OpenMP
Lecture 2

Manfred Liebmann
Technische Universität München
Chair of Optimal Control
Center for Mathematical Sciences, M17
manfred.liebmann@tum.de

December 8, 2015

Fork-Join Execution Model

OpenMP is an explicit programming model, offering the programmer full control over the parallelization. OpenMP uses the fork-join model of parallel execution. All OpenMP programs begin as a single process with a single master thread. The master thread executes sequentially until the first parallel region construct is encountered. The master thread then creates (forks) a team of parallel threads. The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads. When the team threads complete the statements in the parallel region construct, they synchronize and terminate, (join), leaving only the master thread. The number of parallel regions and the threads that comprise them are arbitrary.

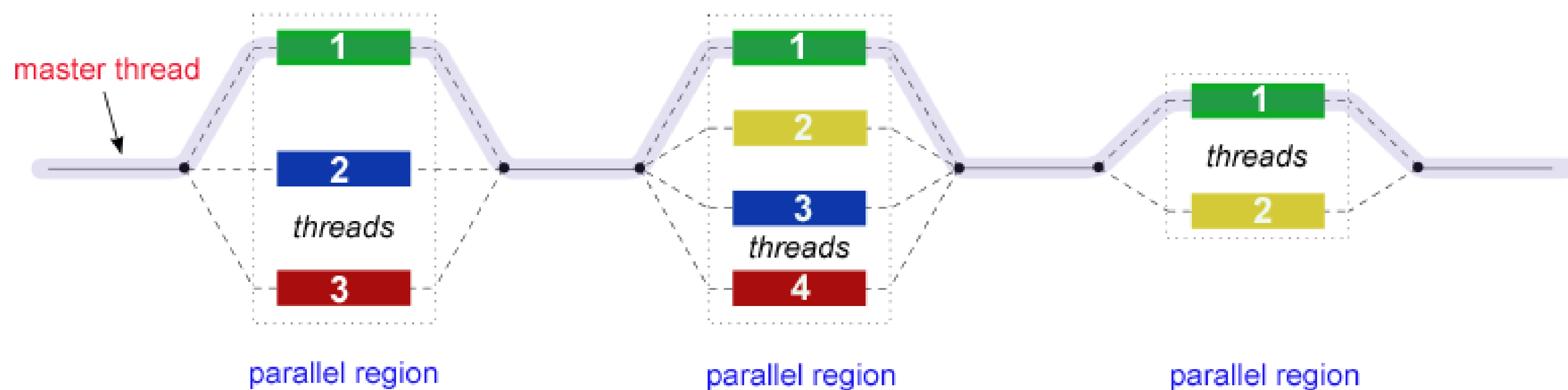

Figure 1: Fork-Join Execution Model

Work-Sharing Constructs

A work-sharing construct must be enclosed dynamically within a parallel region in order for the directive to execute in parallel. Work-sharing constructs must be encountered by all members of a team or none at all. Successive work-sharing constructs must be encountered in the same order by all members of a team.

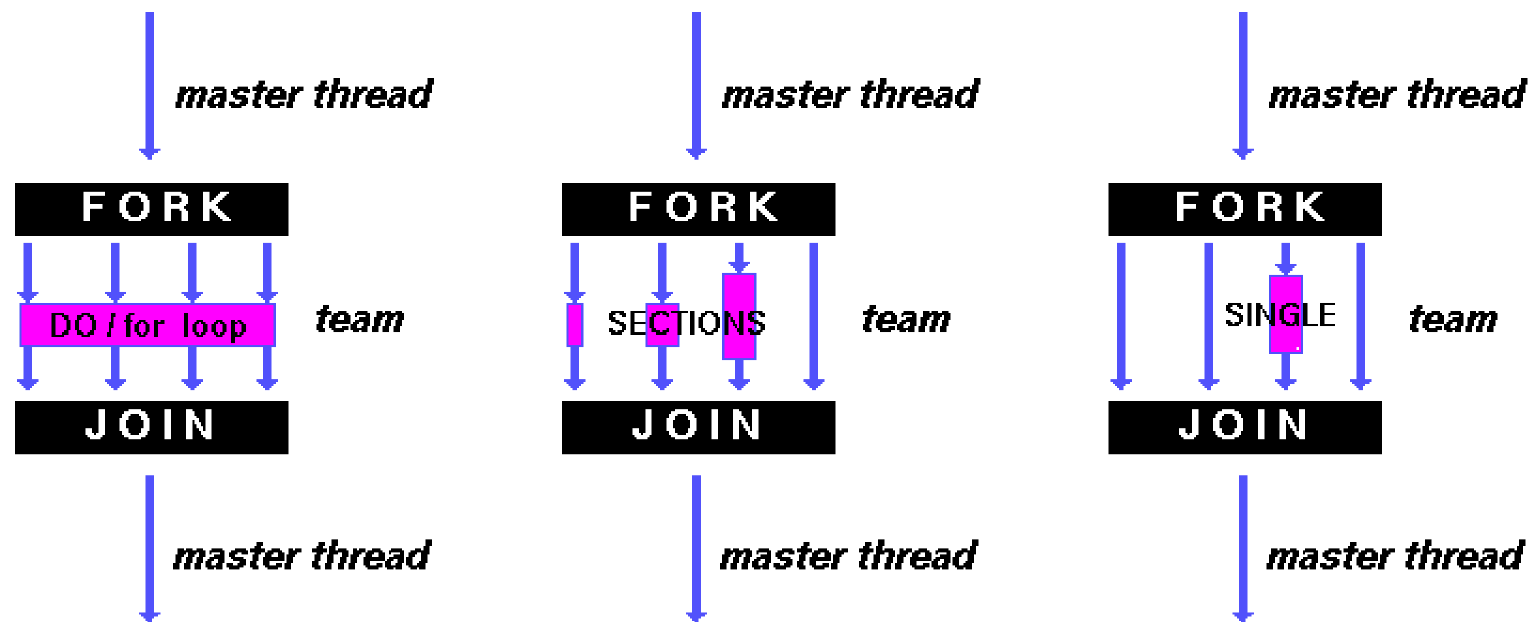

Figure 2: OpenMP work-sharing constructs: Data parallelism: for loop; Functional parallelism: sections; Serialize a code section: single

Levels of Control over Program Execution

Program execution is influenced by internal control variables (ICV). The ICVs can be configured with the following priorities:

- ICV default behavior
- Environment variables
- OpenMP routines
- OpenMP directives

The ICV default behavior is defined by the compiler and runtime depending on the execution environment.

Environment Variables

- OMP_SCHEDULE type[, chunk] (type: static, dynamic, guided, auto; chunk: chunk size)
- OMP_NUM_THREADS nth0[, nth1[, nth2,...]] (threads to use in nested levels)
- OMP_DYNAMIC true | false (dynamic adjustment of number of threads)
- OMP_PROC_BIND true | false (bind thread to processor)
- OMP_NESTED true | false (nested parallelism)
- OMP_STACKSIZE size | sizeB | sizeK | sizeM | sizeG (stack size for threads)
- OMP_WAIT_POLICY active | passive (active: consume processor cycles while waiting)
- OMP_MAX_ACTIVE_LEVELS levels (maximum number of nested parallel regions)
- OMP_THREAD_LIMIT threads (maximum number of threads for the whole program)

Runtime Library Routines

- Execution environment routines
  - omp_set_num_threads
  - omp_get_num_threads
  - omp_get_max_threads
  - omp_get_thread_num
  - omp_get_num_procs
  - omp_in_parallel
  - omp_set_dynamic
  - omp_get_dynamic
  - omp_set_nested
  - omp_get_nested
  - omp_set_schedule
  - omp_get_schedule
  - omp_get_thread_limit
  - omp_set_max_active_levels
  - omp_get_max_active_levels

  - omp_get_level
  - omp_get_ancestor_thread_num
  - omp_get_team_size
  - omp_get_active_level
  - omp_in_final

- Lock routines
  - omp_init_lock
  - omp_destroy_lock
  - omp_set_lock
  - omp_unset_lock
  - omp_test_lock
  - omp_init_nest_lock
  - omp_destroy_nest_lock

- Portable timer routines
  - omp_get_wtime
  - omp_get_wtick

Loop Directive Schedule Clauses

```
void simple(int n, float *a, float *b)
{
    int i;
#pragma omp parallel for schedule(kind,chunk)
    for (i=1; i<n; i++) /* i is private by default */
        b[i] = (a[i] + a[i-1]) / 2.0;
}
```

Schedule clause kind values: static, dynamic, guided, auto, runtime

- When schedule(static, chunk_size) is specified, iterations are divided into chunks of size chunk_size, and the chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number.
- When schedule(dynamic, chunk_size) is specified, the iterations are distributed to threads in the team in chunks as the threads request them. Each thread executes a chunk of iterations, then requests another chunk, until no chunks remain to be distributed. Each chunk contains chunk_size iterations, except for the last chunk to be distributed, which may have fewer iterations.

- When schedule(guided, chunk_size) is specified, the iterations are assigned to threads in the team in chunks as the executing threads request them. Each thread executes a chunk of iterations, then requests another chunk, until no chunks remain to be assigned. For a chunk_size of 1, the size of each chunk is proportional to the number of unassigned iterations divided by the number of threads in the team, decreasing to 1. For a chunk_size with value k (greater than 1), the size of each chunk is determined in the same way, with the restriction that the chunks do not contain fewer than k iterations (except for the last chunk to be assigned, which may have fewer than k iterations).

- When schedule(auto) is specified, the decision regarding scheduling is delegated to the compiler and/or runtime system. The programmer gives the implementation the freedom to choose any possible mapping of iterations to threads in the team.

- When schedule(runtime) is specified, the decision regarding scheduling is deferred until run time, and the schedule and chunk size are taken from the ICV.

CPU Thread Affinity

Bind OpenMP threads to specific CPU core:

```
export GOMP_CPU_AFFINITY="0-7"
```

The variable should contain a space-separated or comma-separated list of CPUs. This list may contain different kinds of entries: either single CPU numbers in any order, a range of CPUs (M-N) or a range with some stride (M-N:S). CPU numbers are zero based. For example, GOMP_CPU_AFFINITY="0 3 1-2 4-15:2" will bind the initial thread to CPU 0, the second to CPU 3, the third to CPU 1, the fourth to CPU 2, the fifth to CPU 4, the sixth through tenth to CPUs 6, 8, 10, 12, and 14 respectively and then start assigning back from the beginning of the list. GOMP_CPU_AFFINITY=0 binds all threads to CPU 0.

If both GOMP_CPU_AFFINITY and OMP_PROC_BIND are set, OMP_PROC_BIND has a higher precedence. If neither has been set and OMP_PROC_BIND is unset, or when OMP_PROC_BIND is set to FALSE, the host system will handle the assignment of threads to CPUs.

Reduction Clause

Reduction clause:

```
reduction(operator:list)
```

Reduction operations:

```
+  *  -  &  |  ^  &&  ||  max  min
```

Valid data types for the reduction clause: char, wchar_t, int, float, double, or bool, possibly modified with long, short, signed, or unsigned.