

Parallel Regions und Work-Sharing Konstrukte

Um eine Parallelisierung von größeren Programmabschnitten, als es einzelne Schleifen sind, zu ermöglichen, stellt OpenMP als allgemeinstes Konzept die “Parallel Regions” Konstruktion zur Verfügung. Die entsprechende Direktive lautet in Fortran-90

```
!$omp parallel clause[,] [clause ...]  
    ...  
!$omp end parallel
```

und in C

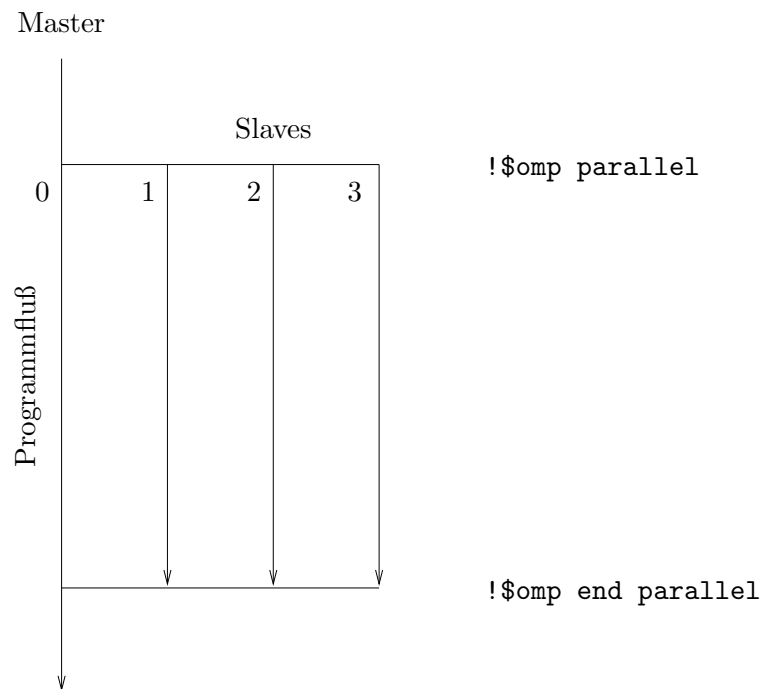
```
#pragma omp parallel [clause [clause ...]]  
{  
    ...  
}
```

wobei *clause* eine der Optionen

```
private(list)  
shared(list)  
default(private|shared|none)  
firstprivate(list)  
reduction(operator: list)  
if(expr)
```

(mit derselben Bedeutung wie bei der `parallel do/for` Direktive) sein kann.

Durch diese Konstruktion werden zusätzlich zu dem bis zu diesem Zeitpunkt allein aktiven “Master Thread” (dem normalen seriellen Programmablauf) eine Anzahl von parallelen “Slave Threads” erzeugt, die ebenfalls den durch die `parallel` Direktive eingeschlossenen Programmblock ausführen. Die Gesamtzahl (Master+Slaves) der Mitglieder dieses “Team of Threads” ist durch den Wert der Umgebungsvariablen `OMP_NUM_THREADS` gegeben und kann durch die Prozedur `omp_set_num_threads` zur Laufzeit verändert werden. Das Ende der `parallel` Direktive stellt einen impliziten Synchronisationspunkt dar: Erst wenn alle Threads mit der Ausführung ihres Exemplars des Programmblocks fertig sind, werden die Slave Threads beendet, und die Kontrolle des Programmablaufs geht wieder allein an den Master Thread über. Threads werden in OpenMP durch laufende Nummern identifiziert, beginnend mit 0, d.h. der Master Thread hat immer die Nummer 0, während die Slave Threads die Nummern 1, 2, 3, ... erhalten.



Die `parallel` Direktive bewirkt zunächst nur, daß der in der Parallel Region enthaltene Codeblock so oft vervielfacht und unabhängig ausgeführt wird, wie Threads vorhanden sind. Da dies nicht zu einer Beschleunigung, sondern zu einer Erhöhung des Rechenaufwandes führen würde, sind in OpenMP drei “Work-Sharing” Konstrukte vorgesehen, mit deren Hilfe die Ausführung von Teilabschnitten¹ der Parallel Region nicht-redundant auf die Threads aufgeteilt werden kann: die `do/for` Direktive, die `sections` Direktive und die `single` Direktive.

Die `do/for` Direktive hat in Fortran-90 die Gestalt

```
!$omp do [clause[,] [clause ...]]
  do-loop
!$omp end do [nowait]
```

bzw. in C

```
#pragma omp for [clause [clause ...]]
  for-loop
```

wo *clause* eine der Optionen `private`, `firstprivate`, `lastprivate`, `reduction`, `ordered` oder `schedule` ist (in C auch `nowait`). Genauso wie bei der `parallel do/for` Direktive wird dadurch eine Aufteilung der Schleifeniterationen auf die Threads bewerkstelligt, sodaß Master und Slaves nicht mehr alle Schleifendurchgänge, sondern nur je einen Teil der Iterationen ausführen. Die frühere `parallel do/for` Direktive kann demnach als Kurzform für

¹Diese müssen nicht textuell innerhalb der Parallel Region stehen, sondern können sich z.B. auch in einem daraus aufgerufenen Unterprogramm befinden; man bezeichnet sie dann als “verwaiste” (orphaned) Konstrukte.

```

!$omp parallel
!$omp do
    do-loop
!$omp end do
!$omp end parallel

```

angesehen werden. Einzelne Schleifen lassen sich im Prinzip auf jede der beiden Arten parallelisieren. Der Vorteil der `do/for` gegenüber der kombinierten `parallel do/for` Direktive liegt jedoch darin, daß bei Parallelisierung mehrerer Schleifen innerhalb ein und derselben Parallel Region der Aufwand zur Erzeugung und Beendigung der Slave Threads nur einmal anfällt:

```

!$omp parallel
    ...
!$omp do
    do-loop_1
!$omp end do
    ...
!$omp do
    do-loop_2
!$omp end do
    ...
!$omp end parallel

```

Das Ende jeder `do/for` Direktive stellt einen impliziten Synchronisationspunkt dar, an dem die Threads warten müssen, bis alle Mitglieder des Teams ihre Schleifendurchgänge beendet haben. Soll am Ende der Schleife *keine* Synchronisation stattfinden, ist die `nowait` Option anzugeben.

Die `sections` Direktive hat in Fortran-90 die Gestalt

```

!$omp sections clause[,] [clause ...]
[!$omp section]
    section_1
[!$omp section]
    section_2
    ...
]
!$omp end sections [nowait]

```

bzw. in C

```

#pragma omp sections [clause [clause ...]]
{
    [#pragma omp section]
        section_1

```

```

    [#pragma omp section
      section_2
      ...
    ]
}

```

wo *clause* eine der Optionen `private`, `firstprivate`, `lastprivate` oder `reduction` (in C auch `nowait`) ist. Durch diese Konstruktion werden die Programmabschnitte *section_1*, *section_2* usw. verschiedenen Threads zugeteilt, sodaß jeder Abschnitt nicht redundant, sondern nur genau einmal ausgeführt wird. Welcher Abschnitt von welchem Thread ausgeführt wird, ist allerdings implementations- und laufzeitabhängig. Sind weniger Abschnitte als Threads vorhanden, bleiben einzelne Threads unbeschäftigt; ist umgekehrt die Anzahl der Abschnitte größer als die Zahl der Threads, müssen manche Threads mehrere Abschnitte ausführen. Das Ende (aber nicht der Anfang) der `sections` Direktive stellt einen impliziten Synchronisationspunkt dar, der durch die Option `nowait` unterdrückt werden kann. Wie bei der `parallel do/for` Direktive gibt es auch hier eine kombinierte Konstruktion

```

!$omp parallel sections
[!$omp section]
  section_1
[!$omp section]
  section_2
  ...
]
!$omp end parallel sections

```

Die `single` Direktive hat in Fortran-90 die Gestalt

```

!$omp single [clause[,] [clause ...]]
  block
!$omp end single [nowait]

```

bzw. in C

```

#pragma omp single [clause [clause ...]]
  block

```

wo *clause* eine der Optionen `private` oder `firstprivate` (in C auch `nowait`) ist. Bei Verwendung dieser Konstruktion in einer Parallel Region wird erreicht, daß der eingeschlossene Programmblock nur von einem einzigen Thread ausgeführt wird, wobei gleichgültig ist, welcher Thread das ist. Ein typischer Anwendungsfall sind I/O-Operationen, die in der Regel nur von einem Thread ausgeführt werden sollen. Das Ende (aber nicht der Anfang) der `single` Direktive ist ebenfalls ein impliziter Synchronisationspunkt, der wieder durch die `nowait` Option unterdrückt werden kann.

Obwohl es sich dabei um kein Work-Sharing Konstrukt im eigentlichen Sinn handelt, hat eine ähnliche Funktion die `master` Direktive

```
!$omp master
  block
!$omp end master
```

bzw. in C

```
#pragma omp master
  block
```

Der eingeschlossene Programmblock wird nur vom Master Thread ausgeführt, von den übrigen Threads aber übersprungen. Im Gegensatz zum `single` Konstrukt findet am Ende der `master` Direktive keine implizite Synchronisation statt.