Advanced OpenMP

Advanced Topics

- Extents and Scoping
- Reality Check

Extent of directives

Most directives have as extent a structured block, or basic block, i.e., a sequence of statements with a flow of control that satisfies:

- there is only one entry point in the block, at the beginning of the block
- there is only one exit point, at the end of the block; the exceptions are that exit() in C and stop in Fortran are allowed

Types of Extents

Two types for the extent of a directive:

- static or lexical extent: the code textually enclosed between the beginning and the end of the structured block following the directive
- dynamic extent: static extent as well as the procedures called from within the static extent

Orphaned Directives

A directive which is in the dynamic extent of another directive but not in its static extent is said to be orphaned

- Work sharing directives can be orphaned
- This allows a work-sharing construct to occur in a subroutine which can be called both by serial and parallel code, improving modularity

Directive Binding

- Work sharing directives (do, for, sections, and single) as well as master and barrier bind to the dynamically closest parallel directive, if one exists, and have no effect when they are not in the dynamic extent of a parallel region
- The ordered directive binds to the enclosing do or for directive having the ordered clause
- critical (and atomic) provide mutual exclusive execution (and update) with respect to all the threads in the program

Directive Nesting

- A parallel directive can appear in the dynamic extent of another parallel, i.e., parallel regions can be nested
- Work-sharing directives binding to the same parallel directive cannot be nested
- An ordered directive cannot appear in the dynamic extent of a critical directive
- A barrier or master directive cannot appear in the dynamic extent of a work-sharing region (DO or for, sections, and single) or ordered block
- In addition, a barrier directive cannot appear in the dynamic extent of a critical or master block

Data Scoping

- Work-sharing and parallel directives accept data scoping clauses
- Scope clauses apply to the static extent of the directive and to variables passed as actual arguments
- The shared clause applied to a variable means that all threads will access the single copy of that variable created in the master thread
- The private clause applied to a variable means that a volatile copy of the variable is cloned for each thread

Data Scoping

- Semi-private data for parallel loops:
  - reduction: variable that is the target of a reduction operation performed by the loop, e.g., sum
  - firstprivate: initialize the private copy from the value of the shared variable
  - lastprivate: upon loop exit, master thread holds the value seen by the thread assigned the last loop iteration (for parallel loops only)

Reality Check

- Irregular and ambiguous aspects are sources of language- and implementation dependent behavior:
- nowait clause is allowed at the beginning of [parallel] for (C/C++) but at the end of [parallel] DO (Fortran)
- default clause can specify private scope in Fortran, but not in C/C++
- Can only privatize full objects, not array elements, or fields of data structures
- For a threadprivate variable or block one cannot specify any clause except for the copyin clause
- In some implementations one cannot specify in the same directive both the firstprivate and lastprivate clauses for a variable

Reality Check

- With MIPSpro 7.3.1, when a loop is parallelized with the do (Fortran) or for (C/C++) directive, the indexes of the nested loops are, by default, private in Fortran, but shared in C/C++
  - Probably, this is a compiler issue
  - Fortunately, the compiler warns about unsynchronized accesses to shared variables
  - This does not occur for parallel do or parallel for