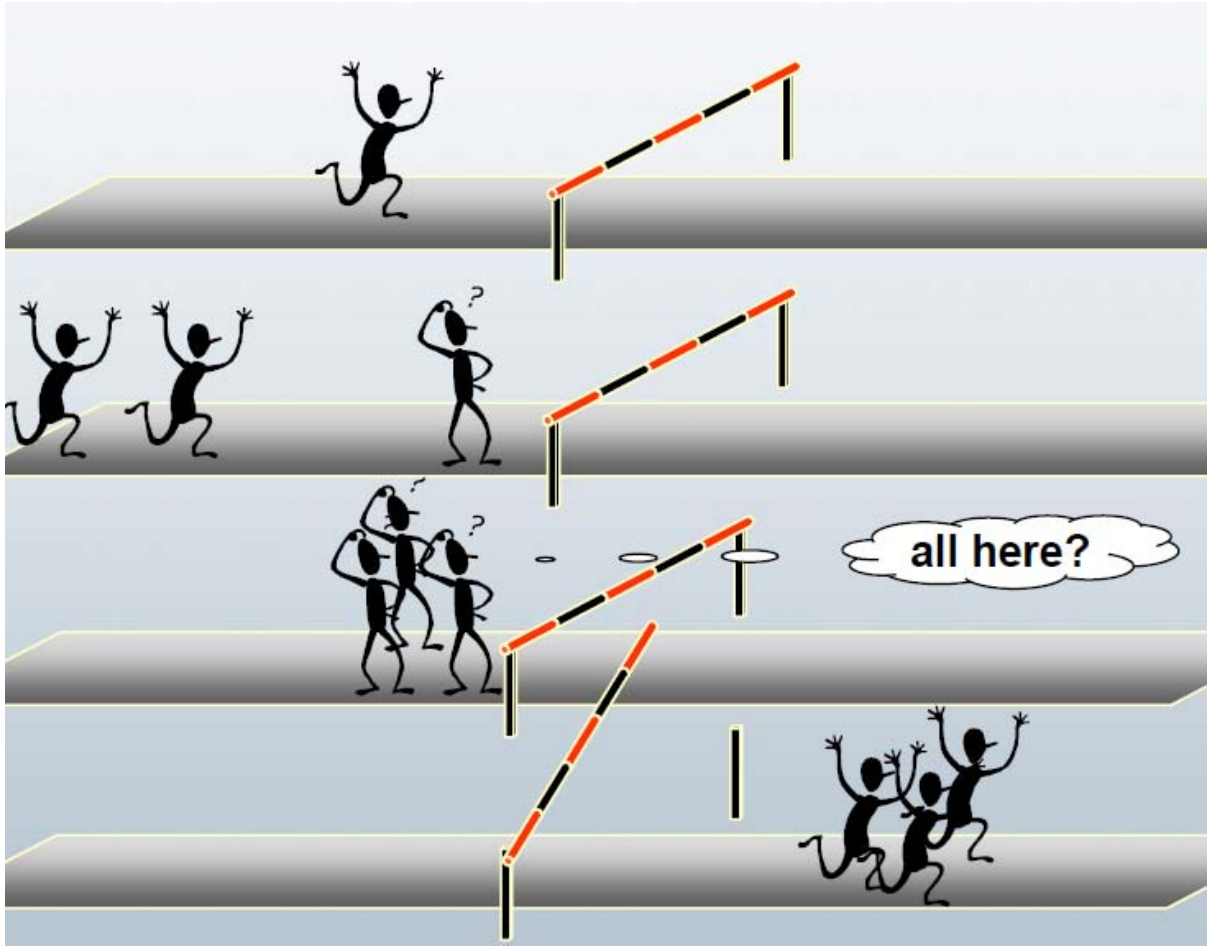
Der MPI_BARRIER Befehl wird zur Programmsteuerung eingesetzt.

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
```

Weitere Möglichkeiten an MPI Send/Receive Befehlen:

Standard Send	MPI_SEND	sent will not complete until send buffer is empty
Synchronous Send	MPI_SSEND	send does not complete until a matching recieve has been posted.
Buffered Send	MPI_BSEND	creates a buffer,copies data and returns control.
Ready Send	MPI_RSEND	always completes
Empfangen	MPI_RECV	

Stefano Cozzini, INFM

Übersicht der Kommunikations-Arten

Stefano Cozzini, INFM

Mode	Completion Condition	Blocking subroutine	Non-blocking subroutine
Standard send	**Message sent (receive state unknown)**	**MPI_SEND**	**MPI_ISEND**
receive	**Completes when a message has arrived**	**MPI_RECV**	**MPI_IRECV**
Synchronous send	**Only completes when the receive has completed**	MPI_SSEND	MPI_ISSEND
Buffered send	**Always completes, irrespective of receiver**	MPI_BSEND	MPI_IBSEND
Ready send	**Always completes, irrespective of whether the receive has completed**	MPI_RSEND	MPI_IRSEND

Danke das wars!

Dr. Hermann-J. Lenhart
hermann.lenhart@informatik.uni-hamburg.de