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:

- Parallel und Workshare Konstrukte
- Clauses
- Synchronisation
- Reduction

OpenMP Combined Construct !!

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
!$OMP SINGLE	Nur **EIN Thread** führt den Programmblock aus
!$OMP WORKSHARE	Parallelisiert Array Syntax

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

!$OMP END DO

OpenMP – Parallel & Workshare Konstrukt I

Parallel Konstruct:

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

Parallele Region

!$ end parallel

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

!$omp do

!$omp sections

!$omp workshare

OpenMP – Parallel & Workshare Konstrukt II

Parallel Konstrukt kombiniert mit Section:

```
!$omp parallel
    !$omp sections                          ! Keine Annahme über Reihenfolge
        !$omp section
                call subroutine A           - strukturierter Block

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

Load-Balance Probleme können auftauchen!

OpenMP – Parallel & Workshare Konstrukt III

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

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

! Es wird nicht spezifiziert wie die Arbeitseinheiten auf die Threads zugeteilt werden
(vergleichbar zu do loop mit static , d.h. n wird auf alle threads aufgeteilt)

! 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.

NOWAIT übersteuert die implizite Barriere am Ende
eines Workshare-Konstruktes.
! Fine-Tuning für Geübte !

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

dynamic Iterationen werden nach der Verfügbarkeit der Threads zugewiesen

Guided ??????????????????????

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 – Clause IV

SCHEDULE nur für Loops anzuwenden

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

runtime die Auswahl des Schedules erfolgt während der Laufzeit des Programmes.

Das Schedule und chunk_size (optional) werden durch die Umgebungsvariable OMP_SCHEDULE definiert.

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

OpenMP – Reduction I

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

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

Zum Beispiel:
```
!$omp parallel for default(none) shared(n,a)  &
        reduction(+:sum)
        sum = sum + a                (Schleife über n)
```

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