OpenMP 5.0 API シンタックス・クイック・リファレンス・カード

OpenMP API は、スケーラブルでポータブルな並列処理を実現します。ポータブルな C/C++ および Fortran 並列アプリケーションを開発するためのシンプルで柔軟性のあるインターフェイスです。

OpenMP はマルチコアノードとマルチチップ、NUMA システム、GPU、および CPU に接続されたデバイスで実行される多様なアルゴリズムに適しています。

C/C++ C/C++ 向け　For Fortran 向け

OpenMP 5.0 で追加/変更された機能はこの色で示し、OpenMP 4.5 の機能はこの色で示します。

[n.n.n] は 5.0 のセクションを示し、 **[n.n.n]** は 4.5 を示します。● は、5.0 では非標準となりました。

ディレクティブと構造

OpenMP 実行ディレクティブは、後続の構造化ブロックに適用されます。構造化ブロックは、OpenMP 構造、または単一の入口と出口を持つ実行文のブロックです。**simd** と宣言ディレクティブを除く OpenMP ディレクティブは、**PURE** または **ELEMENTAL** プロシージャーには現れません。

異形ディレクティブ

metadirective [2.3.4]
指定する OpenMP コンテキストを基に metadirective を置き換えるため、条件付きで選択できる複数のディレクティブ・バリアントを指定するディレクティブです。

C/C++
```
#pragma omp metadirective [節[ [, ]節] ...]
      または
#pragma omp begin metadirective [節[ [, ]節] ...]
      文
#pragma omp end metadirective
```

For
```
!$omp metadirective [節[ [, ]節] ...]
      または
!$omp begin metadirective [節[ [, ]節] ...]
      文
!$omp end metadirective
```

節:
when(*context-selector-specification* :[*declare-variant*])
default(*declare-variant*)

declare variant [2.3.5]
ベース関数の特殊バリアントとそれらが使用されるコンテキストを宣言します。

C/C++
```
#pragma omp declare variant(variant-func-id) 節
[#pragma omp declare variant(variant-func-id) 節]
   […]
   関数定義または宣言
```

For
```
!$omp declare variant ( &
    [base-proc-name:] variant-proc-name) 節
```

節:
match(*context-selector-specification*)

variant-func-id: C/C++
ベース言語の識別子、または C++ では *template-id* である関数バリアント名。
variant-proc-name: For
ベース言語の識別子である関数バリアント名。

要求ディレクティブ

requires [2.4]
コードをコンパイルして正しく実行するために実装が提供しなければならない機能を指定します。

C/C++
```
#pragma omp requires 節 [[[,] 節] ...]
```

For
```
!$omp requires節 [[[,] 節] ...]
```

節:
reverse_offload
unified_address
unified_shared_memory
atomic_default_mem_order(&
seq_cst | acq_rel | relaxed)
dynamic_allocators

並列構造

parallel [2.6] [2.5]
並列領域を実行する OpenMP スレッドのチームを形成します。

C/C++
```
#pragma omp parallel [節 [[,] 節] ...]
   構造化ブロック
```

For
```
!$omp parallel [節 [[,] 節] ...]
   構造化ブロック
!$omp end parallel
```

節:
private(*list*)、**firstprivate**(*list*)、**shared**(*list*)
copyin(**list**)
reduction([*reduction-modifier,*] *reduction-identifier: list*)
proc_bind(**master** | **close** | **spread**)
allocate ([*allocator:*] *list*)
C/C++ **if** ([**parallel**:] *scalar-expression*)
C/C++ **num_threads** (*integer-expression*)
C/C++ **default** (**shared** | **none**)
For **if** ([**parallel**:] *scalar-logical-expression*)
For **num_threads** (*scalar-integer-expression*)
For **default** (**shared** | **firstprivate** | **private** | **none**)

チームス構造

teams [2.7] [2.10.7]
各チームのマスタースレッドが領域を実行する、スレッドのリーグを作成します。

C/C++
```
#pragma omp teams [節 [[,] 節] ...]
   構造化ブロック
```

For
```
!$omp teams [節 [[,] 節] ...]
   構造化ブロック
!$omp end teams
```

節:
private(*list*)、**firstprivate**(*list*)、**shared**(*list*)
reduction([**default**,] *reduction-identifier: list*)
allocate ([*allocator:*] *list*)
C/C++ **num_teams**(*integer-expression*)
C/C++ **thread_limit**(*integer-expression*)
C/C++ **default**(**shared** | **none**)
For **num_teams**(*scalar-integer-expression*)
For **thread_limit**(*scalar-integer-expression*)
For **default** (**shared** | **firstprivate** | **private** | **none**)

ワークシェア構造

sections [2.8.1] [2.7.2]
スレッドのチームに分散され実行される一連の構造化ブロックを含む非反復型のワークシェア構造です。

C/C++
```
#pragma omp sections [節[ [, ] 節] ...]
    {
    [#pragma omp section]
       構造化ブロック
    [#pragma omp section
       構造化ブロック] …
    }
```

For
```
!$omp sections [節[ [, ] 節] ...]
    {
    [!$omp section]
       構造化ブロック
    [!$omp section
       構造化ブロック] …
!$omp end sections [nowait]
```

節:
private(*list*)、**firstprivate**(*list*)
lastprivate([*lastprivate-modifier:*] *list*)
reduction([*reduction-modifier,*] *reduction-identifier: list*)
allocate ([*allocator:*] *list*)
C/C++ **nowait**

single [2.8.2] [2.7.3]
指定する構造化ブロックは、スレッドチームの 1 つのスレッドでのみ実行されます。

C/C++
```
#pragma omp single [節[ [, ]節] ...]
   構造化ブロック
```

For
```
!$omp single [節[ [, ]節] ...]
   構造化ブロック
!$omp end single [end_節 [[,] end_節] …]
```

節:
private(*list*)、**firstprivate**(*list*)
allocate ([*allocator:*] *list*)
C/C++ **copyprivate**(*list*)
C/C++ **nowait**
end_節: For
copyprivate(*list*)、**nowait**

workshare [2.8.3] [2.7.4]
構造化ブロックの実行を異なるワーク単位に分割し、それぞれが 1 つのスレッドによって一度だけ実行されます。

For
```
!$omp workshare
   構造化ブロック
!$omp end workshare [nowait]
```

ワークシェア・ループ構造

for / do [2.9.2] [2.7.1]
関連するループ反復がチーム内のスレッドによって並列実行されることを指定します。

C/C++
```
#pragma omp for [節[ [, ]節] ...]
   for ループ
```

For
```
!$omp do [節[ [, ]節] ...]
   do ループ
[!$omp end do [nowait]]
```

節:
private(*list*) 、**firstprivate**(*list*)
lastprivate([*lastprivate-modifier:*] *list*)
linear(*list* [:*linear-step*])
schedule([*modifier* [,*modifier*]:] *kind*[,*chunk_size*])
collapse(*n*)、ordered[(*n*)]
allocate([*allocator:*] *list*)
order(concurrent)
reduction([*reduction-modifier,*] *reduction-idenrifier:list*)
C/C++ **nowait**

kind:
- **static:** 反復は同一の *chunk_size* に分割され、そのチャンクはラウンドロビン方式 (総当り) でスレッド番号の順番にチームのスレッドへ振り分けられます。

ディレクティブと構造 (続き)

- **dynamic:** スレッドは反復チャンクを実行して、実行が終わったら、次のチャンクを要求します。チャンクがなくなるまで、これを繰り返します。
- **guided:** スレッドは反復チャンクを実行して、実行が終わったら、次のチャンクを要求します。割り当てるチャンクがなくなるまで、これを繰り返します。チャンクサイズはチャンクごとに異なり、後になるほど小さくなります。
- **auto:** スケジュールの決定権は、コンパイラーやランタイムに任せられています(実装依存)。
- **runtime:** スケジュール・タイプとチャンクサイズは、内部制御変数 ***run-sched-var*** から取得されます。

modifier:

- **monotonic:** 各スレッドは、論理的に割り当てられた昇順のチャンクを実行します。**static** スケジュールのデフォルトです。
- **nonmonotonic:** チャンクはスレッドに任意の順番で割り当てられ、チャンクの実行順序に依存するアプリケーションの動作は未定義です。 **static** を除くすべてのスケジュール kind のデフォルトです。
- **simd:** ループが SIMD 構造に関連付けられていない場合は無視されますが、それ以外では最初と最後のチャンクを除き *new_chunk_size* は、[*chunk_size/simd_width*] * *simd_width* となります (ここで、*simd_width* は実装依存です)。

SIMD ディレクティブ

simd [2.9.3.1] [2.8.1]

SIMD ループに変換可能なループであることを明示するため、ループに適用します。

C/C++
```
#pragma omp simd [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp simd [節[ [, ]節] ...]
  do ループ
[!$omp end simd]
```

節:

safelen(*length*)、simdlen(*length*)
linear(*list [:linear-step]*)
aligned(*list [:alignment]*)
nontemporal(*list*)
private(*list*)
lastprivate(*[lastprivate-modifier:] list*)
reduction(*[reduction-modifier,]* *reduction-idenrifier:list*)
collapse(*n*)
C/C++ **if**(*[simd:] scalar-expression*)
For **if**(*[simd:] scalar-logical-expression*)

ワークシェア・ループ simd [2.9.3.2] [2.8.3]

スレッドのチームで並列に実行できるSIMD ループに変換可能なループであることを明示するためループに適用します。

C/C++
```
#pragma omp for simd [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp do simd [節[ [, ]節] ...]
  do ループ
[!$omp end do simd[nowait]]
```

節: **simd** もしくは **for/do** ディレクティブと同一の節と制限が適用されます。

declare simd [2.9.3.3] [2.8.2]

SIMD ループから呼び出される関数が、SIMD 命令を使用する複数のバージョンを生成することを有効にすることを指示します。

C/C++
```
#pragma omp declare simd [節[ [, ]節] ...]
[#pragma omp declare simd [節[ [, ]節] ...]]
  関数定義または宣言
```

For
```
!$omp declare simd [(proc-name)][節[ [, ]節]
  ...]
```

節:

simdlen(*length*)
linear(*list [:linear-step]*)
aligned(*argument-list [:alignment]*)
uniform(*argument-list*)
inbranch
notinbranch

ディストリビュート・ループ構造

distribute [2.9.4.1] [2.10.8]

スレッドチームで実行されるループを指定します。

C/C++
```
#pragma omp distribute [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp distribute [節[ [, ]節] ...]
  do ループ
[!$omp end distribute]
```

節:

private(*list*)
firstprivate(*list*)
lastprivate(*list*)
collapse(*n*)
dist_schedule(*kind[, chunk_size]*)
allocate(*[allocator :] list*)

distribute simd [2.9.4.2] [2.10.9]

SIMD 命令を使用してスレッドチームで同時に実行されるループを指定します。

C/C++
```
#pragma omp distribute simd [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp distribute simd [節[ [, ]節] ...]
  do ループ
[!$omp end distribute simd]
```

節: **distribute** や **simd** ディレクティブと同一の節と制限が適用されます。

distribute parallel ワークシェア・ループ [2.9.4.3] [2.10.10]

複数のチームのメンバーである複数のスレッドによって同時に実行できるループを指定します。

C/C++
```
#pragma omp distribute parallel for ¥
                          [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp distribute parallel do [節[ [, ]節] ...]
  do ループ
[!$omp end distribute parallel do]
```

節: **distribute** や parallel ワークシェア・ループ・ディレクティブと同一の節と制限が適用されます。

distribute parallel ワークシェア・ループ simd [2.9.4.4] [2.10.11]

SIMD 命令を使用して複数のチームのメンバーである複数のスレッドで同時に実行できるループを指定します。

C/C++
```
#pragma omp distribute parallel for simd ¥
                          [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp distribute parallel do simd [節[ [, ]節]
  ...]
  do ループ
[!$omp end distribute parallel do simd]
```

節: **distribute** や parallel ワークシェア・ループ simd ディレクティブと同一の節と制限が適用されます。

ループ構造

loop [2.9.5]

複数のスレッドで関連するループ反復を並列に実行できることを示します。

C/C++
```
#pragma omp loop [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp loop [節[ [, ]節] ...]
  do ループ
[!$omp end loop]
```

節:

bind(*binding*)
collapse(*n*)
order(**concurrent**)
private(*list*)、**lastprivate**(*list*)
reduction(*[**default**,] reduction-idenrifier:list*)

binding:

teams、**parallel**、**thread**

スキャン・ディレクティブ

scan [2.9.6]

スキャン計算が各反復でリスト項目を更新することを指定します。

C/C++
```
ループに関連するディレクティブ
for ループ {
  構造化ブロック
  #pragma omp scan 節
  構造化ブロック
}
```

For
```
ループに関連するディレクティブ
do ループ
  構造化ブロック
  !$omp scan 節
  構造化ブロック
end do 文
[end ループに関連するディレクティブ]
```

節:

inclusive(*list*)、**exclusive**(*list*)

ループに関連するディレクティブ: C/C++
for、**for simd**、**simd** ディレクティブ

[end]ループに関連するディレクティブ: For
do(**end do**)
do simd(**end do simd**)
simd(**end simd**)

タスク構造

task [2.10.1] [2.9.1]

明示的にタスクを定義します。タスクのデータ環境は、**task** 構造のデータ共有属性節と適用されるデフォルトに従って作成されます。

C/C++
```
#pragma omp task [節[ [, ]節] ...]
  構造化ブロック
```

For
```
!$omp task [節[ [, ]節] ...]
構造化ブロック
!$omp end task
```

節:

untied、**mergeable**
private(*list*)、**firstprivate**(*list*)、**shared**(*list*)
in_reduction(*reduction-identifier: list*)
depend(*[depend-modifier,] dependence-type: locator-list*)
priority(*priority-value*)
allocate(*[allocator:] list*)
affinity(*[aff-modifier:] locator-list*)
- *aff-modifier* は **iterator**(*iterator-definition*)

detach(*event-handle*)
- *event-handle* は、
  omp_event_handle タイプ C/C++
  kind **omp_event_handle_kind** For

C/C++ **default**(**shared** | **none**)
C/C++ **if**([**task**:] *scalar-expression*)
C/C++ **final**(*scalar-expression*)
For **default**(**private** | **firstprivate** | **shared** | **none**)
For **if**([**task**:] *scalar-logical-expression*)
For **final**(*scalar-logical-expression*)

taskloop [2.10.2] [2.9.2]

1 つ以上の関連するループ反復を、OpenMP タスクを使用して並列に実行します。

C/C++
```
#pragma omp taskloop [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp taskloop [節[ [, ]節] ...]
  do ループ
[!$omp end taskloop]
```

ディレクティブと構造 (続き)

節:
shared(*list*)、**private**(*list*)
firstprivate(*list*)、**lastprivate**(*list*)
reduction([***default***,][*reduction-identifier: list*)]
in_reduction(*reduction-identifier: list*)
grainsize(*grain-size*)、**num_tasks**(*num_tasks*)
collapse(*n*)、**priority**(*priority-value*)
untied、**mergeable**、**nogroup**
allocate([*allocator:*] *list*)
C/C++ **if**([**taskloop**:] *scalar-expression*)
C/C++ **default**(**shared** | **none**)
C/C++ **final**(*scalar-expression*)
For **if**([**taskloop**:] *scalar-logical-expression*)
For **default**(**private** | **firstprivate** | **shared** | **none**)
For **final**(*scalar-logical-expression*)

taskloop simd [2.10.3] [2.9.3]

ループが SIMD 命令を使用して同時に実行可能であり、またその反復は OpenMP タスクを使用して並列時実行できることを示します。

C/C++
```
#pragma omp taskloop simd [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp taskloop simd [節[ [, ]節] ...]
  do ループ
[!$omp end taskloop simd]
```

節: **simd** もしくは **taskloop** ディレクティブと同一の節と制限が適用されます。

taskyield [2.10.4] [2.11.2]

現在のタスクを中断し、別のタスクの実行を優先することを許可します。

C/C++
```
#pragma omp taskyield
```

For
```
!$omp taskyield
```

メモリー管理ディレクティブ

メモリー空間 [2.11.1]

定義済みメモリー空間 **[表 2.8]** は、変数の格納と検索向けのストレージリソースを表します。

メモリー空間	ストレージ用途
omp_default_mem_space	システムのデフォルトストレージ
omp_large_cap_mem_space	大容量ストレージ
omp_const_mem_space	定数値の変数用に最適化されたストレージ
omp_high_bw_mem_space	高帯域幅のストレージ
omp_low_lat_mem_space	低レイテンシーのストレージ

allocate [2.11.3]

変数の割り当て方法を指定します。

C/C++
```
#pragma omp allocate (list) [節]
```

For
```
!$omp allocate (list) [節]
  または
!$omp allocate [(list)] 節
[!$omp allocate (list) 節 [···]]
  割り当て文
```

節:
allocator (*allocator*)
- *allocator* は次の表現:
C/C++ *omp_allocator_handle_t* 型
For *kind* **omp_allocator_handle_kind**

デバイス・ディレクティブと構造

target data [2.12.2] [2.10.1]

領域範囲のデバイスデータ環境を作成します。

C/C++
```
#pragma omp target data 節 [[[,] 節]···]
  構造化ブロック
```

For
```
!$omp target data節 [[[,] 節]···]
  構造化ブロック
!$omp end target data
```

節:
map([*[map-type-modifier[,] [map-type-modifier[,] ...] map-type*:] *locator-list*)
use_device_ptr(*list*)、**use_device_addr**(*list*)
C/C++ **if**([**target data**:] *scalar-expression*)
C/C++ **device**(*scalar-expression*)
For **if**([**target data**:] *scalar-logical-expression*)
For **device**(*scalar-integer-expression*)

target enter data [2.12.3] [2.10.2]

変数をデバイスのデータ環境にマップします。

C/C++
```
#pragma omp target enter data [節[ [, ]節] ...]
```

For
```
!$omp target enter data [節[ [, ]節] ...]
```

節:
map([*map-type-modifier[,] [map-type-modifier[,] ...] map-type*: *locator-list*)
depend([*depend-modifier,*] *dependence-type: locator-list*)
nowait
C/C++ **if**([**target enter data**:] *scalar-expression*)
C/C++ **device**(*integer-expression*)
For **if**([**target enter data**:] *scalar-logical-expression*)
For **device**(*scalar-integer-expression*)

target exit data [2.12.4] [2.10.3]

変数をデバイスのデータ環境からアンマップします。

C/C++
```
#pragma omp target exit data [節[ [, ]節] ...]
```

For
```
!$omp target exit data [節[ [, ]節] ...]
```

節:
map([*map-type-modifier[,] [map-type-modifier[,] ...] map-type*: *locator-list*)
depend([*depend-modifier,*] *dependence-type: locator-list*)
nowait
C/C++ **if**([**target exit data**:] *scalar-expression*)
C/C++ **device**(*integer-expression*)
For **if**([**target exit data**:] *scalar-logical-expression*)
For **device**(*scalar-integer-expression*)

target [2.12.5] [2.10.4]

デバイスのデータ環境に変数をマップし、デバイス上で構造を実行します。

C/C++
```
#pragma omp target [節[ [, ]節] ...]
  構造化ブロック
```

For
```
!$omp target [節[ [, ]節] ...]
  構造化ブロック
!$omp end target
```

節:
private (*list*), firstprivate (*list*)
in_reduction (*reduction-identifier* : *list*)
map (*[[map-type-modifier[,]*
[map-type-modifier[,] ...] map-type:] *locator-list*)
is_device_ptr (*list*)
defaultmap (*implicit-behavior[:variable-category]*)
nowait
depend([*depend-modifier,] dependence-type* : *locator-list*)
allocate ([*allocator* :] *list*)
uses_allocators (*allocator[* (*allocator-traits-array*)*]*
[,allocator[(*allocator-traits-array*) *] ...]*)
C/C++ **if** ([**target** :] *scalar-expression*)
C/C++ **device**([*device-modifier:*] *integer-expression*)
For **if** ([**target** :] *scalar-logical-expression*)
For **device** ([*device-modifier:*] *scalar-integer-expression*)

device-modifier: **ancestor**、**device_num**
allocator: C/C++
omp_allocator_handle_t 型の識別子
allocator: For
kind **omp_allocator_handle_kind** の整数式
allocator-traits-array: C/C++
const omp_alloctrait_t * 型の識別子
allocator-traits-array: For
type(omp_alloctrait) 型の配列

target update [2.12.6] [2.10.5]

モーション節で指定される、元のリスト項目と一致するデバイスデータ環境のリスト項目を作成します。

C/C++
```
#pragma omp target update [節[ [, ]節] ...]
```

For
```
!$omp target update [節[ [, ]節] ...]
```

節: モーション節または次のいずれか:
nowait
depend ([*depend-modifier,] dependence-type* : *locator-list*)
C/C++ **if** ([**target update** :] *scalar-expression*)
C/C++ **device** (*integer-expression*)
For **if** ([**target update** :] *scalar-logical-expression*)
For **device** (*scalar-integer-expression*)

モーション節:
to ([**mapper**(*mapper-identifier*) :] *locator-list*)
from ([**mapper**(*mapper-identifier*) :] *locator-list*)

declare target [2.12.7] [2.10.6]

デバイスにマップされる変数、関数およびサブルーチンを指定します。

C/C++
```
#pragma omp declare target
  宣言定義シーケンス
#pragma omp end declare target
または
#pragma omp declare target (extended-list)
または
#pragma omp declare target 節[ [, ]節 ...]
```

For
```
!$omp declare target (extended-list)
または
!$omp declare target 節[ [, ]節 ...]
```

節:
to (*extended-list*), **link** (*list*)
device_type (**host** | **nohost** | **any**)

extended-list: 名前付き変数、プロシージャー名、および名前付き共通ブロックをカンマで区切ったリスト。

結合構造

Parallel ワークシェア・ループ [2.13.1] [2.11.1]

1 つ以上の関連するループを含む 1 つのワークシェア・ループ構造を持つ **parallel** 構造を指定します。

C/C++
```
#pragma omp parallel for [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp parallel do [節[ [, ]節] ...]
  do ループ
!$omp end parallel do
```

節: **nowait** 節を除く **parallel** または **for** ディレクティブと同一の節と制限が適用されます。

parallel loop [2.13.2]

1 つ以上の関連するループの **loop** 構造だけを含み、その他の文を含まない **parallel** 構造を簡潔に指定します。

C/C++
```
#pragma omp parallel loop [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp parallel loop [節[ [, ]節] ...]
  do ループ
[!$omp end parallel loop]
```

節: **parallel** または **loop** ディレクティブと同一の節と制限が適用されます。

parallel sections [2.13.3] [2.11.2]

1 つの **sections** 構造だけを含み、その他の文を含まない **parallel** 構造を簡潔に指定します。

C/C++
```
#pragma omp parallel sections [節[ [, ]節] ...]
  {
  [#pragma omp section]
    構造化ブロック
  [#pragma omp section]
    構造化ブロック
  ...
  }
```

ディレクティブと構造 (続き)

For
```
!$omp parallel sections [節[ [, ]節] ...]
[!$omp section]
  構造化ブロック
[!$omp section]
  構造化ブロック
  …
!$omp end parallel sections
```

節: **parallel** もしくは **sections** ディレクティブと同一の節と制限が適用されます。

parallel workshare [2.13.4] [2.11.3]

1 つの **workshare** 構造だけを含み、その他の文を含まない **parallel** 構造を簡潔に指定します。

For
```
!$omp parallel workshare [節[ [, ]節] ...]
  構造化ブロック
!$omp end parallel workshare
```

節: **parallel** ディレクティブと同一の節と制限が適用されます。

parallel ワークシェア・ループ simd [2.13.5] [2.11.4]

1 つの **for/do simd** 構造だけを含み、その他の文を含まない **parallel** 構造を簡潔に指定します。

C/C++
```
#pragma omp parallel for simd [節[ [, ]節] ...]
  for ループ
```

For
```
!$ omp parallel do simd [節[ [, ]節] ...]
  do ループ
[!$ omp end parallel do simd]
```

節: **parallel** または **for/do simd** ディレクティブと同一の節と制限が適用されます。

parallel master [2.13.6]

1 つの **master** 構造だけを含み、その他の文を含まない **parallel** 構造を簡潔に指定します。

C/C++
```
#pragma omp parallel master [節[ [, ]節] ...]
  構造化ブロック
```

For
```
!$omp parallel master [節[ [, ]節] ...]
  構造化ブロック
!$omp end parallel master
```

節: **parallel** ディレクティブと同一の節と制限が適用されます。

master taskloop [2.13.7]

1 つの **taskloop** 構造だけを含み、その他の文を含まない **master** 構造を簡潔に指定します。

C/C++
```
#pragma omp master taskloop [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp master taskloop [節[ [, ]節] ...]
  do ループ
[!$omp end master taskloop]
```

節: **taskloop** ディレクティブと同一の節と制限が適用されます。

master taskloop simd [2.13.8]

1 つの **taskloop simd** 構造だけを含み、その他の文を含まない **master** 構造を簡潔に指定します。

C/C++
```
#pragma omp master taskloop simd ¥
    [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp master taskloop simd [節[ [, ]節] ...]
  do ループ
[!$omp end master taskloop simd]
```

節: **taskloop simd** ディレクティブと同一の節と制限が適用されます。

parallel master taskloop [2.13.9]

1 つの **master taskloop** 構造だけを含み、その他の文を含まない **parallel** 構造を簡潔に指定します。

C/C++
```
#pragma omp parallel master taskloop ¥
    [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp parallel master taskloop [節[ [, ]節] ...]
  do ループ
[!$omp end parallel master taskloop]
```

節: **in_reduction** 節を除く **parallel** または **master taskloop** ディレクティブと同一の節と制限が適用されます。

parallel master taskloop simd [2.13.10]

1 つの **master taskloop simd** 構造だけを含み、その他の文を含まない **parallel** 構造を簡潔に指定します。

C/C++
```
#pragma omp parallel master taskloop simd ¥
    [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp parallel master taskloop simd
    [節[ [, ]節] ...]
  do ループ
[!$omp end parallel master taskloop simd]
```

節: **in_reduction** 節を除く **parallel** または **master taskloop simd** ディレクティブと同一の節と制限が適用されます。

teams distribute [2.13.11] [2.11.10]

1 つの **distribute** 構造だけを含み、その他の文を含まない **teams** 構造を簡潔に指定します。

C/C++
```
#pragma omp teams distribute [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp teams distribute [節[ [, ]節] ...]
  do ループ
[!$omp end teams distribute ]
```

節: **teams** または **distribute** ディレクティブと同一の節と制限が適用されます。

teams distribute simd [2.13.12] [2.11.11]

1 つの **distribute simd** 構造だけを含み、その他の文を含まない **teams** 構造を簡潔に指定します。

C/C++
```
#pragma omp teams distribute simd ¥
    [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp teams distribute simd [節[ [, ]節] ...]
  do ループ
[!$omp end teams distribute simd ]
```

節: **teams** または **distribute simd** ディレクティブと同一の節と制限が適用されます。

teams distribute parallel ワークシェア・ループ [2.13.13] [2.11.14]

1 つの distribute parallel ワークシェア・ループ構造だけを含み、その他の文を含まない **teams** 構造を簡潔に指定します。

C/C++
```
#pragma omp teams distribute parallel for ¥
    [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp teams distribute parallel do
    [節[ [, ]節] ...]
  do ループ
[!$omp end teams distribute parallel do]
```

節: **teams** または **distribute parallel do/for** ディレクティブと同一の節と制限が適用されます。

teams distribute parallel ワークシェア・ループ simd [2.13.14] [2.11.7]

1 つの distribute parallel ワークシェア・ループ simd 構造だけを含み、その他の文を含まない **teams** 構造を簡潔に指定します。

C/C++
```
#pragma omp teams distribute parallel for ¥
    simd [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp teams distribute parallel do simd
    [節[ [, ]節] ...]
  do ループ
[!$omp end teams distribute parallel do simd]
```

節: **teams** または **distribute parallel do/for simd** ディレクティブと同一の節と制限が適用されます。

teams loop [2.13.15]

1 つの **loop** 構造だけを含み、その他の文を含まない **teams** 構造を簡潔に指定します。

C/C++
```
#pragma omp teams loop [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp teams loop [節[ [, ]節] ...]
  do ループ
[!$omp end teams loop]
```

節: **teams** または **loop** ディレクティブと同一の節と制限が適用されます。

target parallel [2.13.16] [2.11.5]

1 つの **parallel** 構造だけを含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target parallel [節[ [, ]節] ...]
  構造化ブロック
```

For
```
!$omp target parallel [節[ [, ]節] ...]
  構造化ブロック
[!$omp end target parallel]
```

節: **copyin** を除く **target** または **parallel** ディレクティブと同一の節と制限が適用されます。

target parallel ワークシェア・ループ [2.13.17] [2.11.6]

1 つの parallel ワークシェア・ループ 構造だけを含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target parallel for [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp target parallel do [節[ [, ]節] ...]
  do ループ
[!$omp end target parallel do]
```

節: **copyin** を除く **target** または **parallel for/do**ディレクティブと同一の節と制限が適用されます。

target parallel ワークシェア・ループ simd [2.13.18] [2.11.7]

1 つの parallel ワークシェア・ループ simd 構造だけを含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target parallel for simd ¥
    [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp target parallel do simd [節[ [, ]節] ...]
  do ループ
[!$omp end target parallel do simd]
```

節: **target** または **parallel for/do simd** ディレクティブと同一の節と制限が適用されます。

target parallel loop [2.13.19]

1 つの **parallel loop** 構造だけを含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target parallel loop ¥
    [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp target parallel loop [節[ [, ]節] ...]
  do ループ
[!$omp end target parallel loop]
```

節: **target** または **parallel loop** ディレクティブと同一の節と制限が適用されます。

target simd [2.13.20] [2.11.8]

1 つの **simd** 構造だけを含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target simd [節[ [, ]節] ...]
  for ループ
```

For
```
!$omp target simd [節[ [, ]節] ...]
  do ループ
[!$omp end target simd]
```

節: **target** または **simd** ディレクティブと同一の節と制限が適用されます。

target teams [2.13.21] [2.11.9]

1 つの **teams** 構造だけを含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target teams [節[ [, ]節] ...]
  構造化ブロック
```

For
```
!$omp target teams [節[ [, ]節] ...]
  構造化ブロック
[!$omp end target teams]
```

節: **target** または **teams** ディレクティブと同一の節と制限が適用されます。

ディレクティブと構造 (続き)

target teams distribute [2.13.22] [2.11.12]

1 つの **teams distribute** 構造だけを含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target teams distribute ¥
      [節[ [, ]節] ...]
   for ループ
```

For
```
!$omp target teams distribute [節[ [, ]節] ...]
   do ループ
[!$omp end target teams distribute]
```

節: **target** または **teams distribute** ディレクティブと同一の節と制限が適用されます。

target teams distribute simd [2.13.22] [2.11.13]

1 つの **teams distribute simd** 構造だけを含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target teams distribute simd ¥
      [節[ [, ]節] ...]
   for ループ
```

For
```
!$omp target teams distribute simd
      [節[ [, ]節] ...]
   do ループ
[!$omp end target teams distribute simd]
```

節: **target** または **teams distribute simd** ディレクティブと同一の節と制限が適用されます。

target teams loop [2.13.24]

1 つの **teams loop** 構造だけを含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target teams loop [節[ [, ]節] ...]
   for ループ
```

For
```
!$omp target teams loop [節[ [, ]節] ...]
   do ループ
[!$omp end target teams loop]
```

節: **target** または **teams loop** ディレクティブと同一の節と制限が適用されます。

target teams distribute parallel ワークシェア・ループ [2.13.25] [2.11.15]

teams distribute parallel ワークシェア・ループ構造を含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target teams distribute ¥
      parallel for [節[ [, ]節] ...]
   for ループ
```

For
```
!$omp target teams distribute parallel do &
      [節[ [, ]節] ...]
   do ループ
[!$omp end target teams distribute parallel do]
```

節: **teams distribute parallel for/do** または **target** ディレクティブと同一の節と制限が適用されます。

target teams distribute parallel ワークシェア・ループ simd [2.13.26] [2.11.17]

teams distribute parallel ワークシェア・ループ simd 構造を含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++
```
#pragma omp target teams distribute ¥
      parallel for simd [節[ [, ]節] ...]
   for ループ
```

For
```
!$omp target teams distribute parallel &
      do simd [節[ [, ]節] ...]
   do ループ
[!$omp end target teams distribute parallel &
  do simd]
```

節: **teams distribute parallel for/do simd** または **target** ディレクティブと同一の節と制限が適用されます。

マスター構造

master [2.16] [2.13.1]

チームのマスタースレッドで実行される構造化ブロックを指定します。

C/C++
```
#pragma omp master
   構造化ブロック
```

For
```
!$omp master
   構造化ブロック
!$omp end master
```

同期構造

critical [2.17.1] [2.13.2]

指定する構造化ブロックの実行を一度に 1 つのスレッドに制限します。

C/C++
```
#pragma omp critical [(name) [[,] ¥
      hint (hint 式)]]
   構造化ブロック
```

For
```
!$omp critical [(name) [[,] hint (hint 式)]]
   構造化ブロック
!$omp end critical [(name)]
```

hint 式: C/C++
有効な hint を評価する整数定数式

hint 式: For
kind **omp_sync_hint_kind** のスカラー値、および有効な同期 hint 値を評価する定数式

barrier [2.12.2] [2.13.3]

このディレクティブは、ベース言語の文が許可される位置にのみ配置でき、構造が記述された位置に明示的なバリアを指定します。

C/C++
```
#pragma omp barrier
```

For
```
!$omp barrierr
```

taskwait [2.17.5] [2.13.4]

現在のタスクの子タスクの完了を待機することを指示します。

C/C++
```
#pragma omp taskwait [節[ [,] 節] …]
```

For
```
!$omp taskwait [節[ [,] 節] …]
```

節:
depend (*[depend-modifier,] dependence-type* **:** *locator-list*)

taskgroup [2.17.6] [2.13.5]

現在のタスクの子タスクの完了を待機し、子孫タスクを待機することを指示します。

C/C++
```
#pragma omp taskgroup [節[ [,] 節] …]
   構造化ブロック
```

For
```
!$omp taskgroup [節[ [,] 節] …]
   構造化ブロック
!$omp end taskgroup
```

節:
task_reduction (*reduction-identifier* : *list*)
allocate (*[allocator:*]*list*)

atomic [2.17.7] [2.13.6]

特定のストレージの位置がアトミックにアクセスされることを保証します。次の 7 つの形式のいずれかです。

C/C++
```
#pragma omp atomic [節[[,] 節] ... ] [,]] ¥
      atomic 節 [[,] 節 [[[,] 節] ...]]
   式文
#pragma omp atomic [節[[,] 節] ...]
   式文
#pragma omp atomic [節[[[,] 節] ... ] [,]  ¥
      capture [[,] 節 [[[,] 節] ... ]]
   構造化ブロック
```

For
```
!$omp atomic [節[[[,] 節] ... ] [,]] read &
      [[,] 節 [[[,] 節] ...]]
   capture 文
[!$omp end atomic]

!$omp atomic [節[[[,] 節] ... ] [,]] write &
      [[,] 節 [[[,] 節] ...]]
   write 文
[!$omp end atomic]

!$omp atomic [節[[[,] 節] ... ] [,]] update &
      [[,] 節 [[[,] 節] ...]]
   update 文
[!$omp end atomic]

!$omp atomic [節[[,] 節] ...]
   update 文
[!$omp end atomic]

!$omp atomic [節[[[,] 節] ... ] [,]] capture &
      [[,] 節 [[[,] 節] ...]]
   update 文 capture 文
!$omp end atomic

!$omp atomic [節[[[,] 節] ... ] [,]] capture &
      [[,] 節 [[[,] 節] ...]]
   capture 文 update 文
!$omp end atomic

!$omp atomic [節[[[,] 節] ... ] [,]] capture &
      [[,] 節 [[[,] 節] ...]]
   capture 文 write 文
!$omp end atomic
```

atomic 節: **read**、**write**、**update**、**capture**
メモリーオーダー節: *seq_cst*、*acq_rel*、*release*、*acquire*、*relaxed*
節: メモリーオーダー節、または *hint (hint 式)*
式文: C/C++

節	式文
read	*v = x;*
write	*x = expr;*
update またはなし	*x++; x--; ++x; --x;* *x binop= expr; x = x binop expr;* *x = expr binop x;*
capture	*v=x++; v=x--; v=++x; v= --x;* *v=x binop= expr; v=x = x binop expr;* *v=x = expr binop x;*

構造化ブロックは次のいずれかの形式です: C/C++
{v = x; x binop= expr;}{x binop= expr; v = x;}
{v = x; x = x binop expr;}{v = x; x = expr binop x;}
{x = x binop expr; v = x;}{x = expr binop x; v = x;}
{v = x; x = expr;} {v = x; x++;} {v = x; ++x;}
{++x; v = x;} {x++; v = x;} {v = x; x--;}
{v = x; --x;} {--x; v = x;} {x--; v = x;}

capture 文、*write* 文、*update* 文: For
atomic 節が次の場合 …

capture または read	capture 文: v = x;
capture または write	Write 文: x = 式;
Capture または update、または指定なし	update 文: x = x operator 式 x = 式 operator x x = 組込み関数名(x, 式リスト) x =組込み関数名(式リスト, x)

組込み関数名は、**MAX**、**MIN**、**IAND**、**IOR**、**IEOR**
operator は、**+**、*、**-**、**/**、**.AND.**、**.OR.**、**.EQV.**、**.NEQV.**

ディレクティブと構造 (続き)

flush [2.17.8] [2.13.7]
スレッドのテンポラリー・メモリー・ビューとメモリーの一貫性を保ち、変数のメモリー操作の順番を強制します。

C/C++	**#pragma omp flush** *[メモリーオーダー節] [(list)]*
For	**!$omp flush** *[メモリーオーダー節] [(list)]*

メモリーオーダー節: *acq_rel*、*release*、*acquire*

ordered [2.17.9] [2.13.8]
ループの反復順に実行されるワークシェア・ループ、simd、またはワークシェア・ループ simd 領域内の構造化ブロックを指定します。

C/C++	**#pragma omp ordered** *[節[[,]節] ...]* 構造化ブロック または **#pragma omp ordered** *[節[[,]節] ...]*
For	**!$omp ordered** *[節[[,]節] ...]* 構造化ブロック **!$omp end ordered***]* または **!$omp ordered** *[節[[,]節] ...]*

節 (最初の形式): **threads** または **simd**
節 (2 番目の形式):
depend (source) または **depend (sink : vec)**

depobj [2.17.10.1]
OpenMP depend オブジェクトを初期化、更新、または破棄します。

C/C++	**#pragma omp depobj (***depobj***)** 節
For	**!$omp depobj (***depobj***)** 節

節*:*
depend (依存関係タイプ: *locator***)**
destroy
update (依存関係タイプ**)**

キャンセル構造

cancel [2.18.1] [2.14.1]
指定したタイプの最内領域のキャンセル要求を行います。

C/C++	**#pragma omp cancel** 構造タイプ節 *[[,]] if* 節
For	**!$omp cancel** 構造タイプ節 *[[,]] if* 節

C/C++ 構造タイプ節: **parallel**、**sections**、**taskgroup**、**for**
if 節: **if** (*[* **cancel** *:]* スカラー式)
For 構造タイプ節: **parallel**、**sections**、**taskgroup**、**do**
if 節: **if** (*[* **cancel** *:]* スカラー論理式)

cancellation point [2.18.2] [2.14.2]
タスクが、指定されたタイプの最内領域のキャンセルが要求されたかどうかをチェックするユーザー定義のキャンセルポイントを定義します。

C/C++	**#pragma omp cancellation point** 構造タイプ節
For	**!$omp cancellation point** 構造タイプ

構造タイプ節:
C/C++ **parallel**、**sections**、**taskgroup**、**for**
For **parallel**、**sections**、**taskgroup**、**do**

データ環境ディレクティブ

threadprivate [2.19.2] [2.15.2]
各スレッドが独自のコピーを持つように、変数を複製します。**threadprivate** 変数のそれぞれのコピーは、最初に参照される前に一度だけ初期化されます。

C/C++	**#pragma omp threadprivate (***list***)**
For	**!$omp threadprivate (***list***)**

list: C/C++
不完全な型を持たないファイルスコープ、名前空間スコープ、または静的ブロックスコープ変数のカンマ区切りリスト。
list: For
名前付き変数と名前付き共通ブロックのカンマ区切りリスト。共通ブロック名はスラッシュで囲まなければなりません。

declare reduction [2.19.5.7] [2.16]
reduction 節で使用できるリダクション演算子を宣言します。

C/C++	**#pragma omp declare reduction** ¥ (リダクション演算子**:**タイプ名リスト**:**コンバイナー) *[initializer* 節*]*
For	**!$omp declare reduction &** (リダクション演算子**:**タイプ名リスト**:**コンバイナー) *[initializer* 節*]*

C/C++
タイプ名リスト: 型指定子のリスト
initializer 節: **initializer** (*initializer* 式)
initializer 式は、**omp_priv** = 初期化子または関数名 (引数リスト)
リダクション識別子:
ベース言語の識別子 (C)、ID 式 (C++)、または次のいずれかの演算子: +、-、*、&、|、^、&&、||
コンバイナー: 式
For
タイプリスト: 型指定子のリスト
initializer 節: **initializer** (*initializer* 式)
initializer 式は、**omp_priv** = 初期化子または関数名 (引数リスト)
リダクション識別子**:** ベース言語の識別子、
ユー ザー定義オペレーター、または次のいずれかのオペレーター:
+、-、 *、.and.、.or.、.eqv.、.negv.、
または次のいずれかの組込みプロシージャー名:
max、min、iand、ior、ieor
コンバイナー: 代入文またはサブルーチン名とそれに続く引数リスト

declare mapper [2.19.7.3]
与えられたタイプに対するユーザー定義マッパーを宣言し、**map** 節で使用するマッパー識別子を定義します。

C/C++	**#pragma omp declare mapper** ¥ *([*マッパー識別子*:]* type var*) [*節*[[,]]* 節*] …]*
For	**!$omp declate mapper** *([*マッパー識別子*:]* **&** *type var) [*節*[[,]]* 節*] …]*

マッパー識別子: ベース言語の識別子または **default**

type: スコープ内で有効なタイプ
var: 有効なベース言語の識別子

節: **map** *([[*マップタイプ修飾子*[,] [*マップタイプ修飾子*[,] …]] マップタイプ:] list)*
マップタイプ:
alloc、**to**、**from**、**tofrom**
マップタイプ修飾子:
always、**close**

ランタイム・ライブラリー・ルーチン

実行環境ルーチン

omp_set_num_threads [3.2.1] [3.2.1]
現在のタスクの内部制御変数 *nthreads-var* の最初の要素の値を *num_threads* に設定することで、**num_threads** 節を持たないその後の parallel 領域で適用されるスレッド数に影響します。

C/C++	**void omp_set_num_threads** (int *num_threads*);
For	**subroutine omp_set_num_threads** (*num_threads*) **integer** *num_threads*

omp_get_num_threads [3.2.2] [3.2.2]
現在のチームのスレッド数を返します。
omp_get_num_threads 領域にバインドされる領域は、最も内側の **parallel** 領域です。シーケンシャル領域から呼び出された場合 1 を返します。

C/C++	**int omp_get_num_threads** (void);
For	**integer function omp_get_num_threads** ()

omp_get_max_threads [3.2.3] [3.2.3]
このルーチンからリターンした後に **num_threads** 節を持たない **parallel** 構造があった場合、新しいチームの形成に使用できる最大スレッド数を返します。

C/C++	**int omp_get_max_threads** (void);
For	**integer function omp_get_max_threads** ()

omp_get_thread_num [3.2.4] [3.2.4]
現在のチーム内の呼び出し元のスレッド数を返します。

C/C++	**int omp_get_thread_num**(void);
For	**integer function omp_get_thread_num** ()

omp_get_num_procs [3.2.5] [3.2.5]
ルーチンが呼び出されたときに、デバイスで利用可能なプロセッサー数を返します。

C/C++	**int omp_get_num_procs** (void);
For	**integer function omp_get_num_procs** ()

omp_in_parallel [3.2.6] [3.2.6]
内部制御変数 *active-levels-var* がゼロより大きい場合 *true* を返します。それ以外は *false* を返します。

C/C++	**int omp_in_parallel** (void);
For	**logical function omp_in_parallel** ()

ランタイム・ライブラリー・ルーチン (続き)

omp_set_dynamic [3.2.7] [3.2.7]
スレッド数の動的調整が有効か無効かを示す内部制御変数 *dyn-var* の値を設定します。

C/C++	**void omp_set_dynamic** (int *dynamic_threads*);
For	**subroutine omp_set_dynamic** (*dynamic_threads*) **logical** *dynamic_threads*

omp_get_dynamic [3.2.8] [3.2.8]
このルーチンは内部制御変数 *dyn-var* の値を返します。スレッド数の動的調整が現在のタスクに対し有効であれば値は *true* です。

C/C++	**int omp_get_dynamic** (void);
For	**logical function omp_get_dynamic** ()

omp_get_cancellation [3.2.9] [3.2.9]
内部制御変数 *cancel-var* の値を返します。キャンセルが有効である場合は *true*、そうでなければ *false* を返します。

C/C++	**int omp_get_cancellation** (void);
For	**logical function omp_get_cancellation** ()

● **omp_set_nested [3.2.10] [3.2.10]**
入れ子構造の並列処理を有効または無効にする、内部制御変数 *max-active-levels-var* を設定します。

C/C++	**void omp_set_nested** (int *nested*);
For	**subroutine omp_set_nested** (*nested*) **logical** *nested*

● **omp_get_nested [3.2.11] [3.2.11]**
内部制御変数 *max-active-levels-var* の値に応じて、入れ子構造の並列処理が有効または無効かを示す値を返します。

C/C++	**int omp_get_nested** (void);
For	**logical function omp_get_nested** ()

omp_set_schedule [3.2.12] [3.2.11]
runtime スケジュールを使用する場合、スケジュールに適用される内部制御変数 *run-sched-var* の値を設定します。

C/C++	**void omp_set_schedule(**omp_sched_t *kind*, int *chunk_size***);**
For	**subroutine omp_set_schedule (***kind*, *chunk_size***)** **integer (kind=omp_sched_kind)** *kind* **integer** *chunk_size*

omp_get_schedule の *kind* を参照。

omp_get_schedule [3.2.13] [3.2.13]
runtime スケジュールに適用される内部制御変数 *run-sched-var* の値を返します。

C/C++	**void omp_get_schedule(**omp_sched_t **kind*, Int **chunk_size***);**
For	**subroutine omp_get_schedule (***kind*, *chunk_size***)** **integer (kind=omp_sched_kind)** *kind* **integer** *chunk_size*

omp_set_schedule と **omp_get_schedule** の *kind* は、実装定義のスケジュールまたは *omp_sched_static*、*omp_sched_dynamic*、*omp_sched_guided*、*omp_sched_auto* のいずれかです。
+ や | 演算子 (C/C++) または + 演算子 (For) を使用して *omp_sched_monotonic* 修飾子と組み合わせることができます。

omp_get_thread_limit [3.2.14] [3.2.14]
利用可能な OpenMP スレッドの最大数を示す、内部制御変数 *thread-limit-var* の値を返します。

C/C++	**int omp_get_thread_limit** (void);
For	**integer function omp_get_thread_limit** ()

omp_get_supported_active_levels [3.2.15]
利用可能なアクティブな並列処理レベル数を返します。

C/C++	**int omp_get_supported_active_levels** (void);
For	**integer function** **omp_get_supported_avtive_levels** ()

omp_set_max_active_levels [3.2.16] [3.2.15]
入れ子構造のアクティブな並列領域の数を制限する内部制御変数 *max-active-levels-var* を設定します。

C/C++	**void omp_set_max_active_levels** (int max_levels);
For	**subroutine omp_set_nested** (*max_levels*) **logical** *nested*

omp_get_max_active_levels [3.2.17] [3.2.16]
入れ子構造のアクティブな並列領域の最大数を決定する内部制御変数 *max-active-levels-var* の値を返します。

C/C++	**int omp_get_max_active_levels** (void);
For	**integer function omp_get_nested** ()

omp_get_level [3.2.18] [3.2.17]
デバイス領域に対する、呼び出しを含むタスクを囲む入れ子構造の **parallel** 領域の数を示す内部制御変数 *levels-vars* の値を返します。

C/C++	**int omp_get_level** (void);
For	**integer function omp_get_level** ()

omp_get_ancestor_thread_num [3.2.19] [3.2.18]
現在のスレッドの入れ子レベルについて、現在のスレッドの先祖のスレッド番号を返します。

C/C++	**int omp_get_ancestor_thread_num** (int *levels*);
For	**integer function** **omp_get_ancestor_thread_bum** (*levels*) **integer** *levels*

omp_get_team_size [3.2.20] [3.2.19]
現在のスレッドの入れ子レベルについて、先祖もしくは現在のスレッドが属するスレッドチームのサイズを返します。

C/C++	**int omp_get_team_size** (int *level*);
For	**integer function omp_get_team_size** (*level*) **integer** *level*

omp_get_active_level [3.2.21] [3.2.20]
呼び出しを含むタスクを囲むアクティブな入れ子の **parallel** 領域の数を決定する内部制御変数 *active-level-vars* の値を返します。

C/C++	**int omp_get_active_level** (void);
For	**integer function omp_get_active_level** ()

omp_in_final [3.2.22] [3.2.21]
final タスク領域内で実行された場合、*true* を返します。そうでない場合は、*false* を返します。

C/C++	**int omp_in_final** (void);
For	**logical function omp_in_final** ()

omp_get_proc_bind [3.2.23] [3.2.22]
proc_bind 節を指定しない後続の入れ子になった **parallel** 領域に適用するスレッド・アフィニティー・ポリシーを返します。

C/C++	**omp_proc_bind_t omp_get_proc_bind** (void);
For	**integer (kind=omp_proc_bind_kind)** **function omp_get_proc_bind** ()

次のいずれかを返します:
omp_proc_bind_false
omp_proc_bind_true
omp_proc_bind_master
omp_proc_bind_close
omp_proc_bind_spread

omp_get_num_places [3.2.24] [3.2.23]
プレースリスト内の実行環境で使用可能なプレース数を返します。

C/C++	**int omp_get_num_places** (void);
For	**integer function omp_get_num_places** ()

omp_get_place_num_procs [3.2.25] [3.2.24]
指定された位置の実行環境で使用可能なプロセッサー数を返します。

C/C++	**int omp_get_place_num_procs** (int *place_num*);
For	**integer function** **omp_get_place_bum_procs** (*place_num*) **integer** *place_num*

omp_get_place_proc_ids [3.2.26] [3.2.25]
指定された位置の実行環境で使用可能なプロセッサーの数値識別子を返します。

C/C++	**int omp_get_place_proc_ids** (int *place_num*, int **ids*);
For	**subroutine omp_get_place_proc_ids** (*place_num, ids*) **integer** *place_num* **integer** *ids (*)*

omp_get_place_num [3.2.27] [3.2.26]
遭遇したスレッドがバインドされている位置のプレース番号を返します。

C/C++	**int omp_get_place_num** (void);
For	**integer function omp_get_place_num** ()

omp_get_partition_num_places [3.2.28] [3.2.27]
最も内側の暗黙のタスクのプレース・パーティション内のプレース数を返します。

C/C++	**int omp_get_partition_num_places** (void);
For	**integer function** **omp_get_partition_num_places** ()

ランタイム・ライブラリー・ルーチン (続き)

omp_get_partition_place_nums [3.2.29] [3.2.28]

最も内側の暗黙のタスクの内部制御変数 *place-partition-var* 内の位置に対応するプレース番号のリスト返します。

C/C++
```
void omp_get_partition_place_nums (
  int *place_nums);
```
For
```
subroutine omp_get_partition_place_nums (
  place_nums)
integer place_nums (*)
```

omp_set_affinity_format [3.2.30]

デバイスで使用されるアフィニティー形式を設定する内部制御変数 *affinity-format-var* の値を設定します。

C/C++
```
void omp_set_affinity_format (
  const char *format);
```
For
```
subroutine omp_set_affinity_format (format)
character(len=*), intent(in)::format
```

omp_get_affinity_format [3.2.31]

デバイスの内部制御変数 *affinity-format-var* の値を返します。

C/C++
```
size_t omp_get_affinity_format (
  char *buffer, size_t size);
```
For
```
integer function
  omp_get_affinity_format (buffer)
character(len=*), intent(in)::buffer
```

omp_display_affinity [3.2.32]

提供された書式で OpenMP スレッド・アフィニティー情報を表示します。

C/C++
```
void omp_display_affinity (const char *format);
```
For
```
subroutine omp_display_affinity (format)
character(len=*), intent(in)::format
```

omp_capture_affinity [3.2.33]

提供された書式で、OpenMP スレッド・アフィニティー情報をバッファーに出力します。

C/C++
```
size_t omp_capture_affinity (char *buffer,
  size_t size, const char* format);
```
For
```
integer function omp_capture_affinity (
  buffer, format)
character(len=*), intent(out)::buffer
character(len=*), intent(in)::format
```

omp_set_default_device [3.2.34] [3.2.29]

デフォルトのターゲットデバイスを決定する内部制御変数 *default-device-var* を設定します。

C/C++
```
void omp_set_default_device (int device_num);
```
For
```
subroutine omp_set_default_device (
  device_num)
integer device_num
```

omp_get_default_device [3.2.35] [3.2.30]

デフォルトのターゲットデバイスを示す、内部制御変数 *default-device-var* 値を返します。

C/C++
```
int omp_get_default_device (void);
```
For
```
integer function omp_get_default_device ()
```

omp_get_num_devices [3.2.36] [3.2.31]

ターゲットデバイス数を返します。

C/C++
```
int omp_get_num_devices (void);
```
For
```
integer function omp_get_num_devices ()
```

omp_get_device_num [3.2.37]

呼び出しスレッドが実行しているデバイスのデバイス番号を返します。

C/C++
```
int omp_get_device_num (void);
```
For
```
integer function omp_get_device_num ()
```

omp_get_num_teams [3.2.38] [3.2.32]

現在の **teams** 領域のチーム数を返します。**teams** 領域外から呼び出された場合 1 を返します。

C/C++
```
int omp_get_num_teams (void);
```
For
```
integer function omp_get_num_teams ()
```

omp_get_team_num [3.2.39] [3.2.33]

呼び出しスレッドのチーム番号を返します。チーム番号は、0 から **omp_get_num_teams** で返される値より 1 つ少ない整数値です。

C/C++
```
int omp_get_team_num (void);
```
For
```
integer function omp_get_team_num ()
```

omp_is_initial_device [3.2.40] [3.2.34]

現在のタスクがホストデバイス上で実行している場合 *true* を返します。それ以外は、*false* を返します。

C/C++
```
int omp_is_initial_device (void);
```
For
```
integer function omp_is_initial_device ()
```

omp_get_initial_device [3.2.41] [3.2.35]

ホストデバイスのデバイス番号を返します。

C/C++
```
int omp_get_initial_device (void);
```
For
```
integer function omp_get_initial_device ()
```

omp_get_max_task_priority [3.2.42] [3.2.36]

priority 節で指定できる最大値を返します。

C/C++
```
int omp_get_max_task_priority (void);
```
For
```
integer function omp_get_max_task_priority ()
```

omp_pause_resource [3.2.43] omp_pause_resource_all [3.2.44]

ランタイムが、指定されたデバイス上の OpenMP によって使用されるリソースを放棄することを許可します。

C/C++
```
int omp_pause_resource (
  omp_pause_resource_t kind, int device_num);

int omp_pause_resource_all (
  omp_pause_resource_t kind);
```
For
```
integer function omp_pause_resource (
  kind, device_num)
integer (kind=omp_pause_resource_kind) kind
integer device_num

integer function omp_pause_resource_all (kind)
integer (kind=omp_pause_resource_kind) kind
```

ロックルーチン

汎用ロックルーチン。2 つのタイプのロックがサポートされます: シンプルなロックと入れ子可能なロック。入れ子可能なロックは、解除 (unset) される前に同じタスクで複数回設定 (set) することができます。シンプルなロックは、設定しようとするタスクですでに所有されている場合、設定することはできません。

ロックの初期化 [3.3.1] [3.3.1]

OpenMP ロックを初期化します。

C/C++
```
void omp_init_lock (omp_lock_t *lock);

void omp_init_nest_lock (omp_nest_lock_t
                         *lock);
```
For
```
subroutine omp_init_lock (svar)
integer (kind=omp_lock_kind) svar

subroutine omp_init_nest_lock (nvar)
integer (kind=omp_nest_lock_kind) nvar
```

ヒント付きのロックの初期化 [3.3.2] [3.3.2]

ヒント付きで OpenMP ロックを初期化します。

C/C++
```
void omp_init_lock_with_hint (
  omp_lock_t *lock,
  omp_sync_hint_t hint);

void omp_init_nest_lock_with_hint (
  omp_nest_lock_t *lock,
  omp_sync_hint_t hint);
```
For
```
subroutine omp_init_lock_with_hint (svar, hint)
integer (kind=omp_lock_kind) svar
integer (kind=omp_sync_hint_kind) hint

subroutine omp_init_nest_lock_with_hint (
  nvar, hint)
integer (kind=omp_nest_lock_kind) nvar
integer (kind=omp_sync_hint_kind) hint
```

hint: [2.17.12] を参照

ロックの破棄 [3.3.3] [3.3.3]

OpenMP ロックを破棄します。ロックを非初期化状態に戻します。

C/C++
```
void omp_destroy_lock (omp_lock_t *lock);

void omp_destroy_nest_lock (omp_nest_lock_t
                            *lock);
```
For
```
subroutine omp_destroy_lock (svar)
integer (kind=omp_lock_kind) svar

subroutine omp_destroy_nest_lock (nvar)
integer (kind=omp_nest_lock_kind) nvar
```

ロックの設定 [3.3.4] [3.3.4]

OpenMP ロックを設定します。ロックが設定されるまで、呼び出したタスクは中断されます。

C/C++
```
void omp_set_lock (omp_lock_t *lock);

void omp_set_nest_lock (omp_nest_lock_t
                        *lock);
```
For
```
subroutine omp_set_lock (svar)
integer (kind=omp_lock_kind) svar

subroutine omp_set_nest_lock (nvar)
integer (kind=omp_nest_lock_kind) nvar
```

ロックの解除 [3.3.5] [3.3.5]

OpenMP ロックを解除します。

C/C++
```
void omp_unset_lock (omp_lock_t *lock);

void omp_unset_nest_lock (omp_nest_lock_t
                          *lock);
```
For
```
subroutine omp_unset_lock (svar)
integer (kind=omp_lock_kind) svar

subroutine omp_unset_nest_lock (nvar)
integer (kind=omp_nest_lock_kind) nvar
```

ランタイム・ライブラリー・ルーチン (続き)

ロックをテスト [3.3.6] [3.3.6]
OpenMP ロックの設定を試みますが、ルーチンを実行しているタスクの実行を中断しません。

C/C++
```
void omp_test_lock (omp_lock_t *lock);
void omp_test_nest_lock (omp_nest_lock_t *lock);
```
For
```
logical function omp_test_lock (svar)
integer (kind=omp_lock_kind) svar
integer function omp_test_nest_lock (nvar)
integer (kind=omp_nest_lock_kind) nvar
```

タイミングルーチン

タイミングルーチンは、ポータブルなウォールクロック・タイマーをサポートします。これらは、スレッドごとの経過時間を記録しますが、アプリケーションを構成するすべてのスレッド間の一貫性が保証されるわけではありません。

omp_get_wtime [3.4.1] [3.4.1]
ウォールクロックの経過時間を秒単位で返します。

C/C++
```
double omp_get_wtime (void);
```
For
```
double precision function omp_get_wtime ()
```

omp_get_wtick [3.4.2] [3.4.2]
omp_get_wtime で使用されるタイマーの精度 (ティック間の秒数) を返します。

C/C++
```
double omp_get_wtick (void);
```
For
```
double precision function omp_get_wtick ()
```

イベントルーチン

イベントルーチンは、OpenMP のイベント・オブジェクトをサポートします。このオブジェクトには、このセクションで説明するルーチン、または **task** 構造の **detach** 節を介してアクセスする必要があります。

omp_fulfill_event [3.5.1]
OpenMP イベントを作成または破棄します。

C/C++
```
void omp_fulfill_event (
        omp_event_handle_t event);
```
For
```
subroutine omp_fulfill_event (event)
integer (kind=omp_event_handle_kind) event
```

デバイス・メモリー・ルーチン

デバイス・メモリー・ルーチンは、ターゲットデバイスのデータ環境でポインターの割り当てと管理をサポートします。

omp_target_alloc [3.6.1] [3.5.1]
デバイスデータ環境にメモリーを割り当てます。

C/C++
```
void *omp_targetl_alloc (
        size_t size, int device_num);
```

omp_target_free [3.6.2] [3.5.2]
omp_target_alloc ルーチンで割り当てられたデバイスメモリーを開放します。

C/C++
```
void omp_targetl_free (
        void *device_ptr, int device_num);
```

omp_target_is_present [3.6.3] [3.5.3]
ホストポインターが、指定されたデバイス上のデバイスバッファーに関連付けられているか確認します。

C/C++
```
int omp_targetl_is_present (
        const void *ptr, int device_num);
```

omp_target_memcpy [3.6.4] [3.5.4]
ホストとデバイスポインターの任意の組み合わせでメモリーをコピーします。

C/C++
```
int omp_target_memcpy (void *dst,
    const void *src, size_t length,
    size_t dst_offset, size_t src_offset,
    int dst_device_num, int src_device_num);
```

omp_target_memcpy_rect [3.6.5] [3.5.5]
多次元配列からほかの多次元配列へ矩形サブボリュームをコピーします。

C/C++
```
int omp_target_memcpy_rect (void *dst,
    const void *src, size_t element_size,
    int num_dims, const size_t *volume,
    const size_t *dst_offsets,
    const size_t *src_offsets,
    const size_t *dst_dimensions,
    const size_t *src_dimensions,
    int dst_device_num, int src_device_num);
```

omp_target_associate_ptr [3.6.6] [3.5.6]
omp_target_alloc もしくは実装定義のランタイムルーチンから返されるデバイスポインターをホストポインターにマップします。

C/C++
```
int omp_target_associate_ptr (
    const void *host_ptr,
    const void *device_ptr, size_t size,
    size_t device_offset, int device_num);
```

omp_target_disassociate_ptr [3.6.7] [3.5.7]
ホストポインターから、指定されたデバイスに関連するポインターを削除します。

C/C++
```
int omp_target_disassociate_ptr (
    const void *ptr, int device_num);
```

メモリー管理ルーチン

メモリー管理タイプ [3.7.2]
C/C++ の **omp_alloctrait_t** 構造体と Fortran の **omp_alloctrait** 型は、次の型と値を持つメンバー名 *key* と *value* を定義します:

enum **omp_alloctrait_key_t** (*C/C++*)
integer **omp_alloctrait_key_kind** (*For*)

omp_atk_X。ここで X は sync_hint、alignment、access、pool_size、fallback、fb_data、pinned、partition のいずれかです。

enum **omp_alloctrait_value_t** (*C/C++*)
integer **omp_alloctrait_val_kind** (*For*)

omp_atv_X。こで X は false、true、default、contended、uncontended、sequential、private、all、thread、pteam、cgroup、default_mem_fb、null_fb、abort_fb、allocator_fb、environment、nearest、blocked、interleaved のいずれかです。

omp_init_allocator [3.7.2]
アロケーターを初期化してメモリー空間に関連付けます。

C/C++
```
omp_allocator_handle_t omp_init_allocator (
    omp_memspace_handle_t memspace,
    int ntraits, const omp_alloctrait_t traits[]);
```
For
```
integer (kind=omp_allocator_handle_kind) &
function omp_init_allocator (&
memspace, ntraits, traits)
integer (kind=omp_memspace_handle_kind),&
intent (in) :: memspace
integer, intent (in) :: ntraits
type (omp_alloctrait), intent (in) :: traits (*)
```

omp_destroy_allocator [3.7.3]
アロケーター・ハンドルによって使用されたすべてのリソースを解放します。

C/C++
```
void omp_destroy_allocator (
    omp_allocator_handle_t allocator);
```
For
```
subroutine omp_destroy_allocator (allocator)
integer (kind=omp_allocator_handle_kind), &
intent (in) :: allocator
```

omp_set_default_allocator [3.7.4]
アロケーション呼び出し、 **allocate** ディレクティブ、および **allocate** 節で使用されるデフォルトのメモリー・アロケーターを指定します。

C/C++
```
void omp_set_default_allocator (
    omp_allocator_handle_t allocator);
```
For
```
subroutine omp_set_default_allocator ( &
    allocator)
integer (kind=omp_allocator_handle_kind), &
intent (in) :: allocator
```

omp_get_default_allocator [3.7.5]
アロケーション呼び出し、 **allocate** ディレクティブ、およびアロケーターを指定しない **allocate** 節で使用されるメモリー・アロケーターを返します。

C/C++
```
omp_allocator_handle_t
  omp_get_default_allocator (void);
```
For
```
integer (kind=omp_allocator_handle_kind)
function omp_get_default_allocator ()
```

omp_alloc [3.7.6]
メモリー・アロケーターにメモリー割り当てを要求します。

C
```
void *omp_alloc (size_t  size,
    omp_allocator_handle_t allocator);
```
C++
```
void *omp_alloc (size_t size,
    const omp_allocator_t
    allocator=omp_null_allocator);
```

omp_free [3.7.7]
割り当てたメモリーを解放します。

C
```
void *omp_free (void *ptr,
    omp_allocator_handle_t allocator);
```
C++
```
void *omp_free (void *ptr,
    omp_allocator_handle_t
    allocator=omp_null_allocator);
```

ツール制御ルーチン

omp_control_tool [3.8]
プログラムがアクティブなツールにコマンドを渡すことを可能にします。

C/C++
```
int omp_control_tool (int command,
    int modifier, void *arg);
```
For
```
integer function omp_control_tool ( &
  command, modifier )
integer (kind=omp_control_tool_kind) &
  command
integer modifier
```

command:

omp_control_tool_start: 監視がオフの場合は監視を開始または再開します。監視がオンの場合は保持されます。監視が永続的にオフの場合は効果がありません。
omp_control_tool_pause: 一時的に監視をオフにします。オフの場合は保持されます。
omp_control_tool_flush: バッファーのデータをフラッシュします。監視がオン/オフでも適用されます。
omp_control_tool_end: 監視を永続的にオフにします。ツールは自動的に終了し、すべての出力をフラッシュします。

節

節のすべてのリスト項目は、ベース言語のスコープ規則に従って表記される必要があります。ここに記載されている節は、すべてのディレクティブで有効であるとは限りません。

allocate 節 [2.11.4]

allocate (*[allocator:] list*)
ディレクティブのプライベート変数のストレージを取得するメモリー・アロケーターを指定します。

allocator:
C/C++ **omp_allocator_handle_t** 型の式
For **omp_allocator_handle_kind** *kind* の整数式

データコピー節 [2.19.6] [2.15.4]

copyin (*list*)
マスタースレッドの threadprivate 変数の値を、**parallel** 領域を実行するチームのメンバーの threadprivate 変数にコピーします。

copyprivate (*list*)
暗黙的なタスクのデータ環境から **parallel** 領域に属しているほかの暗黙的なタスクのデータ環境に値をブロードキャストします。

データ共有属性節 [2.19.4] [2.15.3]

データ共有属性は、節が指定された構造で名前がリストされている変数に対してのみ有効です。

default (**shared** | **none**) C/C++
default (**private** | **firstprivate** | **shared** | **none**) For
parallel、**teams**、およびタスク生成構造内で参照される変数のデフォルトのデータ共有属性を明示的に決定し、構造内のデータ共有属性が暗黙的に決定されるすべての変数参照に適用します。

shared (*list*)
parallel、**teams**、またはタスク生成構造で生成されるタスクが共有するリスト項目を宣言します。明示的な **task** 領域が実行を完了するまで、その領域で共有されるストレージは存続していなければなりません。

private (*list*)
タスクまたは SIMD レーンでプライベートであるリスト項目を宣言します。構造内でリスト項目を参照する各タスクまたは SIMD レーンは、構造内に 1 つ以上の関連するループがあり、**ordered (concurrent)** 節が存在しないかぎり、1 つの新しいリスト項目のみを受け取ります。

firstprivate (*list*)
リスト項目がタスクでプライベートであることを宣言し、それぞれ構造内で使用するときにオリジナルの項目の値で初期化します。

lastprivate (*[lastprivate-modifier:] list*)
暗黙のタスクもしくは SIMD レーンでプライベートである 1 つ以上のリスト項目を宣言し、領域終了後にオリジナルのリスト項目の値を更新します。
lastprivate-modifier: **conditional**

linear (*linear-list [: linear-step]*)
プライベートであり、節が指定された構造に関連するループの反復空間とリニアな関係を持つ 1 つ以上のリスト項目を宣言します。
linear-list: list もしくは *modifier(list)*
modifier: **ref**、**val**、または **uval** (C では **val** のみ)

defaultmap 節 [2.19.7.2] [2.15.5.2]

defaultmap (*implicit-behavior [:variable-ategory]*)
target 構造で参照されるデータマッピング属性を明示的に決定します。それ以外は暗黙的に決定されます。
map-type: **alloc**、**to**、**from**、**tofrom**、**firstprivate**、**none**、**default**
variable-category: C/C++
scalar、**aggregate**、**pointer**
variable-category: For
scalar、**aggregate**、**pointer**、**allocatable**

depend 節 [2.17.11] [2.13.9]

タスクまたはループ反復のスケジュールに追加の制約を適用します。これらの制約は、兄弟タスクまたはループ反復間のみの依存関係を設定します。

depend (*dependence-type*)
dependence-type は、ソースです。

depend (*dependence-type: vec*)
dependence-type は、*sink* で *vec* 反復は次の形式です: x1 [± d1], x2 [± d2], . . . , xn [± dn]

depend(*[depend-modifier,]dependence-type* : *locator-list*)
dependence-modifier: **iterator** (iterators-definition)
dependence-type: **in**、**out**、**inout**、**depobj**、**mutexinoutset**

- **in**: 生成されるタスクは、**out** または **inout** *dependence-type* リストの 1 つ以上のリスト項目を参照する、以前に生成されたすべての兄弟タスクに依存します。
- **out** と **inout**: 生成されるタスクは、**in**、**out**、**mutexinoutset**、または **inout** *dependence-type* リストの 1 つ以上のリスト項目を参照する、以前に生成されたすべての兄弟タスクに依存します。
- **mutexinoutset:** リスト項目の少なくとも 1 つの格納場所が、兄弟タスクが生成された構造の **in**、**out**、**inout** *dependence-type* を持つ **depend** 節のリスト項目の格納場所と同じである場合、生成されるタスクは兄弟タスクの従属タスクになります。リスト項目の少なくとも 1 つの格納場所が、兄弟タスクが生成された構造の **mutexinoutset** *dependence-type* を持つ **depend** 節のリスト項目の格納場所と同じである場合、兄弟タスクは相互に排他的なタスクとなります。
- **depobj:** タスク依存関係は、現在の構造で **depobj** 構造の **depend** 節が指定されたかのように、**depend** 節で指定された依存オブジェクトの依存関係を初期化した **depobj** 構造の **depend** 節から派生します。

if 節 [2.5] [2.12]

if 節の作用は、適用される構造に依存します。結合されたもしくは複合構造においては、*directive-name-modifier* で指定される構造のセマンティクスにのみ適用されます。結合もしくは複合構造で **if** 節が指定されていない場合、**if** 節は適用されるすべての構造に作用します。

if (*[directive-name-modifier :] scalar-expression*)
C/C++
if (*[directive-name-modifier :] scalar-logical-expression*) For

map 節 [2.19.7.1] [2.15.5.1]

map (*[[map-type-modifier[,] [map-type-modifier [,] ...] map-type* : *] locator-list*)

オリジナルのリスト項目を現在のタスクのデータ環境から構造で指定されたデバイスのデータ環境の対応するリスト項目へマップします。

map-type:
alloc、**to**、**from**、**tofrom**、**release**、**delete**
map-type-modifier:
always、**close**、**mapper** (*mapper-idenrifier*)

ordered 節 [2.9.2]

ordered *[(n)]*
ループ、または構造に関連付けるループの数を示します。

reduction 節 [2.19.5]

in_reduction (*reduction-identifier*: *list*)
タスクがリダクションに参加することを指定します。
reduction-identifier: **reduction** と同じ

task_reduction (*reduction-identifier*: *list*)
タスク間のリダクションを指定します。

reduction (*[reduction-modifier ,] reduction-identifier* : *list*)
reduction-identifier と 1 つ以上のリスト項目を指定します。

reduction-modifier: **inscan**、**task**、**default**
reduction-identifier: C++ *id-expression* または次の演算子を指定します: **+**、**-**、*、**&**、**|**、**^**、**&&**、**||**
reduction-identifier: C *identifier* または次の演算子を指定します: **+**、**-**、*、**&**、**|**、**^**、**&&**、**||**
reduction-identifier: For ベース言語の識別子、ユーザー定義演算子、または次の演算子のいずれかを指定します: **+**、**-**、*、**.and.**、**.or.**、**.eqv.**、**.neqv.**、または次の組込みプロシージャー名のいずれかを指定します: **max**、**min**、**iand**、**ior**、**ieor**

SIMD 節 [2.9.3] [2.8]

aligned(*argument-list [:alignment]*)
指定するバイト数にアライメントする 1 つ以上のリスト項目を宣言します。*alignment* は正の定数整数式でなければなりません。このパラメーターを指定しない場合、ターゲット・プラットフォーム上の SIMD 命令の実装で定義されるデフォルトのアライメントが適用されます。

collapse(*n*)
正の定数整数式で、より大きな 1 つの反復空間のループに畳み込む構造内の入れ子ループの数を指定します。

inbranch
関数が常に SIMD ループの条件文から呼び出されることを指定します。

nontemporal (*list*)
list 項目が参照するストレージへのアクセスが、それらのストレージがアクセスされる反復間の参照の局所性が低いことを指定します。

notinbranch
関数が常に SIMD ループの条件文から呼び出されないことを指定します。

safelen (*length*)
SIMD 命令で同時に実行される 2 つの反復の距離が、論理反復空間において指定した値以下でなければならないことを指定します。

simdlen(*length*)
正の定数整数式で同時に実行する反復数を指定します。

uniform (*argument-list*)
単一の SIMD ループで実行されるすべての同時関数呼び出しで不変値を持つように 1 つ以上の引数を宣言します。

task 節 [2.10] [2.9]

affinity (*[aff-modifier:] locator-list*)
リスト項目の位置に隣接して実行するヒントを指定します。*aff-modifier* は **iterator** (iterators-definition) です。

allocate (*[allocator: list*)
10 ページの allocate 節を参照してください。

collapse(*n*)
10 ページの SIMD 節を参照してください。

default (**shared** | **none**) C/C++
default (**private** | **firstprivate** | **shared** | **none**)
For
10 ページのデータ共有属性節を参照してください。

depend (*[depend-modifier,] dependence-type: locator-list*)
10 ページの depend 節を参照してください。

detach (list)
構造内のすべての変数リストへの暗黙の参照を引き起こします (taskloop では使用されません)。

final (*scalar-expression*) C/C++
final (*scalar-logical-expression*) For
final 式が true と評価される場合、生成されるタスクは final タスクとなります。

firstprivate (*list*)
10 ページのデータ共有属性節を参照してください。

grainsize (*grain-size*)
生成される各タスクに割り当てられる論理ループの反復数は、*grain-size* 式の値と論理ループの反復回数の最小値より大きいか等しく、*grain-size* 式の値の 2 倍未満です。

if (*[* **task** **:** *]* *scalar-expression*) C/C++
if (*[* **task** **:** *]* *scalar-logical-expression*) For
10 ページの if 節を参照してください。

in_reduction (*reduction-identifier: list*)
10 ページの reduction 節を参照してください。

lastprivate (*list*)
10 ページのデータ共有属性節を参照してください。

mergeable
生成されるタスクがマージ可能であることを指定します。

nogroup
暗黙の **taskgroup** 領域を作成しないようにします。

num_tasks (*num-tasks*)
num-tasks 式の値と論理ループの反復回数の最小値と同じ数のタスクを生成します。

priority (*priority-value*)
生成されるタスクに優先順位を付けるためのヒントを指定する負ではない数値スカラー式です。

private (*list*)
10 ページのデータ共有属性節を参照してください。

reduction (*[default ,] reduction-identifier*: *list*)
10 ページの reduction 節を参照してください。

shared (*list*)
10 ページのデータ共有属性節を参照してください。

untied
この節が指定されると、チーム内の任意のスレッドは中断後にタスク領域を再開できます。

イテレーター

iterators [2.1.6]
句の中で複数の値に展開される識別子。

iterator (*iterators-definition*)

iterators-definition:
iterator-specifier [, iterators-definition]

iterators-specifier:
[iterator-type] identifier **=** *range-specification*

identifier: ベース言語の識別子。

range-specification: *begin* : *end*[: *step*]
begin, *end*: 型を *iterator-type* に変換できる式。

step: 整数式
iterator-type: 型名 C/C++
iterator-type: 型指定子 For

内部制御変数 (ICV) 値

ホストおよびターゲットデバイスの ICV は、OpenMP API 構造またはルーチンが実行される前に初期化されます。初期値が割り当てられた後、ユーザーによって設定された環境変数の値が読み取られ、それに応じてホストデバイスの関連する ICV が変更されます。ターゲットデバイスの ICV を初期化する方法は実装定義です。

ICV 初期値の表 (表 2.1) および ICV 値の変更と取得方法 (表 2.2) [2.5.2-3] [2.3.2-3]

ICV	環境変数	初期値	値の変更	値の取得	参照
dyn-var	**OMP_DYNAMIC**	実装がスレッド数の動的変更をサポートする場合は実装定義。それ以外の初期値は false。	**omp_set_dynamic()**	**omp_get_dynamic()**	[6.3] [4.3]
nest-var	**OMP_NESTED**	実装定義	**omp_set_nested()**	**omp_get_nested()**	[6.9] [4.6]
nthreads-var	**OMP_NUM_THREADS**	実装定義リスト	**omp_set_num_threads()**	**omp_get_max_threads()**	[6.2] [4.2]
run-sched-var	**OMP_SCHEDULE**	実装定義	**omp_set_schedule()**	**omp_get_schedule()**	[6.1] [4.1]
def-sched-var	(なし)	実装定義	(なし)	(なし)	---
bind-var	**OMP_PROC_BIND**	実装定義リスト	(なし)	**omp_get_proc_bind()**	[6.4] [4.4]
stacksize-var	**OMP_STACKSIZE**	実装定義	(なし)	(なし)	[6.6] [4.7]
wait-policy-var	**OMP_WAIT_POLICY**	実装定義	(なし)	(なし)	[6.7] [4.8]
thread-limit-var	**OMP_THREAD_LIMIT**	実装定義	**thread_limit 節**	**omp_get_thread_limit()**	[6.10] [4.10]
max-active-levels-var	**OMP_MAX_ACTIVE_LEVELS**、**OMP_NESTED**	実装がサポートする並列処理のレベル数	**omp_set_max_active_levels()**、**omp_set_nested()**	**omp_get_max_active_levels()**	[6.8] [4.9]
active-levels-var	(なし)	ゼロ	(なし)	**omp_get_active_level()**	---
levels-var	(なし)	ゼロ	(なし)	**omp_get_level()**	---
place-partition-var	**OMP_PLACES**	実装定義	(なし)	**omp_get_partition_num_places()** **omp_get_partition_place_nums()** **omp_get_place_num_procs()** **omp_get_place_proc_ids()**	[6.5] [4.5]
cancel-var	**OMP_CANCELLATION**	false	(なし)	**omp_get_cancellation()**	[6.11] [4.11]

内部制御変数 (ICV) 値 (続き)

ICV	環境変数	初期値	値の変更	値の取得	参照
display-affinity-var	**OMP_DISPLAY_AFFINITY**	false	(なし)	(なし)	[6.13]
affinity-format-var	**OMP_AFFINITY_FORMAT**	実装定義	**omp_set_affinity_format()**	**omp_get_affinity_format()**	[6.14]
default-device-var	**OMP_DEFAULT_DEVICE**	実装定義	**omp_set_default_device()**	**omp_get_default_device()**	[6.15] [4.13]
target-offload-var	**OMP_TARGET_OFFLOAD**	**デフォルト**	(なし)	(なし)	[6.17]
max-task-priority-var	**OMP_MAX_TASK_PRIORITY**	ゼロ	(なし)	**omp_get_max_task_priority()**	[6.16] [4.14]
tool-var	**OMP_TOOL**	有効	(なし)	(なし)	[6.18]
tool-libraries-var	**OMP_TOOL_LIBRARIES**	空の文字列	(なし)	(なし)	[6.19]
debug-var	**OMP_DEBUG**	無効	(なし)	(なし)	[6.20]
def-allocator-var	**OMP_ALLOCATOR**	実装定義	**omp_set_default_allocator()**	**omp_get_default_allocator()**	[6.21]

環境変数

環境変数は大文字で表され、代入される値は大文字と小文字が区別されず、先頭と末尾にスペースを含めることができます。

OMP_ALLOCATOR *arg* [6.21]

アロケーターを指定しないアロケーション呼び出し、ディレクティブ、および節のデフォルト・アロケーターを指定する内部変数 *def-allocator-var* を設定します。*arg* は、大文字と小文字を区別しない、事前定義された次のアロケーターです (詳細は、表 2.10 を参照)。

omp_default_mem_alloc
omp_low_lat_mem_alloc
omp_large_cap_mem_alloc
omp_const_mem_alloc
omp_high_bw_mem_alloc
omp_cgroup_mem_alloc
omp_pteam_mem_alloc
omp_thread_mem_alloc

OpenMP メモリー・アロケーターを使用して割り当て要求を行います。割り当てプロセスの動作は、指定されたアロケーター特性の影響を受けます。次の **[表 2.9]** は、許容されるアロケーター特性と設定可能な値を示します。デフォルト値は青で示します。

アロケーター特性	許容値 (デフォルト)
sync_hint	**contended**、**uncontended**、**serialized**、**private**
alignment	**1 バイト**、2 のべき乗である正の整数値
access	**all**、**cgroup**、**pteam**、**thread**
pool_size	正の整数値 (デフォルトは実装定義)
fallback	**default_mem_fb**、**null_fb**、**abort_fb**、**allocator_fb**
fb_data	アロケーター・ハンドル (デフォルトなし)
pinned	**true**、**false**
partition	**environment**、**nearest**、**blocked**、**interleaved**

OMP_AFFINITY_FORMAT *format* [6.14]

OpenMP スレッド・アフィニティー情報を表示する際の形式を定義する内部制御変数 *affinity-format-var* の初期値を設定します。引数は文字列で、他の文字に加えて、サブ文字列として 1 つ以上のフィールド指定子を含むことができます。各フィールド指定子の形式は、%[[[0].*size*] *type* で、フィールドタイプは次にリストされる短縮名または長い名前です **[表 6.2]**。

t team_num	**n** thread_num
T num_teams	**N** num_threads
L nesting_level	**a** ancestor_tnum
P process_id	**A** thread_affinity
H host	**i** native_thread_id

OMP_CANCELLATION *var* [6.11] [4.11]
内部制御変数 *cancel-var* を設定します。*var* は、**true** または **false** です。**true** の場合、cancel 構造と cancellation point が有効になり、取り消しがアクティブになります。

OMP_DEBUG *var* [6.20]
内部制御変数 *debug-var* を設定します。*var* は **enabled** または **disabaled** です。**enabled** の場合、OpenMP 実装はサードパーティー・ツールに提供する追加のランタイム情報を収集します。**disabled** の場合、デバッガーが利用できる機能が制限される可能性があります。

OMP_DEFAULT_DEVICE *device* [6.13] [4.13]
device 構造で使用するデフォルトのデバイス数を制御する内部制御変数 *default-device-var* を設定します。

OMP_DISPLAY_AFFINITY *var* [6.13]
並列領域内のすべての OpenMP スレッドの書式付きアフィニティー情報を表示することをランタイムに指示します。情報は、最初の並列領域に入ったとき、および **OMP_AFFINITY_FORMAT** で指定されるフォーマット指定子によってアクセス可能な情報に変更がある場合に表示されます。並列領域内のスレッドのアフィニティーが変更された場合、その領域内のすべてのスレッドのアフィニティー情報が表示されます。

OMP_DISPLAY_ENV *var* [6.12] [4.12]
var が **TRUE** の場合、実行時に OpenMP バージョン番号と環境変数に関連する内部制御変数の値を *name=value* として表示するよう指示します。*var* が **VERBOSE** の場合、実行時にベンダー固有の値も表示されます。*var* が **FALSE** の場合、情報は表示されません。

OMP_DYNAMIC *var* [6.3] [4.3]
内部制御変数 *dyn-var* を設定します。**TRUE** の場合、**parallel** 領域を実行するスレッド数を動的に調整することを許可します。

OMP_MAX_ACTIVE_LEVELS *levels* [6.8] [4.9]
入れ子になったアクティブな **parallel** 領域の最大数を制御する内部制御変数 *max-active-levels-var* を設定します。

OMP_MAX_TASK_PRIORITY *level* [6.16] [4.14]
使用するタスクの優先順位を制御する内部制御変数 *max-task-priority-var* を設定します。

● **OMP_NESTED** *nested* [6.9] [4.6]
入れ子になった並列処理を制御する内部制御変数 *nest-var* を設定します。

OMP_NUM_THREADS *list* [6.2] [4.2]
parallel 領域で使用するスレッド数を制御する内部制御変数 *nthreads-var* を設定します。

OMP_PLACES *places* [6.5] [4.5]
実行環境で利用可能な OpenMP プレースを定義する内部制御変数 *place-partition-var* を設定します。*places* は、抽象的な名称 (**threads**、**cores**、**sockets**、または実装定義)、または負ではない数のリストです。

OMP_PROC_BIND *policy* [6.4] [4.4]
対応する入れ子レベルの **parallel** 領域で使用するスレッド・アフィニティー・ポリシーを定義するグローバル内部制御変数 *bind-var* を設定します。*policy* には、**true**、**false**、または引用符で囲ったカンマ区切りの **master**、**close**、もしくは **spread** リストを指定できます。

OMP_SCHEDULE [*modifier:*] *kind* [,*chunk*] [6.1] [4.1]
ランタイム・スケジュールのタイプとチャンクサイズを制御する内部制御変数 *run-sched-var* を設定します。*modifier* は、**monotonic** または **nonmonotonic** で、*kind* は、**static**、**dynamic**、**guided**、または **auto** です。

環境変数 (続き)

OMP_STACKSIZE *size[*B | K | M | G*]* [6.6] [4.7]
OpenMP の実装によって作成されるスレッドのスタックサイズを定義する内部変数 *stacksize-var* を設定します。*size* は、スタックサイズを指定する正の整数値です。単位が指定されない場合、*size* はキロバイト (K) 単位です。

OMP_TARGET_OFFLOAD *arg* [6.17]
target-offload-var の初期値を設定します。引数は、**MANDATORY**、**DISABLED**、または **DEFAULT** のいずれかである必要があります。

OMP_THREAD_LIMIT *limit* [6.10] [4.10]
OpenMP プログラムに関連するスレッド数を制御する内部制御変数 *thread-limit-var* を設定します。

OMP_TOOL (enabled | disabled) [6.18]
tool-var を設定します。disabled にすると、ファーストパーティー・ツールはロードも初期化も行われません。enabled の場合、OpenMP 実装はファーストパーティー・ツールを検出して有効にしようとします。

OMP_TOOL_LIBRARY *library-list* [6.19]
tool-libraries-var を、OpenMP 実装が初期化されるデバイスで使用されると想定されるライブラリーのリストに設定します。*library-list* 引数は、絶対パスで指定されたコロン区切りのダイナミック・リンク・ライブラリーのリストです。

OMP_WAIT_POLICY *policy* [6.7] [4.8]
待機中のスレッドの動作に関するヒントを OpenMP 実装に提供する内部制御変数 *wait-policy-var* に設定します。有効な *policy* の値は **ACTIVE** (待機中のスレッドはプロセッサー時間を消費) もしくは **PASSIVE** です。

ツールのアクティベーション

OMPT ツールをアクティブにする [4.2]
OpenMP 実装がツールをアクティブにするには 3 つのステップがあります。ここでは、そのためにツールと OpenMP 実装がどのように作用するか説明します。

ステップ 1: 初期化するかどうか決定する [4.2.2]
ツールは、**ompt_start_tool** からの戻り値として、OpenMP 実装の **ompt_start_tool_result_t** 構造体へ **NULL** 以外のポインターを提供することで、OMPT インターフェイスを使用することを示します。

ツールが OpenMP 実装に **ompt_start_tool** 定義を提供する方法は、次の 3 つがあります。

- ツールの **ompt_start_tool** 定義を OpenMP アプリケーションに静的にリンクします。
- ツールの **ompt_start_tool** 定義をアプリケーションのアドレス空間に含めるダイナミック・リンク・ライブラリーを提供します。
- *tool-libraries-var* を使用して、アプリケーションが使用するアーキテクチャーとオペレーティング・システムに適したダイナミック・リンク・ライブラリー名を提供します。

ステップ 2: ファーストパーティー・ツールを初期化する [4.2.3]
ツールが提供する **ompt_start_tool** 実装が、**ompt_start_tool_result_t** 構造体へ **NULL** 以外のポインターを返す場合、OpenMP 実装は OpenMP イベントが発生する前に、この構造で指定されたツール初期化子を呼び出します。

ステップ 3: ホストでアクティビティーを監視する [4.2.4]
ホストデバイスで OpenMP プログラムの実行を監視するには、ツールの初期化子が OpenMP プログラムの実行中に発生したイベントを受け取るように登録する必要があります。ツールは、**ompt_set_callback** ランタイム・エントリー・ポイントを使用して、OpenMP イベントのコールバックを登録します。 **ompt_set_callback** は次のコードを返します。

ompt_set_error
ompt_set_never
ompt_set_sometimes
ompt_set_sometimes_paired
ompt_set_always
ompt_set_impossible

ompt_set_callback ランタイム・エントリー・ポイントが、ツールの初期化子外部から呼び出された場合、サポートされるコールバックの登録はリターンコード **ompt_set_error** で失敗する可能性があります。

ompt_set_callback で登録された、または **ompt_get_callback** で返されたすべてのコールバックは、ダミー型シグネチャー **ompt_callback_t** を使用します。これは、最適ではありませんが、正確な型シグネチャーを持つユニークなランタイム・エントリー・ポイントを指定して、ランタイム・エントリー・ポイント型シグネチャーごとにコールバックを設定して取得するよりも適切です。

メモ