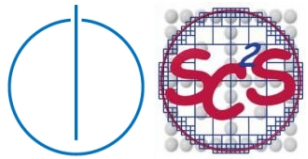
Masterpraktikum - High Performance Computing

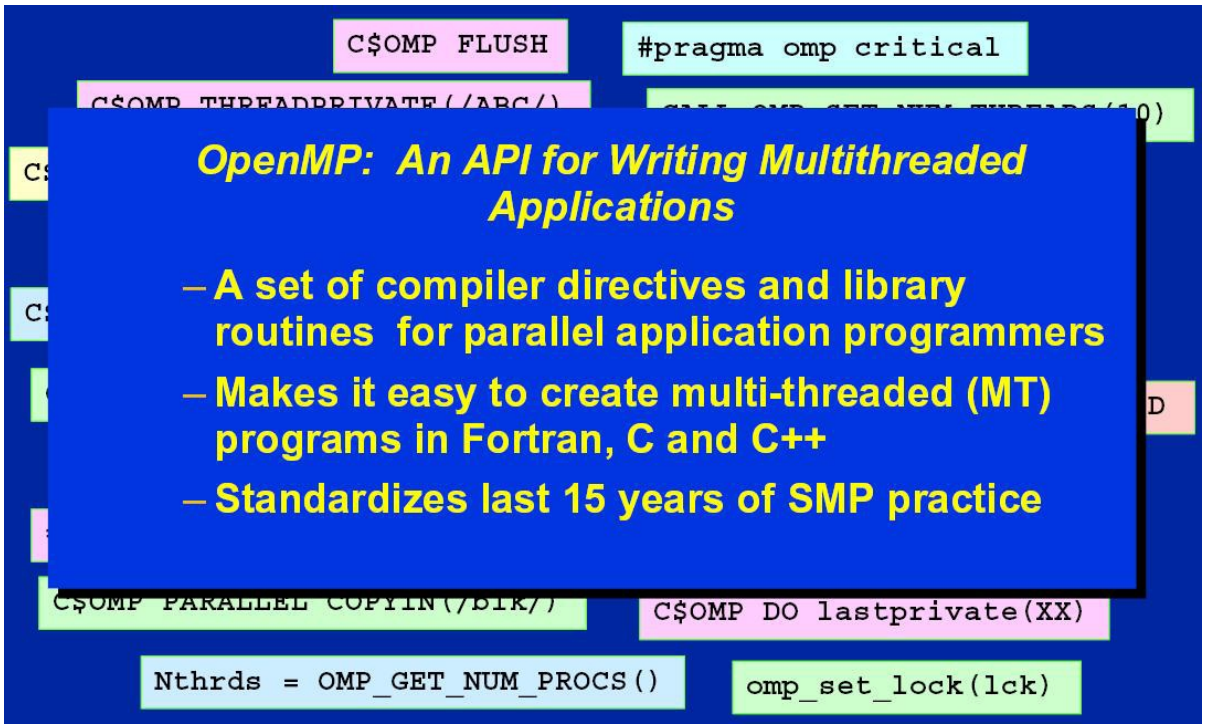
OpenMP

Michael Bader
Alexander Heinecke
Alexander Breuer

Technische Universität München, Germany

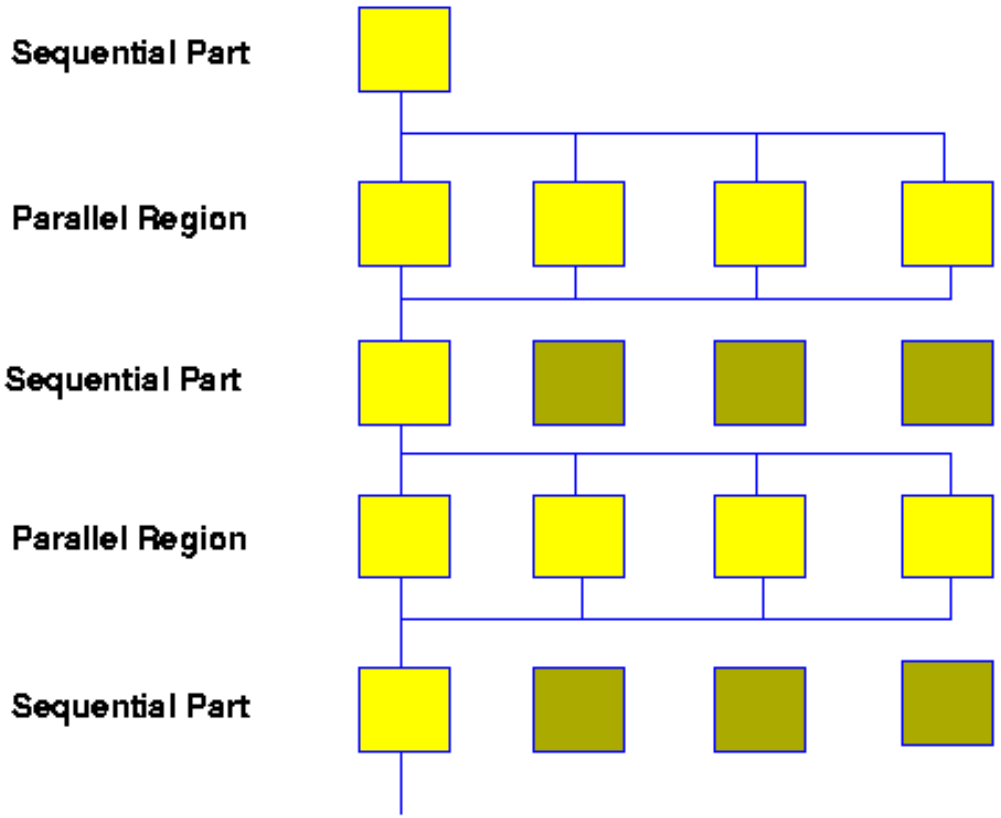


OpenMP


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)

```
#include <omp.h>
...
#pragma omp parallel for
for(i = 0; i < n; i++)
{
   for(j = 0; j < n; j++)
   {
      do_some_work(i, j);
   }
}
```

OpenMP Execution Model



Overview

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

compiling and excuting

- compile with additional Compiler-Flag `-openmp` (`-fopenmp` in case of gcc):
  ```
  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
```

- code within `parallel region` are executed by **all** threads

Example

```
#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`
  - within a `parallel region` and directly in front of a `for`-loop
  - iterations are scheduled across different threads
  - `schedule clause` determines how to map iterations to threads
    `schedule(static[, chunksize])` default-value `chunksize` is #iterations divided by #hreads
    `schedule(dynamic[, chunksize])` default-values `chunksize` is 1
    `schedule(runtime)` Scheduling is given by environment variable `OMP_SCEDULE`
  - implicit synchronization in the end of `for`-loop (can be disabled with `nowait` clause)
  - shortcut possible: `#pragma omp parallel for`

work sharing constructs II

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

where `clause` can be...

- `final(expression)`: if `expression` is `true`, task is executed sequentially; no recursive task generation
- `untied`: the execution of a task is not tied to one single thread
- `shared | firstprivate | private`
- some more...

```
#pragma omp taskwait
```

- waits for children completition

work sharing constructs III

- `#pragma omp single [clause[[,] clause]...]`
  `structured block`
  - use only one (arbitrary) thread
  - implicit synchronization
- `#pragma omp master`
  `structured block`
  - use only master thread for structures block
  - **NO** synchronization in the end

synchronization constructs

- `#pragma omp barrier`
  - blocks execution until all threads have reached the barrier
- `#pragma omp critical`
  `structured block`
  - only one thread a time can execute the structured blocked encapsulated by `critical`
  - ATTENTION: use carefully! Will definitely kill performance.

`reduction` clause

- `reduction` (*operator: list*)
  - executes a reduction of variables in *list* using *operator*
  - available operators: $+$, $*$, $-$, &&, $||$

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*): DECLARES variables in *list* as private for each thread (no copy!)
- `shared`(*list*): variables in *list* are used by all threads (race conditions are possible!), write accesses have to be handled by programmer
- `firstprivate`(*list*): private varibles AND init them with latest valid value before parallel region
- `lastprivate`(*list*): after parallel execution, in serial part reused latest modification.
- default data scope clause is `shared`, but exceptions exists $\Rightarrow$ be precise!
  - local variables are always `private`
  - loop-variables of `for`-loops are `private` (C++ only)

OpenMP data scope clauses - Exercise

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;
}
```

Questions?