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

FORTRAN Einführung II:

- Programmstrukturen
- Subroutinen und Funktionen
- Module
- Felder
- Pointer
- Ausgabe
- Dateien Ein- und Ausgabe

FORTRAN Programm-Struktur (I)

Ein **FORTRAN Programm** kann aufgebaut werden aus den Komponenten:

- Hauptprogramm
- Externe Subprogramme (Funktionen und Subroutinen)
- Module

FORTRAN Programm-Struktur (II)

Ein **FORTRAN Haupt-Programm** besteht aus:

[program Programm – Name]	program test
FORTRAN Ausdruck	print*, 'Hello World'
End [Programm – Name]	end program test

FORTRAN Programm-Struktur (III)

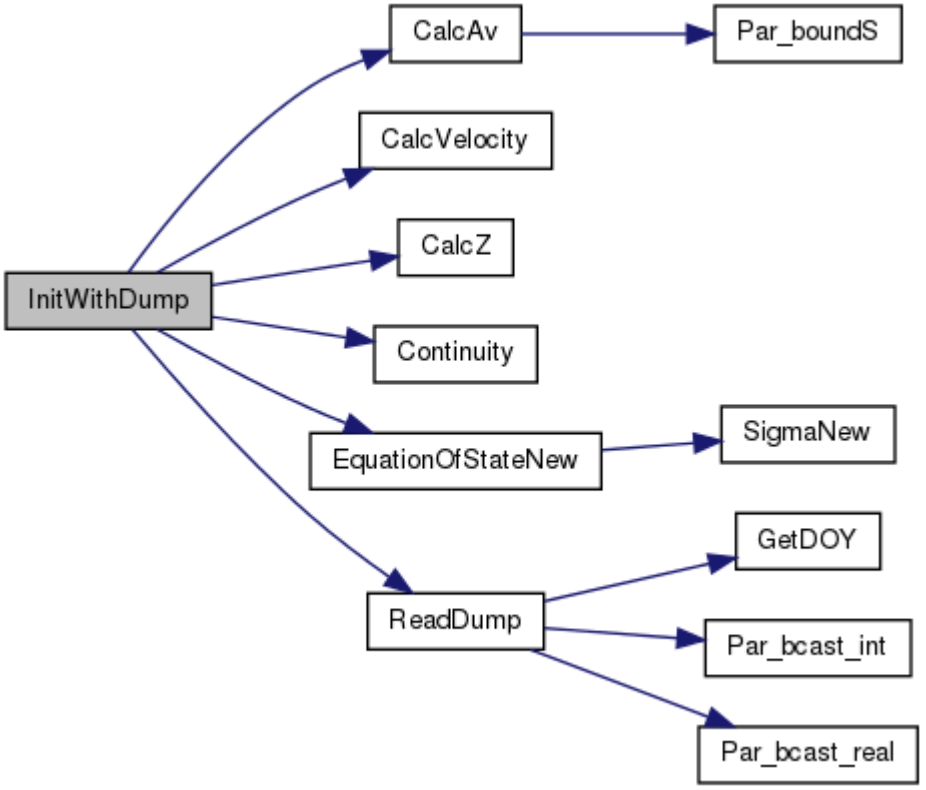
Ein gutes **FORTRAN Programm** übergibt wohl definierte Arbeitsaufgaben in Subroutinen oder Funktionen, die dann vom Hauptprogramm aufgerufen werden.

```
program  model
      call initial
      call calculate
      call write_output
end program model
```

In komplexeren Programmen können Subroutinen selber weitere Subroutinen und Funktionen aufrufen

FORTRAN Programm-Struktur (IV)

Beispiel für Verzweigung von Subroutinen aus dem Dokumentationstool **doxygen**.



FORTRAN Subroutine (I)

Eine **FORTRAN Subroutine** übernimmt eine definierte Arbeitsaufgabe.
Die in der Argumentliste übergebenen Argumente werden
innerhalb der Prozedur der Subroutine bearbeitet.
Vom Hauptprogramm wird die Subroutine über einen CALL Aufruf angesprochen.
Die Argumente werden entsprechend ihrer Reihenfolge übergeben.

```
program  subtest
  real :: x, y, z
        call multi (x, y, z)
end program subtest
```

```
subroutine multi (X,Y,Z)
  real :: X, Y,  Z
          Z = X * Y
end subroutine multi
```

FORTRAN Subroutine (II)

Eine **FORTRAN Subroutine** bildet eine eigenständige Einheit,
in der die genutzten Variablen eigenständig geführt werden.
Die inhaltliche Logik muss nur der Argumentenliste im CALL Aufruf entsprechen.

```
program  subtest2
implicit none
real :: u, v, w, x, y,  z
     call multi (x, y, z)
     call multi (u,v, w)
end program subtest2
```

```
subroutine multi (l, m, n)
 implicit none
    real :: l, m, n
         l = m * n
end subroutine multi
```

FORTRAN Subroutine (III)

Innerhalb einer **FORTRAN Subroutine** können die genutzten Variablen entsprechend ihrer Verwendung als Eingabe- oder Ausgabewert definiert werden. Hierzu dient die **intent** Angabe in der Typendeklaration:

```
program  subtest2
implicit none
real :: u, v, w, x, y,  z
    call multi (x, y, z)
   call multi (u,v, w)
end program subtest2
```

```
subroutine multi (l, m, n)
implicit none
real, intent (in)    :: m, n
real, intent(out)  :: l
        l = m * n
 end subroutine multi
```

FORTRAN Funktion

Eine **FORTRAN Funktion** ist eine Prozedur
die über den Funktionsnamen einen Wert zurück liefert.

```
program  ftest
real :: x, y, z
      z = betr (x, y)
end program ftest
```

```
real fuction betr (x, y)
real :: x, y
      betr = sqrt (x*x + y*y)
end
```

FORTRAN intrinsische Funktionen (I)

Eine Reihe von Prozeduren sind als **intrinsische Funktionen** vordefiniert:

ABS	Absolutwert
SIN, COS	Sinus bzw. Cosinuswert
SQRT, EXP, LOG	Quadratwurzel, Exponent oder Logarithmus
MIN, MAX	Minimum bzw. Maximum
MOD	Modulo Funktion (Divisionsrest)
SIGN	Vorzeichen
INT	Umwandlung Real -> Integer
FLOAT	Umwandlung Integer -> Real

FORTRAN intrinsische Funktionen (II)

Argumentenabhängige intrinsische Funktionen.

Einige intrinsische Funktionen haben zusätzlich zu den generischen auch spezifische Namen. Das Resultat der Funktion kann dann einen von den Argumenten abweichenden Typ haben.

z.B. für die MAX-Funktion:

Spezifischer Name	Argument Typ	Ergebnis Typ
AMAX0	Integer	Real
MAX1	Real	Integer

(Bei Bedarf nachschlagen und im Allgemeinen, wegen der besseren Lesbarkeit, die generischen Namen verwenden)

FORTRAN intrinsische Funktionen (III)

FORTRAN bietet noch **weitere vordefinierte Prozeduren** für unterschiedliche Anwendungen:

Spezifischer Name	#Parameter	Ergebnis
date_and_time	1 […4]	liefert Datum und/oder Zeit
mvbits	5	überträgt Bits zwischen Integer Objekten
random_number	1	Liefert eine Zufallszahl
Random_seed	0[,1]	Initialisiert und startet Zufallszahlengenerator
System_clock	1[,2,3]	Liefert Info der „real time clock“ des Rechners

Beispiel Aufruf Sysemzeit:

```
INTEGER :: count, count_rate, count_max

CALL SYSTEM_CLOCK(count, count_rate, count_max)
```

FORTRAN Module

Ein FORTRAN Modul ist eine nicht ausführbare Programmeinheit zur Definition von Spezifikationen und zugehörigen Programmteilen. Das Modul wird ähnlich einer Subroutine beschrieben und mittels USE Statement im Hauptprogramm oder anderen Programmteilen eingebunden. Das ermöglicht z.B., auf include statements zu verzichten.

Ein Modul kann auch Subroutinen und Funktionen enthalten!

```
program  model
Use model_structure
   call initial
   call calculate
   call write-output
end program model
```

```
module model_structure
    ! Struktur der 3D Modellfelder
     real (kind=8, dimension 82,88,24) :: u,v,w
end module model_structure
```

FORTRAN Felder (I)

Zur **Belegung von FORTRAN Felder** mit Konstanten als Wertezuweisung gibt es mehrere Möglichkeiten:

```
program matrix_structure
Real, dimension (2:9) :: X = 1.
Real, dimension (1:5) :: u, v, y

    u = (/1., 2., 3., 4., 5./)
    v = (/ Real(i), i=1,5) /)
…
end program matrix_structure
```

Mann kann auch direkt die Matrixwerte nutzen

```
y = ( /  u(1:3,5), v(2) / )
```

FORTRAN Felder (II)

Mit **FORTRAN Feldern** können sogenannte Array Operationen duchgeführt werden, die für das gesamte Feld elementweise aufgeführt werden.

Voraussetzung ist allerdings, dass die Felder in der Operation die gleiche Größe haben.

```
Real, dimension (20,30) : : u, v, w, x, y, z
x = 0.0          Gesamtes Feld wird mit Nullen belegt (z.B. zur Initialisierung)
w = u + v        Elemente weise Zuweisung der Addition von u und v auf w
y = x  * v       Elemente weise Zuweisung der Multiplikation von x und v auf y
z = 7.5 * y      Elemente weise Zuweisung der Multiplikation von y
                                    mit  einem Skalar auf z
```

FORTRAN Felder (III)

FORTRAN verfügt weiterhin über eine Reihe von **Array Prozeduren** die auf Felder angewendet werden können. Hier eine kurze Auswahl:

SUM	gibt Summe aller Feldelemente
MAXVAL, MINVAL	gibt Maximum bzw. Minimum aller Feldelemente
MAXLOC, MINLOC	gibt Index des Maximal- bzw. Minimalwertes an
DOT_PRODUCT	gibt Skalarprodukt zweier ein-dimensionaler Vektoren aus
MATMUL	gibt das Ergebnis einer Matrizen-Multiplikation aus
SIZE	weist die Gesamtanzahl der Arrayelemente aus
SHAPE	weist die Gestalt eines Arrays aus
RESHAPE	erzeugt neues Array mit unterschiedlicher Gestalt

FORTRAN Felder (IV)

Array Prozeduren einzelne Beispiel **:**

```
real,  dimension (M) : : x, y
real   rsum1, rsum2, rprod
Integer maxloc
```

rsum1 = SUM (x)	gibt Summe des Arrays (hier Vektor) X zurück
rsum2 = SUM(x(2:M-1))	gibt Summe eines Teils (2:M-1) des Arrays X zurück
rprod = PRODUCT(x)	gibt Produkte der Array-Elemente zurück
locmax = MAXLOC(x, DIM=1)	ermittelt die Position des Maximum im Array x

FORTRAN Felder (V)

Array Prozeduren am Beispiel von **RESHAPE:**

RESHAPE erzeugt neues Array mit unterschiedlicher Gestalt.

Die Anordnung der Elemente im Speicher bleibt erhalten!

```
Integer, dimension (6) :: m1 =(/ (k,k=1,6)/)
Integer, dimension (2,3) :: m2
```

m1 entspricht Vektor [1,2,3,4,5,6]

```
..
! Reshape in Zielmatrix
m2 = reshape (m1, (/2,3/) )
```

m2 entspricht Matrix

1	2
3	4
5	6

FORTRAN Pointer (I)

Wird ein Object genutzt um auf andere Objekte zu referenzieren,
so muss dies mit einem **Pointer Attribut** deklariert werden.

Datentyp, pointer :: Variable

Real, pointer :: zeiger

Ein Pointer beinhaltet selber keine Date, sonder zeigt auf die Skalar-Variable
oder die Array Variable in der die Date gespeichert ist.

FORTRAN Pointer (II)

Ein so deklarierter Zeiger kann auf eine mit „target“ definierte Variable zeigen:

```
program pointer1
implicit none
real, pointer :: zeiger
real, target   :: ziel
      ziel = 7.7
      zeiger => ziel
      write(*,*) zeiger                    Ausgabe entspricht 7,7000
end program pointer1
```

FORTRAN Pointer (III)

siehe http://de.wikibooks.org/wiki/Fortran:_Fortran_95:_Zeiger

Für einen Pointer gibt es keine initiale Speicherbelegung,
sondern der Speicherbereich für den Pointer wird beim Programmlauf bereitgestellt.

Dies beinhaltet, das im Default-Zustand Pointer undefiniert (dangling) sind,
und es gibt keinen Weg dies zu testen.

Dieser Zustand sollte durch folgende Deklaration vermieden werden:
real, pointer :: zeiger => null() Zustand nicht zugeordnet (disassociated)

Nach Zuordnung zeiger => ziel ist der Zustand zugeordnet (associated)

FORTRAN Ausgabe (I)

Die Ausgabe aus Programmen heraus erfolgt über **Print** oder **Write:**

Die einfachste Möglichkeit zur Ausgabe bietet das Print Statement.
Daher wird Print als „Universal Debugger“ verwendet:
Der * kann auch durch ein Format Statement ersetzt werden.

```
print*,  i, j, x
```

besser mit Bezeichnung der ausgegebenen Größen

```
print*,  'Zeile= ',i,' Spalte= ',j,' Wert : ', x
```

FORTRAN Ausgabe (II)

Strukturierte Ausgaben sind über die Befehle **Print** und **Write** möglich, dabei muss auf den Datentyp geachtet werden**.** Hier am Beispiel des Write Befehls:

zuerst für **Character**:

```
Character (len=20) :: text
    text = ' Anything goes'
      write (*,'(a)') text                  frei definiert als Character

      write (*,'(a20)') text                dezidierte Anweisung der Länge der Character Elemente
```

Der * steht hier ebenfalls für eine Ausgabe auf Standardout.

FORTRAN Ausgabe (II)

Definierte Ausgabe über den **Write(oder Print) Befehl,** für die Datentypen **Integer und Real** stehen eine Reihe von Ausgabeanweisungen zur Verfügung:

iint = 12345

write (*,'(i5)') iint	passt genau, aber reicht z.B. nicht mehr für Vorzeichen

Rreal= $12345.e^{-8}$

write (*,'(f15.7)') rreal	verschenkt Ausgabe an Genauigkeit
write (*,'(f12.9)') rreal	ausreichende Genauigkeit beim Format vorhanden
write (*,'(e15.7)') rreal	Exponentendarstellung mit ausreichender Genauigkeit

FORTRAN Ausgabe (III)

Beispiel **Write Befehl,** für gemischte Ausgabe von Integer, Real und Character:

```
iint = (/1, 2, 3, 4, 5/)
rreal= 12345.e-8
text = 'Lückefüller'

write (*,'(5(I1,1x),F15.7)') iint, rreal                    x=Leerzeichen

write (*,'(5(I1,1x),A13,F15.7)') iint, text, rreal          oder alternativ

      write (*,500) iint, text, rreal
500 format (5(I1,1x),A13,F15.7)
```

FORTRAN Dateiausgabe (I)

Der **Write Befehl** wird genutzt um **Inhalte in eine Datei zu schreiben**.

Dies geht nicht mit dem Print Befehl!

Dazu muss die Datei vorher geöffnet werden, die Zuweisung (Unit) muss dann im Write Statement entsprechend vorgenommen werden. Abschließend wird die Datei geschlossen.

iint = (/1, 2, 3, 4, 5/)

rreal= $12345.e^{-8}$

```
open (unit = 11, File='Testdata.dat‘)
        write (11,'(5(I1,1x),F15.7)') iint, rreal
close (unit = 11)
```

FORTRAN Dateiausgabe (II)

Mit dem **OPEN Befehl** kann neben der Dateizuordnung auch deren Struktur definiert werden:

```
open (unit = 11, File='Testdata.dat', status = 'new')    Neue Datei um Daten zu schreiben
open (unit = 11, File='Testdata.dat', status = 'old')    Bestehende  Datei öffnen (lesen)

open (unit = 11, File='Testdata.dat', status = 'new', access= 'sequential')  sequentieller Zugriff
open (unit = 11, File='Testdata.dat', status = 'new', access='direct' , recl=40)  direkter Zugriff
    und einer Rekordlänge von z.B. 40 Byte (Angabe der Recordnummer im Read statement!)

open (unit = 11, File='Testdata.dat', status = 'new', access= 'formatted')     formatierte Textdatei
open (unit = 11, File='Testdata.dat', status = 'new', access= 'unformatted')  unformatierte Datei
                                                                              = Binärdatei
```

FORTRAN Lesen aus Datei

Der **READ Befehl** wird genutzt um **Inhalte aus einer Datei zu lesen**.

Wie beim Write Befehl muss die Datei vorher geöffnet werden, die Zuordnung der Unit muss dann im Read Statement entsprechend vorgenommen werden. Abschließend wird die Datei geschlossen. Die Unit Nummern 0, 5 und 6 sind im Allgemeinen fest vorbelegt. Daher ist es üblich eine zweistellige Unitnummer zu vergeben.

```
integer, dimension (5) : :iint
real :: rreal

   open (unit = 11, File='Testdata.dat',  status = 'old')
            read (11, '(5(I1,1x), F15.7)') iint, rreal
   close (unit = 11)
```

Schönen Dank !