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

# OpenMP Einführung II:

- Parallele Konstrukte
- Clauses
- Synchronisation
- Reduction

# OpenMP – Parallele Konstrukte I

**Parallel Konstrukt:**

!$omp parallel [clausel 1, …,clausel n] -> siehe Quick reference

Parallele Region

!$ omp end parallel

Innerhalb der vom "!$omp parallel" Konstrukt aufgespannten parallelen Region werden Konstrukte für die Verteilung auf die Threads benutzt, z.B.

| | |
|---|---|
| !$omp do | Iteration in Schleifen |
| !$omp sections | unabhängige Arbeitseinheiten |
| !$omp workshare | Parallelisiert Array Syntax |

# OpenMP – Parallele Konstrukte II

**Parallel Konstrukt kombiniert mit Section:**

```
!$omp parallel                          =>  Beginn Parallele Region
    !$omp sections
        !$omp section
                call subroutine A        - strukturierter Block A

        !$omp section
                call subroutine B        - strukturierter Block B
    !$omp end sections                   - Ende Sections Blöcke A+B
!$ omp end parallel                      => Ende parallele Region
```

! Keine Annahme über Reihenfolge

Load-Balance Probleme können auftauchen!

# OpenMP – Parallele Konstrukte III

**Combined Konstrukt kombiniert mit Workshare:** **(nur in FORTRAN!)**

```
!$omp parallel  workshare shared (n,a,b,c)
        b(1:n) = b(1:n) +1
        c(1:n) = c(1:n) + 2
        a(1:n) = b(1:n) + c(1:n)
!$ omp end parallel  workshare
```

! Es wird nicht spezifiziert wie die Arbeitseinheiten auf die Threads zugeteilt werden

! User muss für Parallelität in den Daten sorgen
(es darf keine versteckten Abhängigkeiten geben)

# OpenMP – Clauses I

**Die OpeMP – Clauses siehe auch Quick Reference pro Direktive**

**SHARED** erlaubt allen Threads den gleichzeitigen Zugriff auf die gelisteten Variablen

**PRIVATE** setzt die gelisteten Variablen private, so dass jeder Thread nur Zugang zu einer lokalen, einmaligen Kopie der Variablen hat.

# OpenMP – Clause II

**DEFAULT** setzt z.B. mit DEFAULT(shared)

alle zugewiesenen Variablen auf shared.

Wird benutzt um schnell der Mehrzahl der Variablen

eine Attribut zuweisen zu können.

# OpenMP – Clause III

**SCHEDULE** nur für Loops anzuwenden

Syntax: !$omp do schedule(kind[,chunk_size])

**static** die direkteste Zuordnung mit dem wenigsten Overhead

Iterationen werden in Portionen der Größe *chunk_size* aufgeteilt

ohne Angabe von *chunk_size* wird der Iterationsraum gleichmäßig auf die Threads aufteilt

# OpenMP – Clause IV

**SCHEDULE** nur für Loops anzuwenden

Syntax: !$omp do schedule(kind[,chunk_size])

**dynamic** Iterationen werden nach der Verfügbarkeit der Threads zugewiesen. Jeder freie Thread bekommt einen Chunk zugewiesen bis alle Iterationen abgearbeitet sind.

**guided** wie dynamic, nur dass die Chunks der noch zu bearbeitenden Iterationen immer kleiner werden.

# OpenMP – Reduction I

Die **REDUCTION CLAUSE** wird von OpenMP bereitgestellt um wiederkehrende Berechnungen, z.B. Summationen, einfach durchzuführen.

Syntax: reduction ({operator I intrinsic_procedure_name} :list)

Sorgt aber intern auch für spezifische Zugriffsrechte.

Dazu folgendes Beispiel.

# OpenMP – Reduction II

Problemstellung:

```
!$omp   do
  do i = 1,1000
     a = a  + i
  end do
!$omp end parallel do
```

a ist „shared“ da alle Threads darauf zugreifen müssen

aber nur ein Thread soll zu einem gegebenen Zeitpunkt schreiben bzw. a updaten können

sonst würde ein undefinierbares Ergebnis erfolgen

Quelle: Miguel Hermanns

# OpenMP – Reduction III

Problemstellung:

```
!$omp   do        reduction(+:a)

  do j = 1,1000
                        nur ein Thread pro Zeiteinhait darf a verändern
     a = a  + i

   end do

!$omp end parallel do
```

Quelle: Miguel Hermanns

# OpenMP – Reduction II

Für die **REDUCTION CLAUSE** stehen folgende Operatoren und Initialwerte bereit:

| Operator | Initialwert |
|---|---|
| + / - | 0 |
| * | 1 |
| .and. / .eqv. | .true. |
| .or. / .neqv. | .false. |

| Intrinsische Funktion | Initialwert |
|---|---|
| max | kleinste Zahl in der reduction Elemente Liste |
| min | größte Zahl in der reduction Elemente Liste |

# OpenMP – Synchronisation

**BARRIER** sind Synchronisationspunkte bei denen die einzelnen Threads aufeinander warten. Keinem Thread wird erlaubt im Programm fortzufahren, bis alle anderen Threads ebenfalls diesen Programmpunkt erreicht haben.

Syntax: !$omp barrier

# Danke das wars!