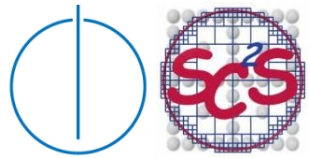
# Masterpraktikum - Scientific Computing

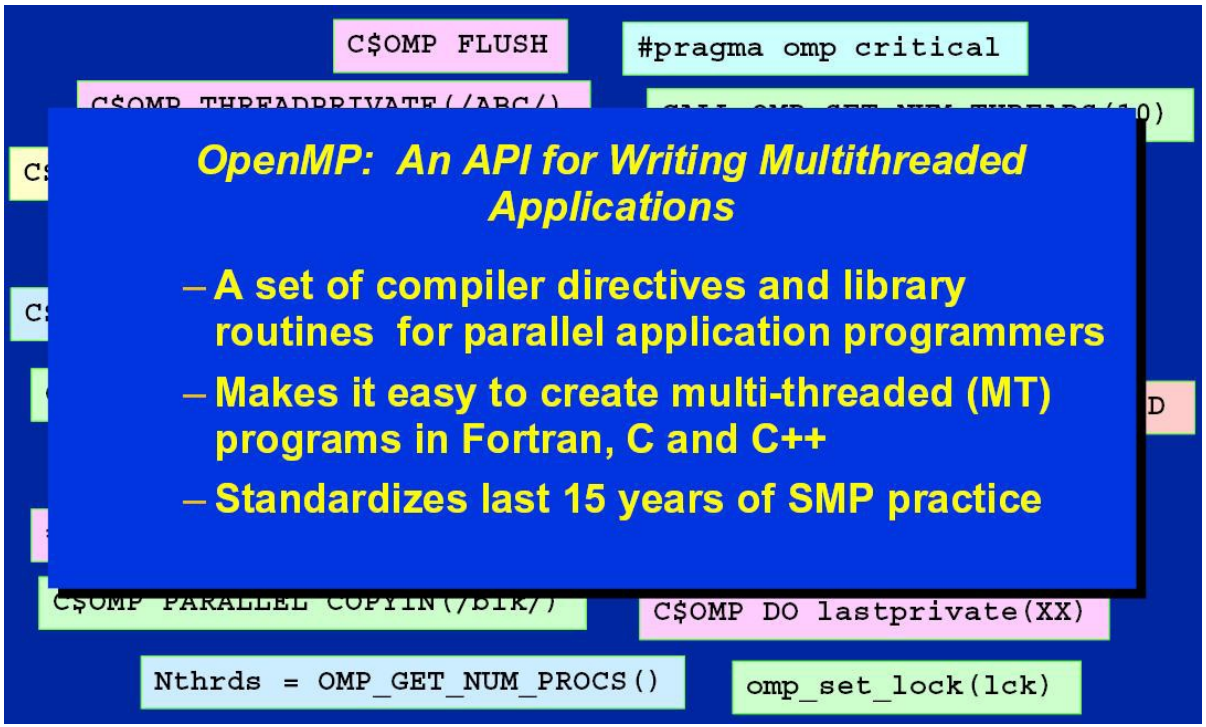
## High Performance Computing

Thomas Auckenthaler
Wolfgang Eckhardt
Prof. Dr. Michael Bader

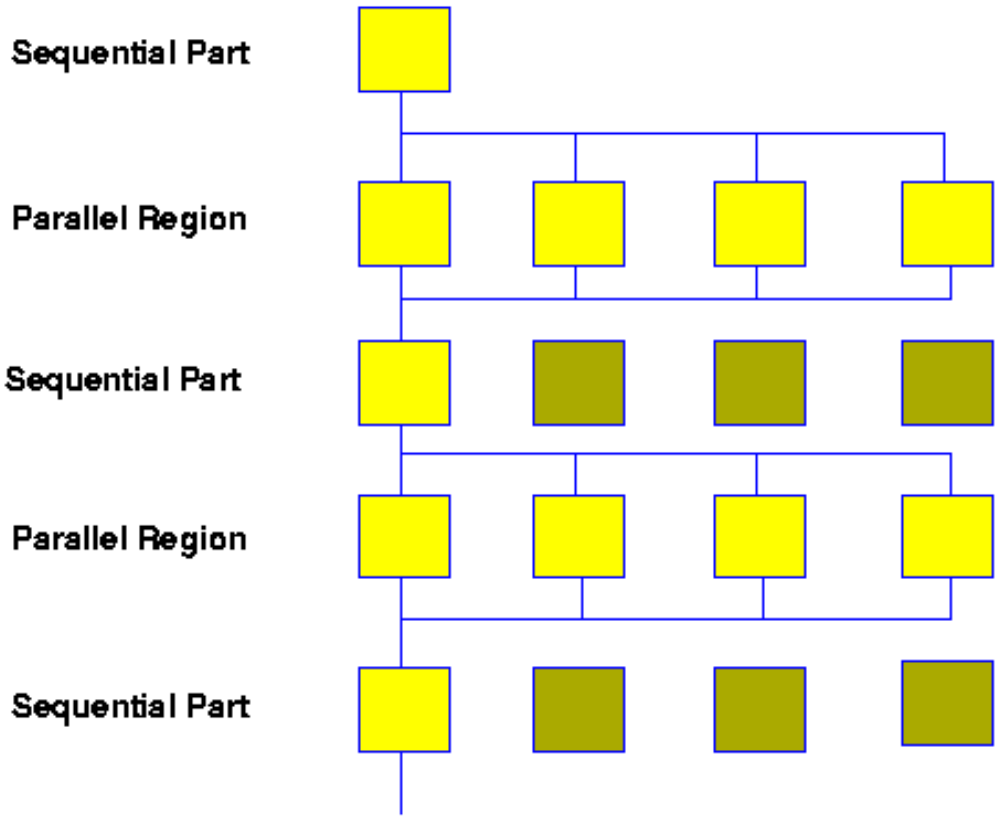
Technische Universität München, Germany



OpenMP


C$OMP FLUSH
#pragma omp critical
OpenMP: An API for Writing Multithreaded Applications
– A set of compiler directives and library routines for parallel application programmers
– Makes it easy to create multi-threaded (MT) programs in Fortran, C and C++
– Standardizes last 15 years of SMP practice
C$OMP DO lastprivate(XX)
Nthrds = OMP_GET_NUM_PROCS()
omp_set_lock(lck)

# OpenMP Execution Model



# Überblick

- Compiler-Direktiven
  `#pragma omp ... [clause[[,] clause]...]`
  - Work-sharing
    `#pragma omp for ...`, u.a.
  - Synchronisation
    `#pragma omp barrier`, u.a.
- Data Scope Clauses
  `shared`, `private`, `firstprivate`, `reduction`, u.a.
- Bibliotheksroutinen
  `omp_get_num_treads()`,
  `omp_get_tread_num()`, u.a.

# Compilieren und ausführen

- compilieren mit zusätzlichem Compiler-Flag `-openmp`:
  ```
  icc -openmp -o my_alg my_alg.c
  ```
- ausführen:
  ```
  export OMP_NUM_THREADS=number_of_threads
  ```
  (default = 1)
  ```
  ./my_alg
  ```

# Parallel Region Construct, OpenMP API

```
#pragma omp parallel [clause[[,] clause]...]
structured block
```

- Anweisungen innerhalb einer `parallel region` werden von **allen** Threads ausgeführt

Beispiel

```
#include <omp.h>
...
#pragma omp parallel private(size, rank)
{
   size = omp_get_num_threads();
   rank = omp_get_thread_num();
   printf("Hello World! (Thread %d of %d)", rank,
size);
}
```

# work sharing constructs

- `#pragma omp for [clause[[,] clause]...]`
  `for-loop`
  - steht innerhalb einer `parallel region` und direkt vor einer `for`-Schleife
  - die Iterationen der Schleife werden auf die Threads verteilt
  - mit der `schedule clause` wird festgelegt, wie die Iterationen auf die Threads verteilt werden
    `schedule(static[, chunksize])` Default-Wert für `chunksize` ist #Iterationen durch #Threads
    `schedule(dynamic[, chunksize])` Default-Wert für `chunksize` ist 1
    `schedule(runtime)` Scheduling wird durch Umgebungsvariable `OMP_SCEDULE` festgelegt
  - Synchronisation am Ende der `for`-Schleife (kann mit clause `nowait` unterdrückt werden)
  - Kurzform `#pragma omp parallel for`

# work sharing constructs II

- `#pragma omp sections [clause[[,] clause]...]`

```
{
   [#pragma omp section]
   structured block
   [#pragma omp section
   structured block]
   ...
}
```

  - ein freier Thread bearbeitet jeweils eine `section`

# work sharing constructs III

- `#pragma omp single [clause[[,] clause]...]`
  `structured block`
  - Bereich wird nur von einem (beliebigen) Thread ausgeführt
  - implizite Synchronisation am Ende des Blocks
- `#pragma omp master`
  `structured block`
  - Bereich wird nur vom Master-Thread ausgeführt
  - **keine** Synchronisation am Ende des Blocks

# synchronization constructs

- `#pragma omp barrier`
  - blockiert, bis alle Threads die barrier erreicht haben
- `#pragma omp critical`
  `structured block`
  - der Block nach `critical` darf nur von einem Thread gleichzeitig ausgeführt werden
  - ACHTUNG: potentieller Performance-Killer

# `reduction` clause

- `reduction` (*operator: list*)
  - führt eine Reduktion der Variablen in *list* mit dem Operator *operator* durch
  - mögliche Operatoren: $+$, $*$, $-$, &&, $||$

Beispiel

```
#pragma omp parallel for private(r), reduction(+:
sum)
for(i = 0; i < n; i++){
   r = compute_r(i);
   sum = sum + r;
}
```

# OpenMP data scope clauses

- `private`(*list*): deklariert die Variablen in *list* als private für jeden Thread.
- `shared`(*list*): Variablen aus *list* werden von allen Threads gemeinsam genutzt.
- `firstprivate`(*list*): private Variablen, die mit dem letztem Wert vor dem parallelen Abschnitt initialisiert werden.
- `lastprivate`(*list*): nach dem Verlassen des parallelen Abschnitts übernimmt die Variable den Wert von dem Thread, der die letzte Schleifeniteration ausgeführt hat.
- default data scope clause ist `shared`, mit Ausnahmen...
  - lokale Variablen sind `private`
  - Schleifen-Variablen der parallelisierten `for`-Schleifen sind `private`

# OpenMP data scope clauses - Übung

Beispiel

```
k = compute_k();
#pragma omp parallel for private(?), shared(?),
firstprivate(?), lastprivate(?), reduction(?)
for(i = 0; i < n; i++){
   k = compute_my_k(i, k);
   r = compute_r(i, k, h);
   sum = sum + r;
}
```

Welche *data scope clauses sind sinnvoll?*

# Fragen?