OpenMP Fortran Summary: Draft 1.0

This is a brief summary of the OpenMP Fortran syntax. To learn how OpenMP works, you'll need to get a copy of the specification from www.openmp.org.

Directive format

```
sentinel directive-name [clause[ clause]…]
```

where sentinel is either fixed source:

```
!$OMP | C$OMP | *$OMP
```

or free source form

```
!$OMP
```

Without loss of generality, we will use the C$OMP sentinel in this summary.

Restrictions

- Fixed form sentinels must start in column 1 and have a space or zero in column 6 unless the sentinel indicates a continuation line
- Free source sentinels can appear in any column as long as it is preceded only by white space. Continuation indicated with an ampersand on the last non-blank line.

Examples

Fixed source:

```
C$OMP PARALLEL DO
C$OMP+SHARED(A, B, C)
```

Free source:

```
!$OMP PARALLEL DO &
!$OMP SHARED(A, B, C)
```

Conditional compilation

```
sentinel [legal-Fortran-statement]
```

Where the sentinels vary depending on fixed source

```
!$ | C$ | *$
```

or free source

```
!$
```

parallel construct

```
C$OMP PARALLEL [clause[ clause] …]
      STRUCTURED-BLOCK
C$OMP END PARALLEL
```

Where clause is:

```
IF (SCALAR-LOGICAL-EXPRESSION)
PRIVATE(LIST)
FIRSTPRIVATE(LIST
DEFAULT (PRIVATE | SHARED | NONE)
SHARED(LIST)
COPYIN(LIST)
REDUCTION({OPERATOR|INTRINSIC}: LIST)
```

Restrictions

- The PARALLEL/END-PARRAL directive pair must appear in the same routine in the executable section of the code.
- A STRUCTURED-BLOCK is a set of statements with a single entry at the top and a single exit at the bottom. It is illegal to branch into or out of a structured block.
- There is an implied barrier at the END PARALLEL construct.
- At most one if clause can appear on the directive

Work-sharing Constructs

DO directive

```
C$OMP DO [clause[[,] clause] …]
      do_loop
[C$OMP END DO [NOWAIT]]
```

Where clause is:

```
PRIVATE(LIST)
FIRSTPRIVATE(LIST)
REDUCTION({OPERATOR|INTRINSIC}: LIST)
ORDERED
SCHEDULE(TYPE[, CHUNK_SIZE])
LASTPRIVATE(LIST)
```

Usage Note:

The schedule clause defines how the iterations are mapped onto the team of threads:

- `schedule(static[, chunk_size])`: iterations are divided into chunks of size chunk_size and assigned to the members of the team in a round robin fashion. If chunk_size isn't given, approximately equal sized chunks are assigned one to each thread.
- `schedule(dynamic[, chunk_size])`: iterations are divided into chunks of size chunk_size and assigned one-by-one to the threads as they finish the previous chunk of iterations. When no chunk_size is given, it defaults to 1.
- `schedule(guided[, chunk_size])`: iterations are assigned to threads as with the dynamic schedule, but the chunks are of decreasing sizes. The number of iterations in a chunk start at some large value and decrease down to chunk_size. If chunk_size equals 1, the size of each chunk is approximately the number of unassigned iterations divided by the number of threads in the team. If chunk_size isn't specified, it defaults to one.

- `schedule(runtime): The schedule and chunk size are determined at runtime by setting the runtime variable OMP_SCHEDULE. If this variable is not set, the behavior is implementation dependent.`

Restrictions:

- Work-sharing constructs must be encountered by all threads in a team or none at all.
- CHUNK-SIZE must be an integer or an integer expression.
- The values of the loop control parameters must be the same for all the threads in the team
- The DO loop iteration variable must be of type integer.
- If used, the END-DO directive must appear immediately after the end of the loop.
- Only a single schedule clause can appear on a DO directive
- Only a single ordered clause can appear on a DO directive.

sections/section directive

```
C$OMP SECTIONS [clause[ clause] …]
[C$OMP SECTION]
      STRUCTURED-BLOCK
[C$OMP SECTION
      STRUCTURED-BLOCK]
. . .
C$OMP END SECTIONS [NOWAIT]
```

Where clause is:
```
PRIVATE(LIST)
FIRSTPRIVATE(LIST)
LASTPRIVATE(LIST)
REDUCTION({OPERATOR|INTRINSIC}: LIST)
```

Restrictions:

- A SECTIONS directive must not be outside the lexical extent of the sections directive.

SINGLE directive

```
C$OMP SINGLE [clause[[,] clause] …]
      STRUCTURED-BLOCK
C$OMP END SINGLE [NOWAIT]
```

Where clause is:
```
PRIVATE(LIST)
FIRSTPRIVATE(LIST)
```

Combined Parallel Work-sharing Constructs

PARALLEL DO directive

```
C$OMP PARALLEL DO [clause[ clause] …]
      do-loop
[C$OMP END PARALLEL DO]
```

Where clause is:
```
IF (SCALAR-EXPRESSION)
PRIVATE(LIST)
FIRSTPRIVATE(LIST)
DEFAULT (SHARED | NONE)
SHARED(LIST)
COPYIN(LIST
SCHEDULE(TYPE[, CHUNK_SIZE])
ORDERED
LASTPRIVATE(LIST)
REDUCTION({OPERATOR|INTRINSIC}: LIST)
```

Usage Note:

The construct is the same as a PARALLEL construct immediately followed by a DO work sharing directive.

Restrictions:

This construct shares restrictions with the PARALLEL and DO directives.

parallel sections construct

```
#C$OMP PARALLEL SECTIONS [clause[[,]clause] … ]
[C$OMP SECTION]
      structured-block
[C$OMP SECTION
      structured-block]
. . .
C$OMP END PARALLEL SECTIONS
```

Where clause is:
```
IF (SCALAR-EXPRESSION)
PRIVATE(LIST)
FIRSTPRIVATE(LIST)
DEFAULT (SHARED | NONE)
SHARED(LIST)
COPYIN(LIST
LASTPRIVATE(LIST)
```

```
REDUCTION({OPERATOR|INTRINSIC}: LIST)
```

Restrictions:

This construct shares restrictions with the PARALLEL and SECTIONS constructs.

Master and Synchronization Constructs

MASTER directive

```
C$OMP MASTER
      STRUCTURED-BLOCK
C$OMP END MASTER
```

CRITICAL directive

```
C$OMP CRITICAL [(name)]
      STRUCTURED-BLOCK
C$OMP END CRITICAL [(name)]
```

Where name is: An identifier

BARRIER directive

```
C$OMP BARRIER
```

ATOMIC directive

```
C$OMP ATOMIC
      expression-stmt
```

Usage Note:

The atomic construct is semantically equivalent to critical statement. The single expression-stmt must use one of the following forms:

```
x = x operator expr
x = expr operator x
x = intrinsic (x, expr)
x = intrinsic (expr, x)
```

Where *x* is a scalar variable of intrinsic type.

expr is a scalar expression that does not reference x.

intrinsic is one of MAX, MIN, IAND, IOR or IEOR

operator is one of +, *, -, /, .AND, .OR., .EQV. or .NEQV.

Restriction

All references to the storage location x are required to have the same type and type parameters.

FLUSH directive

```
C$OMP FLUSH [(list)]
```

Where List is a comma-separated list of variables that need to be flushed

ORDERED directive

```
C$OMP ORDERED
      Structured-block
C$OMP END ORDERED
```

Restrictions:

- An ordered directive must not be in the dynamic extent of a do directive that does not have the ordered clause specified.
- An iteration of a loop with a do construct must not execute the same ordered directive more than once, and it must not execute more than one ordered directive.

Data Environment Constructs

THREADPRIVATE directive

```
C$OMP THREADPRIVATE(/cp[,/cb/] …)
```

Where *cb* is the name of the common block to be made private to a thread.

Restrictions

- The THREADPORIVATE directive must appear after every declaration of a thread private common block.
- Only named common blocks can be made thread private.
- It is illegal for a THREADPRIVATE common block or its constituent variables to appear in any clause other than a COPYIN clause. They are not affected by the DEFAULT clause.

PRIVATE clause

```
PRIVATE(list)
```

Restrictions:

- Variables that are specified private on a parallel directive cannot be specified private again on an enclosed work-sharing directive. As a result, variables that are specified private on a work-sharing directive must be specified shared in the enclosing parallel region

FIRSTPRIVATE clause

```
FIRSTPRIVATE(list)
```

LASTPRIVATE clause

```
LASTPRIVATE(list)
```

SHARED clause

```
SHARED(list)
```

DEFAULT clause

```
DEFAULT(PRIVATE | SHARED | NONE)
```

Restrictions:

- Only a single default clause may be specified on a parallel directive.

REDUCTION clause

```
REDUCTION ({operator | intrinsic}:list)
```

Where *operator* or *intrinsic* are one of:

```
   +         Initial value = 0
   *         Initial value = 1
   -         Initial value = 0
 .AND.       Initial value = .TRUE.
 .OR.        Initial value = .FALSE.
 .EQV.       Initial value = .TRUE.
.NEQV.       Initial value = .FALSE.
 MAX         Initial value = Smallest representable number
 MIN         Initial value = Largest representable number
IAND         Initial value = All bits on
 IOR         Initial value = 0
IEOR         Initial value = 0
```

Usage Note:

A reduction is typically used in a statement with one of the following forms:

```
x = x operator expr
x = expr op x (except for subtraction)
x = intrinsic (x, expr)
x = intrinsic (expr, x)
IF (X .LT. expr) X = expr
```

Restrictions:

- Variables that appear in a reduction clause must be SHARED in the enclosing context.
- Only variables with arithmetic type can appear in the list of variables for the reduction clause.

COPYIN clause

```
COPYIN (list)
```

where *list* contains thread private common blocks or variables included in a thread private common block.

Data Environment Rules

An OpenMP Fortran program must adhere to the following rules and restrictions with respect to data scope:

- Sequential DO loop control variables in the lexical extent of a PARALLEL region that would otherwise be SHARED based on default rules, are automatically made private on the PARALLEL directive.
- Variables that are privatized in a parallel region cannot be privatized again in an enclosed work-sharing directive. As a result, variables that appear in the PRIVATE, FIRSTPRIVATE, LASTPRIVATE, and REDUCTION clauses on a work-sharing directive must have shared scope in the enclosing parallel region.
- Assumed-size and assumed-shape arrays cannot be specified as PRIVATE, FIRSTPRIVATE, or LASTPRIVATE.
- Fortran pointers and allocatable arrays can be declared as PRIVATE or SHARED but not as FIRSTPRIVATE or LASTPRIVATE.
- Within a parallel region, the initial status of a private pointer is undefined.
- Scope clauses apply only to variables in the static extent of the directive on which the clause appears, with the exception of variables passed as actual arguments. Local variables in called routines that don't have the SAVE attribute are PRIVATE. Common blocks and modules in called routines in the dynamic extent of a parallel region always have an implicit SHARED attribute, unless they are THREADPRIVATE common blocks.
- When a named common block is declared as PRIVATE, FIRSTPRIVATE or LASTPRIVATE, none of its constituent elements may be declared in another scope attribute. When individual members of a common block are privatized, the storage of the specified variables is no longer associated with the storage of the common block itself.
- Variables that are not allowed in the PRIVATE and SAHARED clauses are not affected by the DEFAULT(PRIVATE) or DEFAULT(SHAREDO clauses.
- Clauses can be repeated as needed, but each variable can appear explicitly in only one clause per directive, with the following exceptions: (1) a variable can be specified as both FIRSTPRIVATE and LASTPRIVATE; (2) Variables affected by the DEFAULT clause can be listed explicitly in a clause to override the default specification.

Directive binding

An OpenMP Fortran program must adhere to the following rules with respect to directive binding:

- The DO, SECTIONS, SINGLE, MASTER, and BARRIER directives bind to the dynamically enclosing PARALLEL, if one exists.
- The ORDERED directive binds to the dynamically enclosing DO.
- ATOMIC and CRICTICAL directives enforce access with respect to all threads, not just the current team.
- A directive can never bind to any directive outside the closest enclosing PARALLEL.

Directive Nesting

An OpenMP Fortran program must adhere to the following rules with respect to the dynamic nesting of directives:

- A PARALLEL directive dynamically inside another PARALLEL directive logically establishes a new team, which is composed of only the current thread unless nested parallelism is enabled.
- DO, SECTIONS, and SINGLE directives that bind to the same PARALLEL directive are not allowed to be nested one inside the other. Furthermore, these directives are not allowed in the dynamic extent of CRTICAL and MASTER directives.
- BARRIER directives are not permitted in the dynamic extent of DO, SECTIONS, SINGLE, MASTER and CRITICAL directives
- MASTER directives are not permitted in the dynamic extent of DO, SECTIONS, and SIJNGLE, directives.
- ORDERED sections are not allowed in the dynamic extent of CRITICAL sections.
- Any directive set that is legal when executed dynamically inside a PARALLEL region is also legal when executed outside a parallel region. When executed dynamically outside a user-specified parallel region, the directive is executed with respect to a team composed of only the master thread.

Runtime Library Functions

In the description of these routines, scalar_integer_expr is a default scalar integer expression, scalar_logical_expr is a default scalar logical expression, and var is of type integer and a KIND large enough to hold an address.

Execution environment functions

```
SUBROUTINE OMP_SET_NUM_THREADS(scalar_integer_expr)
INTEGER FUNCTION OMP_GET_NUM_THREADS()
INTEGER FUNCTION OMP_GET_MAX_THREADS()
INTEGER FUNCTION OMP_GET_THREAD_NUM()
INTEGER FUNCTION OMP_GET_NUM_PROCS()
LOGICAL FUNCTION OMP_IN_PARALLEL()
SUBROUTINE OMP_SET_SYNAMIC(scalar_logical_expr)
LOGICAL FUNCTION OMP_GET_DYNAMIC()
SUBROUTINE OMP_SET_NESTED (scalar_logical_expr)
LOGICAL FUNCTION OMP_GET_NESTED()
```

Lock functions

```
SUBROUTINE OMP_INIT_LOCK (var)
SUBROUTINE OMP_DESTROY_LOCK(var)
SUBROUTINE OMP_SET_LOCK(var)
SUBROUTINE OMP_UNSET_LOCK(var)
LOGICAL FUNCTION OMP_TEST_LOCK (var)
```

Environment Variables

```
OMP_SCHEDULE "schedule[, chunk_size]"
OMP_NUM_THREADS int
OMP_DYNAMIC TRUE || FALSE
OMP_NESTED TRUE || FALSE
```