



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](mailto:hermann.lenhart@zmaw.de)



## OpenMP Einführung I:

- Allgemeine Einführung
- Prozesse und Threads
- Fork-Join Programmier Model
- Aufruf und Notation (Einführungsbeispiel)
- Syntax
- Bestimmung der Thread Anzahl
- Thread Abfrage



# OpenMP – Allgemeine Einführung I

## OpenMP Merkmale:

OpenMP ist keine Programmiersprache!

OpenMP Notationen werden zu einem sequentiellen FORTRAN Programmen hinzugefügt um anzugeben wie die Arbeit auf die Prozesse verteilt wird und wie der gemeinsame Speicherzugriff erfolgen soll.

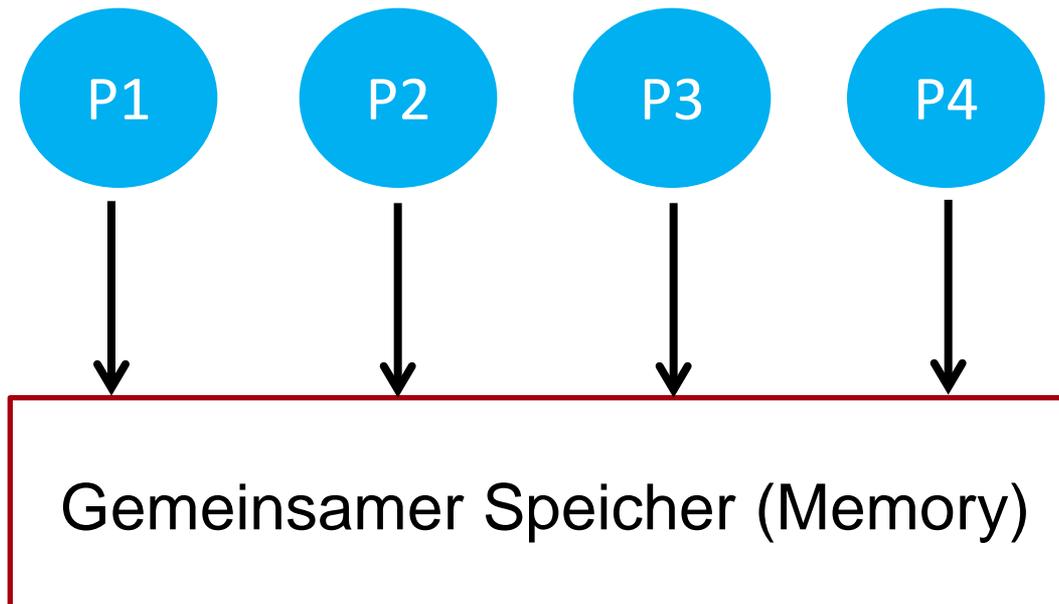
Wichtigstes Kriterium für OpenMP ist die Nutzung eines gemeinsamen Speichers.  
(shared memory application)



## OpenMP – Allgemeine Einführung II

OpenMP Merkmal gemeinsamer Speicher:

SMP:  
Symmetrisches Multiprozessersystem  
(symmetric multiprocessing)





## OpenMP – Allgemeine Einführung III

**Eine Stärke von OpenMP ist der inkrementelle Einsatz in sequentiellen Programmen:**

OpenMP Direktiven werden in einem speziellen Format angegeben,  
die nur der OpenMP Compiler versteht,  
für den regulären FORTRAN Compiler aber nur als Kommentare interpretiert werden.

Erster Schritt für OpenMP Implementierung:

**Finde Bereiche zur Parallelisierung** im sequentiellen FORTRAN Programm !

Dann **inkrementelle Einarbeitung** von OpenMP in existierendes FORTRAN Programm.

(Vorteil: rein sequentielles Debugging immer noch möglich)



# OpenMP – Allgemeine Einführung IV

## OpenMP Inkrementelle Einsatz in sequentiellm FORTRAN Programm:

Subroutine saxpy (z,a,x,y,n)

Integer :: i, n

Real :: a, y, z(n), x(n)

do i = 1,n

    z(i) = a + x(i)+y

enddo

return

end

Subroutine saxpy (z,a,x,y,n)

Integer :: i, n

Real :: a, y, z(n), x(n)

!\$omp parallel do

do i = 1,n

    z(i) = a + x(i)+y

enddo

!\$ end parallel do

return

end



## OpenMP: Prozesse und Threads (nach Vorlesung Hochleistungsrechnen, Prof. Ludwig)

**Traditioneller Prozess:** Ein Prozeß ist der Ablauf einer Programms

Aus Betriebssystemansicht:

- Prozeß ist Einheit der Ressourcenbelegungen (Speicher, Dateien, E/A-Ports)
- Prozeß ist Einheit der Prozessorsteuerung

Die Grundidee von **Threads** (als „leichtgewichtige Prozesse“) liegt in der Aufspaltung dieser Eigenschaften:

- Prozeß ist Einheit der Ressourcenbelegungen
- Thread ist Einheit der Prozessorsteuerung

d.h. eine Aufspaltung innerhalb eines Prozesses



## OpenMP: Prozesse und Threads (nach Vorlesung Hochleistungsrechnen, Prof. Ludwig)

### Eigenschaften von **Threads**:

Die Threads eines Prozesses haben gemeinsame Ressourcen (Speicher, Dateien, ...), d.h. Einfache, effiziente Kooperation möglich

**ABER:** kein wechselseitiger Schutz (Probleme wenn 2 Threads gleiche Datei öffnen)

### **Threadsichere Programmierung:**

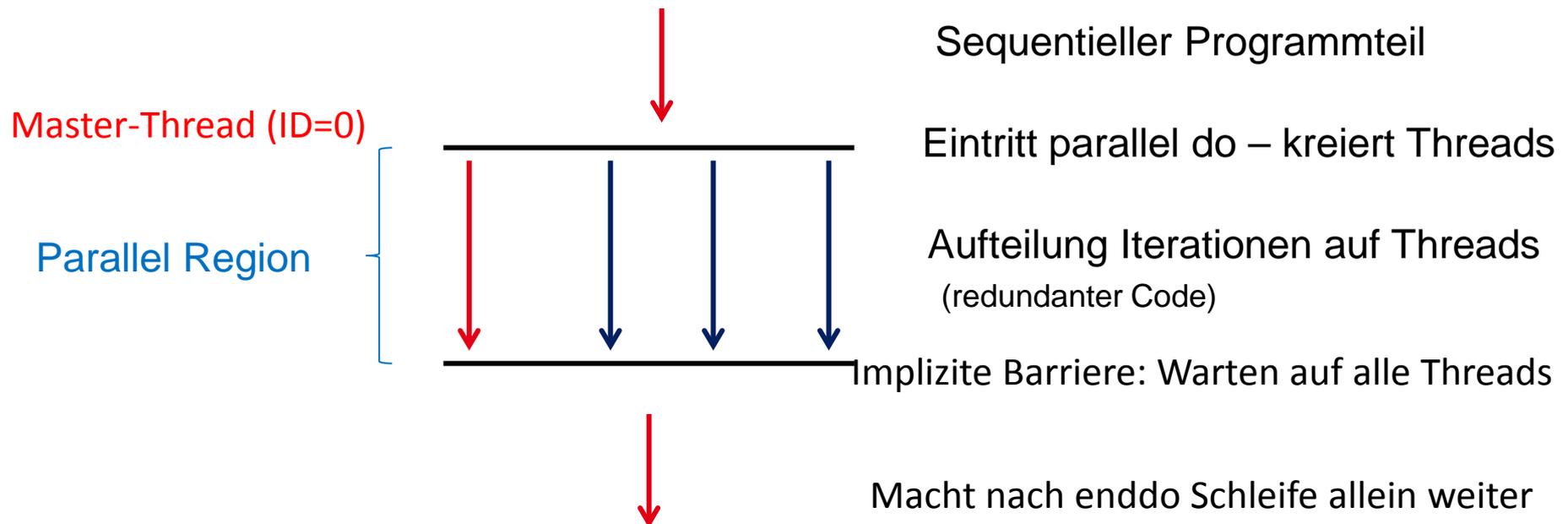
Codeteile müssen von Threads korrekt ausgeführt werden

d.h. kein gleichzeitig schreibender Zugriff auf dieselbe Speicherposition



# OpenMP Fork-Join Programmier Model

## OpenMP Merkmale!





# OpenMP Aufruf und Notation

Aufruf eines OpenMP Programmes:

```
gfortran -fopenmp -o hello hello.f90
```

OpenMP Notation im FORTRAN Code:

```
!$OMP Direktive [Bestimmung1], [Bestimmung2], ..., [Bestimmung n]
```



# OpenMP - Einführungsbeispiel I

Subroutine mxv (m,n,a,b,c)                   ! Matrix-Vektor Produkt

implicit none

integer (kind=4) :: m, n

real (kind = 8)    :: a(1:m), b(1:m,1:n), c(1:n)

integer            :: i,j

!\$OMP PARALLEL DO DEFAULT(NONE)   &

!\$OMP SHARED (m,n,a,b,c) PRIVATE(i,j)

  do i = 1,m

    a(i) = 0.0

    do j = 1,m

      a(i) = a(i) + b(i,j) \* c(j)

    enddo

  enddo

!\$ END PARALLEL DO

....



# OpenMP Einführungsbeispiel II

Subroutine mxv (m,n,a,b,c)

! Matrix-Vektor Produkt

.....

!\$OMP PARALLEL DO DEFAULT(NONE) &

!\$OMP SHARED (m,n,a,b,c) PRIVATE(i,j)

do i = 1,m

a(i) = 0.0

do j = 1,m

a(i) = a(i) + b(i,j) \* c(j)

enddo

enddo

!\$ END PARALLEL DO

....

} parallele Region

OpenMP kann angewendet werden da:

- \* Iterationen voneinander unabhängig sind

z.B.  $vv(i,j) = vv(i+1,j) + a(i,j)$

hier würde der Wert der i+1 Iteration genutzt  
d.h. die Datenabhängigkeiten sind zu prüfen!

- \* die OpenMP Direktiven beziehen sich auf

„strukturierte Blöcke“ von FORTRAN Code

d.h. es darf keine Verzweigung (Branch)

in oder aus der parallelen Region geben



## OpenMP Einführungsbeispiel III

Subroutine mxv (m,n,a,b,c)

.....

```
!$OMP PARALLEL DO DEFAULT(NONE) &
!$OMP SHARED (m,n,a,b,c) PRIVATE(i,j)
```

```
do i = 1,m
```

```
    a(i) = 0.0
```

```
    do j = 1,m
```

```
        a(i) = a(i) + b(i,j) * c(j)
```

```
    enddo
```

```
enddo
```

```
!$ END PARALLEL DO
```

....

Die OpenMP Direktive (PARALLEL DO) bezieht sich auf die direkt nachfolgende FORTRAN Zeile (do i =1,m)

und parallelisiert damit auch nur die i-Schleife.

Verschachtelungen sind möglich.

Wichtig ist die Definition der Nachfolgezeile mittels & nach FORTRAN Notation!

(Syntaxfehler werden ohne Warnung ignoriert!)



## OpenMP Einführungsbeispiel IV

Subroutine mxv (m,n,a,b,c)

.....

!\$OMP PARALLEL DO DEFAULT(NONE) &

!\$OMP SHARED (m,n,a,b,c) PRIVATE(i,j)

do i = 1,m

a(i) = 0.0

do j = 1,m

a(i) = a(i) + b(i,j) \* c(j)

enddo

enddo

!\$ END PARALLEL DO

....

Die OpenMP Direktive beinhaltet ebenfalls die Regelung der Zugriffsrechte:

DEFAULT (NONE) überlässt dem User die persönliche Festlegung der Zugriffsrechte

SHARED (m,n,a,b,c) erlaubt allen Threads den Gleichzeitigen Zugriff auf die gelisteten Variablen

PRIVATE(i,j) setzt die beiden Variablen private, so dass jeder Thread nur Zugang zu einer lokalen, einmaligen Kopie der Variablen hat.



# OpenMP Syntax I

Folgende OpenMP Syntax kann in FORTRAN umgesetzt werden:

!\$OMP DO	Verteilt die Schleifen Iterationen auf die Threads
!\$OMP SECTION	Verteilt unabhängige Arbeitseinheiten z.B. Funktionen
!\$OMP SINGLE	Nur <b>EIN Thread</b> führt den Programmblock aus
!\$OMP WORKSHARE	Parallelisiert Array Syntax

Die Beendigung der parallelen Region erfolgt entsprechen mit !\$ end [Option], z.B.

!\$OMP END DO



## OpenMP Syntax II

**Folgende OpenMP Syntax steuert die Aufteilung der Iterationen auf Threads:**

OMP_SCHEDULE=	Definiert Aufteilung der Arbeiten (in Loops)
static	jeder Thread bekommt statisch gleiche Anzahl an Iterationen
dynamic	jeder Thread holt sich neue Arbeit sobald er fertig ist
guided	wie dynamic, aber zugewiesene Pakete werden kleiner



## OpenMP Bestimmung der Thread Anzahl

**Folgende OpenMP Syntax wird für die Auswahl der Threadanzahl verwendet:**

Aufruf über Subroutine Call im Program z.B. :

```
Call OMP_SET_NUM_THREADS (4)
```

Alternativ kann dies auch im Aufruf des Programmes erfolgen:

```
gfortran -fopenmp -o hello hello.f90
```

Kompilieren

```
export OMP_NUM_THREADS=4
```

Umgebungsvariable setzen

```
./hello
```

Ausführung



## OpenMP Thread Abfrage

**Folgende OpenMP Syntax ermöglicht Informationen zu den Threads:**

<code>OMP_GET_NUM_THREADS ()</code>	Anzahl der arbeitenden (in use) Threads
<code>OMP_GET_THREAD_NUM()</code>	ID des aktuellen Threads
<code>OMP_GET_MAX_THREADS()</code>	Maximale Anzahl der verfügbaren Threads
<code>OMP_GET_WTIME()</code>	Wall Clock Time

z.B. `tid = OMP_GET_THREAD_NUM()`                      `integer :: tid`

`tid = 0`

`tid = OMP_GET_THREAD_NUM()`