# OpenMP 5.2 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コアチップ、NUMA システム、GPU、および CPU に接続されたその他の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 | [n.n.n] 5.2 仕様のセクション | [n.n.n] 5.1 仕様のセクション | ⑨ 9 ページの節情報を参照

## はじめに

### このリファレンス・ガイドの構成

ディレクティブと構造 ... 1
節 ... 9
ランタイム・ライブラリー・ルーチン ... 10
環境変数 ... 14
ICV の値 ... 15
OpenMP ツールを利用 ... 16

### OpenMP の例とドキュメント

サンプルコードとサンプル・ドキュメント、GitHub リポジトリーへのリンクは、link.openmp.org/examples51。

### OpenMP ディレクティブの構文

ディレクティブは、基本言語のメカニズムとディレクティブ仕様 (ディレクティブ名に後にオプション節が続く) の組み合わせです。構造は、ディレクティブと追加の基本言語コードで構成されます。

*C/C++* C ディレクティブは、pragma で形成されます。C++ ディレクティブは、pragma または attribute のいずれかで形成されます。

*Fortran* Fortran ディレクティブは、自由形式または固定形式のソースのコメントとして形成されます。

**例:**

*C/C++* **#pragma omp** *directive-specification*
*C++* **[[omp :: directive(** *directive-specification* **)]]**
*C++* **[[using omp : directive(** *directive-specification* **)]]**
*For* **!$omp** *directive-specification*
*For* **!$omp** *directive-specification*
**!$omp end** *directive-name*

## ディレクティブと構造

OpenMP 構造がディレクティブと構文で定義される場合、それに続く構造化ブロックに適用されます。・simd および宣言型ディレクティブを除く OpenMP ディレクティブは、Fortran の PURE プロシージャーでは使用できません。・構造化ブロックは、実行可能な単一の文またはブロックであり、1 つの入り口と出口があります。・厳密な構造化ブロックは、Fortran の BLOCK 構造である構造化ブロックです。・緩い構造化ブロックは、厳密に構造化されない *Fortran* の BLOCK 構造で開始されない構造化ブロックです。・omp 整数式は、*C/C++* スカラー int 型または *Fortran* スカラー integer 型です。・omp 論理式は、*C/C++* スカラー式または Fortran 論理式です。

### データ環境ディレクティブ

**threadprivate [5.2] [2.21.2]**
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 (**リスト**)**
For **!$omp threadprivate (**リスト**)**

リスト:
*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。
*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 [5.5.11] [2.21.5.7]**
**reduction**、**in_reduction**、**task_reduction** 節で使用できるリダクション識別子を宣言します。

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

コンバイナー: *C/C++* 式、*For* 割り当て文またはサブルーチン名とそれに続く引数リスト。
*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:
*C/C++* 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: +、*、&、|、^、&&、||
*For* 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
*C/C++* タイプ名リスト: タイプ名のリスト
*For* タイプリスト:
**CLASS(*)** または抽象タイプのタイプ指定子のリスト

**scan [5.6] [2.11.8]**
ワークシェア・ループ、ワークシェア・ループ SIMD、または **simd** ディレクティブに関連する入れ子のループの各反復で、スキャン計算がリスト項目を更新することを指定します。

C/C++
{
構造化ブロックシーケンス
**#pragma omp scan** 節
構造化ブロックシーケンス
}
For
構造化ブロックシーケンス
**!$omp scan** 節
構造化ブロックシーケンス

節: **inclusive**(リスト)、**exclusive**(リスト)

**declare mapper [5.8.8] [2.21.7.4]**
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] ⋯]*

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:]* リスト) ⑨
マップタイプ: **alloc**、**to**、**from**、**tofrom**
マップタイプ修飾子: **always**、**close**、**present**、**mapper(default)**、**iterator(**イテレーター定義**)**
マッパー識別子: ベース言語の識別子または **default** ⑨

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

### メモリー管理ディレクティブ

**メモリー空間 [6.1] [2.13.1]**
定義済みメモリー空間は、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 |
| omp_large_cap_mem_space | 大容量 |
| omp_const_mem_space | 定数値の変数用 |
| omp_high_bw_mem_space | 高帯域幅 |
| omp_low_lat_mem_space | 低レイテンシー |

**allocate [6.6] [2.13.3]**
変数の割り当て方法を指定します。

C/C++ **#pragma omp allocate** (リスト) *[節 [[,] 節] ⋯]*
For **!$omp allocate** (リスト) *[節 [[,] 節] ⋯]*

節:
**align** (*アライメント*)
アライメント: 2 の整数乗
**allocator** (*allocator*)
*allocator* は次の表現:
*C/C++* **omp_allocator_handle_t タイプ**
*For* *kind* **omp_allocator_handle_kind**

**allocators [6.7]**
関連するアロケート文で割り当てられる変数に OpenMP メモリー・アロケーターを使用することを指示します。

For **!$omp allocators** *[節 [[,] 節] ⋯]*
アロケート文
**[!$omp end allocators]**

節: **allocate** ⑨
アロケート文: Fortran の ALLOCATE 文

### バリアント・ディレクティブ

**[begin] metadirective [7.4.3、7.4.4] [2.3.4]**
指定する OpenMP コンテキストをベースに **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*])
ディレクティブ・バリアント条件付きで選択。
**otherwise** (*[declare-variant]*)
**otherwise** は **default** と呼ばれていました。

# ディレクティブと構造 (続き)

## [begin] declare variant [7.5.4-5] [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 *[[ [,] 節] ...]*

*[***#pragma omp declare variant(***variant-func-id***) ¥**
節 *[[ [,] 節] ...]*

関数定義または宣言

- または -

***#pragma omp declare variant* 節-*match***

宣言定義シーケンス

***#pragma omp end declare variant***

For

**!$omp declare variant (***[base-proc-name:] & variant-proc-name***) 節** *[[ [,] 節] ...]*

節:

**adjust_args** (*adjust* 操作 *:* 引数リスト)
*adjust* 操作 *:* **nothing**、**need_device_ptr**
**append_args** (*append* 操作 *[[, append* 操作*] …])*
*append* 操作*: interop* (*interop-type [[,interop-type]...]*)

**enter**
**link**
**match**(*context-selector-specification*)
必須。指定されたバリアント関数が置き換えのために選択される際に、基本関数の引数を適合する方法を指定します。
*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。
節*-match:*
**match** (*context-selector-specification*)
**match** 節で必要です。

## dispatch [7.6] [2.3.6]

特定の呼び出しに対してバリアント置換を行うか制御します。

C/C++

**#pragma omp dispatch** ***[***節 *[[,] 節] ...]*
関数ディスパッチ構造化ブロック

For

**!$omp dispatch** ***[***節 *[[,] 節] ...]*
関数ディスパッチ構造化ブロック
**[!$omp end dispatch]**

節:

**depend** (*[depend* 修飾子*,]*
依存関係タイプ *: locator* リスト) ⑨
**device** (*omp* 整数式) ⑨
デバイス構造に関連する target device を識別します。
**is_device_ptr** (リスト)
リストはデバイスポインターです。
**nocontext** (*omp* 論理式)
*omp* 論理式が *true* と評価されると、その構成は *OpenMP* コンテキストの構造セットには追加されません。
**novariants** (*omp* 論理式)
*omp* 論理式が *true* と評価されると、該当するディスパッチ領域での呼び出しに対する関数バリアントは選択されません。
**nowait** ⑨

## declare simd [7.7] [2.11.5.3]

SIMD ループ内の単一の呼び出しから、関数やサブルーチンが、SIMD 命令を使用して複数の引数を処理可能な複数のバージョンを生成できるようにします。

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:

**aligned** (引数リスト *[: アライメント]*)
指定するバイト数でアライメントするリスト項目を宣言します。
アライメント: オプションの正の定数整数式です。
**inbranch**
**linear** (リスト *[: リニアステップ]*) ⑨
**notinbranch**
**simdlen** (レングス)
同時に実行する反復の回数を指定します。
**uniform** (引数リスト)
単一の SIMD ループを実行する際にすべての関数の同時呼び出しに対し、引数が不変値を持つように宣言します。

## [begin] declare target [7.8.1-2] [2.14.7]

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** (*extended* リスト)
- または -
**#pragma omp declare target** 節*[ [, ]節 ...]*
- または -
**#pragma omp begin declare target ¥**
*[節[ [, ]節] ...]*
宣言定義シーケンス
**#pragma omp end declare target**

Fortran

**!$omp declare target** (*extended* リスト)
- または -
**!$omp declare target** *[節[ [, ]節] ...]*

節:

**device_type (host | nohost | any)**
**enter** (extended リスト)
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のカンマで区切ったリストを指定します。
**indirect** *[(invoked-by-fptr)]*
*enter* 節のプロシージャーを間接的に呼び出すことが可能であるか決定します。
**link** (リスト)
リスト項目を参照する *target* 領域で呼び出される関数のコンパイルを可能にします。

- **declare target** の *2* 番目の *C/C++* 形式では、少なくても *1* つの節が **enter** または **link** である必要があります。
- **begin declare target** の場合、**enter** 節と **link** 節は許可されません。

## 情報とユーティリティー・ディレクティブ

## requires [8.2] [2.5.1]

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:

**atomic_default_mem_order (seq_cst | acq_rel |& relaxed)**
**dynamic_allocators**
対応する **target** 構造で **uses_allocators** 節を指定せずに、**target** 領域でメモリー・アロケーターを使用できるようにします (5 ページの **target** を参照)。
**reverse_offload**
**target** 構造が **ancestor** 修飾子で明示されるデバイス節を指定する場合、**target** 領域がそれを囲む **target** 領域の親デバイスで実行できることを保証する必要があります (5 ページの **target** を参照)。
**unified_address**
OpenMP API ルーチンやディレクティブを介して、アクセス可能なすべてのデバイスが、統合されたアドレス空間を使用する必要があります。
**unified_shared_memory**
**unified_address** の要件に加えて、メモリー内のストレージの位置に、使用可能なすべてのデバイスのスレッドがアクセスできることを保証します。

## assumes と [begin] assumes [8.3.2-4] [2.5.2]

最適化に使用可能な実装の不変条件を提供します。

C/C++

**#pragma omp assumes** 節 *[[[,] 節]…]*
- または -
**#pragma omp begin assumes** 節 *[[[,] 節 …]*
宣言定義シーケンス
**#pragma omp end assumes**
- または -
**#pragma omp assume** 節 *[[[,] 節]…]*
構造化ブロック

Fortran

**!$omp assumes** 節 *[[[,] 節]…]*
- または -
**!$omp assume** 節 *[[[,] 節]…]*
緩い構造化ブロック
**!$omp end assume**
**- または -**
**!$omp assume** 節 *[[[,] 節]…]*
厳密な構造化ブロック
*[***!$omp end assume** *]*

節:

**absent** (*directive* 名 *[[,directive* 名*]…])*
スコープに存在しないディレクティブをリストします。
***contains*** (*directive* 名 *[[,directive* 名*]…])*
スコープに存在する可能性があるディレクティブをリストします。
**holds** (*omp* 論理式)
スコープ内で真であることが保証される式。
**no_openmp**
スコープ内に OpenMP コードがないことを示します。
**no_openmp_routines**
スコープ内で OpenMP ランタイム・ライブラリーが呼び出されないことを示します。
**no_parallelism**
OpenMP タスクまたは SIMD 構造がスコープ内で実行されないことを示します。

# ディレクティブと構造 (続き)

### nothing [8.4] [2.5.3]
意図が効果を持たないことを明示的に示します。

C/C++
```
#pragma omp nothing
```
For
```
!$omp nothing
```

### error [8.5] [2.5.4]
メッセージを表示して、エラー処理を実行するようにコンパイラーまたはランタイムに指示します。

C/C++
```
#pragma omp error [節 [[,] 節 …]
```
For
```
!$omp error [節 [[,] 節 …]
```

節:
- **at (compilation | execution)**
- **message** (メッセージ文字列)
- **severity (fatal | warning)**

## loop 変換構造

### tile [9.1] [2.11.9.1]
1 つ以上の loop をタイル化します。

C/C++
```
#pragma omp tile sizes (サイズリスト)
   入れ子のループ
```
For
```
!$omp tile sizes (サイズリスト)
   入れ子のループ
[!$omp end tile]
```

節: **sizes** (サイズリスト)

### unroll [9.2] [2.11.9.2]
loop を完全にまたは部分的にアンロールします。

C/C++
```
#pragma omp unroll [節]
   入れ子のループ
```
For
```
!$omp unroll [節]
   入れ子のループ
[!$omp end unroll]
```

節:
- **full**
- **partial** [(アンロール係数)]

## 並列構造

### parallel [10.1] [2.6]
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
```
!$omp parallel [節 [[,] 節] …]
   緩い構造化ブロック
!$omp end parallel
 - または -
!$omp parallel [節 [[,] 節] …]
   厳密な構造化ブロック
[!$omp end parallel]
```

節:
- **allocate** ⑨
- **copyin**(リスト)
- **default** (データ共有属性) ⑨
- **firstprivate** (リスト) ⑨
- **num_threads** (整数式)
  実行するスレッド数を指定します。
- **private** (リスト) ⑨
- **proc_bind** (**close** | **primary**| **spread**)
  **close**: 親スレッドの場所に近い位置にチームのスレッドを割り当てるように実行環境に指示します。
  **primary**: チーム内のすべてのスレッドをプライマリー・スレッドと同じ位置に割り当てるように実行環境に指示します。
  **spread**: 親 place パーティションの P place 間で T スレッドのチームのスパース分布を行います。
- **reduction** ⑨
- **shared** (リスト) ⑨

### teams [10.2] [2.7]
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
 - または –
!$omp teams [節 [[,] 節] …]
   厳密な構造化ブロック
[!$omp end teams]
```

節:
- **allocate** ⑨
- **default** (データ共有属性) ⑨
- **firstprivate** (リスト) ⑨
- **num_teams** ([下限:] 上限)
- **private** (リスト) ⑨
- **reduction** ⑨
- **shared** (リスト) ⑨
- **thread_limit** (omp 整数式)

### simd [10.4] [2.11.5.1]
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] …]
   入れ子のループ
```
For
```
!$omp simd [節[ [, ]節] …]
   入れ子のループ
[!$omp end simd]
```

節:
- **aligned** (リスト [: アライメント])
  指定するバイト数でアライメントされる 1 つ以上のリスト項目を宣言します。
  アライメント: オプションの正の定数式。
- **collapse** (*n*) ⑨
- **if** ([*simd:*] *omp* 論理式) ⑨
- **lastprivate** ([*lastprivate* 修飾子*:*] リスト) ⑨
- **linear** (リスト [*:* リニアステップ]) ⑨
- **nontemporal** (リスト)
  リスト内のストレージへのアクセスは、ストレージ位置にアクセスする反復全体で時間的な局所性が低くなります。
- **order** ([order 修飾子] **concurrent**) ⑨
  order 修飾子: **reproduce** または **unconstrained**
- **private** (リスト) ⑨
- **reduction** ⑨
- **safelen** (レングス)
  この節を指定する場合、SIMD 命令で同時実行される 2 つの反復は、レングス値よりも論理反復空間の距離を上回ることはありません。
- **simdlen** (レングス)
  同時に実行する反復回数を指定します。

### masked [10.5] [2.8]
現在のチームのスレッドのサブセットで実行される構造化ブロックを指定します。

C/C++
```
#pragma omp masked [節]
   構造化ブロック
```
For
```
!$omp masked [節]
   緩い構造化ブロック
!$omp end masked
 - または –
!$omp masked [節]
   厳密な構造化ブロック
[!$omp end masked ]
```

節:
- **filter** (thread_num)
  実行するスレッド数を指定します。

## ワークシェア構造

### single [11.1] [2.10.2]
指定する構造化ブロックは、チーム内の 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
```
!$omp single [節[ [, ]節] …]
   緩い構造化ブロック
!$omp end single [end 節 [[,] end 節] …]
 - または –
!$omp single [節[ [, ]節] …]
   厳密な構造化ブロック
[!$omp single [end 節 [[,] end 節] …]]
```

節:
- **allocate** ⑨
- **copyprivate** (リスト)
- **firstprivate** (リスト) ⑨
- **nowait** ⑨
- **private** (リスト) ⑨

end 節:
- **copyprivate** (リスト)
- **nowait** ⑨

### workshare [11.4] [2.10.3]
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
```
!$omp workshare
   緩い構造化ブロック
!$omp end workshare [nowait]
 - または –
!$omp workshare
   厳密な構造化ブロック
[!$omp end workshare [nowait]]
```

節:
- **nowait** ⑨

### scope [11.2] [2.9]
チーム内のすべてのスレッドで実行されますが、追加の OpenMP 操作を指定できる構造化ブロックを定義します。

C/C++
```
#pragma omp scope [節[ [, ]節] …]
   構造化ブロック
```
For
```
!$omp scope [節[ [, ]節] …]
   緩い構造化ブロック
!$omp end scope [nowait]
 - または –
!$omp scope [節[ [, ]節] …]
   厳密な構造化ブロック
[!$omp end scope [nowait]]
```

節:
- **allocate** ⑨
- **firstprivate** (リスト) ⑨
- **nowait** ⑨
- **private** (リスト) ⑨
- **reduction** ⑨

# ディレクティブと構造 (続き)

### section と sections [11.3] [2.10.1]
チーム内のスレッド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 and sections [nowait]
```

節:
- **allocate** ⑨
- **firstprivate** (リスト) ⑨
- **lastprivate** (*[lastprivate* 修飾子*:]* リスト) ⑨
- **nowait** ⑨
- **private** (リスト) ⑨
- **reduction** ⑨

### do と for [11.5.1-2] [2.11.4]
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] ...]
    入れ子のループ
```

For
```
!$omp do [節[ [, ]節] ...]
    入れ子のループ
!$omp end do [nowait]
```

節:
- **allocate** ⑨
- **collapse** (*n*) ⑨
- **firstprivate** (リスト) ⑨
- **lastprivate** (*[lastprivate* 修飾子*:]* リスト) ⑨
- **linear** (リスト *[:* リニアステップ*]*) ⑨
- **nowait** ⑨
- **order** ([order 修飾子] **concurrent**) ⑨
  order 修飾子: **reproducible**、**unconstrained**
- **ordered** *[(n)]*
  ループまたは構造に関連付けるループの数。
- **private** (リスト) ⑨
- **reduction** ⑨
- **schedule**(*[*修飾子 *[,*修飾子*]:] kind[,*チャンクサイズ*]*)

***schedule*** *kind* の値:
- **static:** 反復は同一のチャンクサイズに分割され、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内のスレッドへ振り分けられます。
- **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。

***schedule*** 修飾子の値:
- **monotonic:** 各スレッドは、論理的な反復順序の昇順で割り当てられたチャンクを実行します。**schedule(static)** 節または **order** 節は、monotonic であることを意味します。
- **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。
- **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 (*simd_width* は実装依存です)。

### distribute [11.6] [2.11.6.1]
初期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] ...]
    入れ子のループ
```

For
```
!$omp distribute [節[ [, ]節] ...]
    入れ子のループ
[!$omp end distribute]
```

節:
- **allocate** ⑨
- **collapse** (*n*) ⑨
- **dist_schedule** (*kind [,* チャンクサイズ*]*)
- **firstprivate** (リスト) ⑨
- **lastprivate** (リスト) ⑨
- **linear** (リスト *[:* リニアステップ*]*) ⑨
- **order** (*[* order 修飾子 : *]* **concurrent**) ⑨
  order 修飾子: **reproducible**、**unconstrained**
- **private**(リスト) ⑨

### loop [11.7] [2.11.7]
関連するループ反復が同時に実行される可能があり、到達するスレッドがそれに応じて実行できることを示します。

C/C++
```
#pragma omp loop [節[ [, ]節] ...]
    入れ子のループ
```

For
```
!$omp loop [節[ [, ]節] ...]
    入れ子のループ
[!$omp end loop]
```

節:
- **bind** (*binding*)
  *binding*: **teams**、**parallel**、**thread**
- **collapse** (*n*) ⑨
- **lastprivate** (リスト) ⑨
- **order** (*[order* 修飾子 *: ]* **concurrent**) ⑨
  order 修飾子: **reproduce**、**unconstrained**
- **private** (リスト) ⑨
- **reduction** ⑨

## タスク構造

### task [12.5] [2.12.1]
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、データごとの環境 ICV、および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
```
!$omp task [節[ [, ]節] ...]
    緩い構造化ブロック
!$omp end task
 - または –
!$omp task [節[ [, ]節] ...]
    厳密な構造化ブロック
[!$omp end task]
```

節:
- **affinity** (*[aff* 修飾子*:] locator*-リスト)
  *aff* 修飾子*:* **iterator** (イテレーター定義)
- **allocate** ⑨
- **default** (データ共有属性) ⑨
- **detach** (*event* ハンドル)
  指定されるイベントが実行されるまで、タスクは完了しません (omp_fulfilled_event も参照)。
  *event* ハンドル:
  *C/C++* **omp_event_handle_t** タイプ
  *For* **kind omp_event_handle_kind**
- **if** ([**task**:] omp 論理式) ⑨
- **in_reduction** (*reduction* 識別子*:* リスト) ⑨
- **final** (omp 論理式)
  式が true と判断されると、生成されたタスクは最終タスクになります。
- **firstprivate** (リスト) ⑨
- **mergeable**
- **priority** (優先度の値)
  ランタイムにヒントを与え、最優先の値を設定します。
- **private** (リスト) ⑨ **shared**(リスト) ⑨
- **untied**
  タスクは untied タスクです。チーム内のいずれのスレッドも一時停止後にタスク領域を再開できます。

### taskloop [12.6] [2.12.2]
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] ...]
    入れ子のループ
```

For
```
!$omp taskloop [節[ [, ]節] ...]
    入れ子のループ
[!$omp end taskloop]
```

節:
- **allocate** ⑨
- **collapse** (*n*) ⑨
- **default** (データ共有属性) ⑨
- **final** (omp 論理式)
  式が true と判断されると、生成されたタスクは最終タスクになります。
- **firstprivate** (リスト) ⑨
- **grainsize** (*[**strict**:]* グレインサイズ式)
  各タスクに割り当てる論理ループ反復回数を、グレインサイズ式の値と論理ループの反復数の最小値以上、かつグレインサイズ式の 2 倍未満になるようにします。最後の反復を除いて、**strict** は厳密な粒度を強制します。
- **if**(*[***taskloop***:]* omp 論理式) ⑨
- **in_reduction** (*reduction* 識別子*:* リスト) ⑨
- **lastprivate** (リスト) ⑨
- **mergeable**
- **nogroup**
  暗黙の taskgroup の生成を防ぎます。
- **num_tasks** (*[strict:]* タスク数)
  タスク数の最小数と論理ループの反復回数と同数のタスクを生成します。strict は、正確にタスク数を生成することを強制します。
- **priority** (優先度の値)
  ランタイムにヒントを与え、最優先の値を設定します。省略されると優先度はゼロ (最低) です。
- **private** (リスト) ⑨
- **reduction** ⑨
- **shared** (リスト) ⑨
- **untied**
  生成されたタスクは untied タスクです。チーム内のいずれのスレッドも一時停止後にタスク領域を再開できます。

### taskyield [12.7] [2.12.4]
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
```

## デバイス・ディレクティブと構造

### target data [13.5] [2.14.2]
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
```
!$omp target data 節 [[[,] 節]…]
    緩い構造化ブロック
!$omp end target data
 -または-
!$omp target data 節 [[[,] 節]…]
    厳密な構造化ブロック
[!$omp end target data]
```

節:
- **device** (omp 整数式) ⑨
- **if** (*[***target data***:]* omp 論理式) ⑨
- **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**:** *]* リスト) ⑨
- **use_device_ptr** (リスト)
- **use_device_addr** (リスト)

## ディレクティブと構造 (続き)

### target enter data [13.6] [2.14.3]
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:
**depend** (*[依存関係修飾子,]* 依存関係タイプ *: locator* リスト) ⑨
**device** (omp 整数式) ⑨
**if**(*[***target data***:]* omp 論理式) ⑨
**map**(*[[*マップ修飾子, *[*マップ修飾子, *...]]*
マップタイプ**:** *]* リスト) ⑨
**nowait** ⑨

### target exit data [13.7] [2.14.4]
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: **target enter data** で利用できるすべての節。
**map** 節の例外を参照してください。

### target [13.8] [2.14.5]
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] ...]*<br>構造化ブロック |
| For | **!$omp target** *[節[ [, ]節] ...]*<br>緩い構造化ブロック<br>**!$omp end target**<br>- または -<br>**!$omp target** *[節[ [, ]節] ...]*<br>厳密な構造化ブロック<br>*[***!$omp end target***]* |

節:
**allocate** ⑨
アロケーター:
*C/C++* **omp_allocator_handle_t** タイプの識別子
*For kind* **omp_allocator_handle_kind** の整数式
**defaultmap** (暗黙の動作 *[:*変数カテゴリー*]*)
暗黙の動作: **alloc**、**default**、**firstprivate**、**from**、**none**、**present**、**to**、**tofrom**
変数カテゴリー: **aggregate**、**all**、**pointer**、**scalar**、*For* **allocatable**
**depend** (*[*依存関係修飾子, *]* 依存関係タイプ **:** *locator* リスト) ⑨
**device** (*[device* 修飾子*:] omp* 整数式) ⑨
*device* 修飾子*:* **ancestor**、**device_num**
**firstprivate** (リスト) ⑨
**has_device_addr** (リスト)

**if** (*[* **target :** *]* omp 論理式) ⑨
**in_reduction** (*reduction* 識別子 : リスト) ⑨
**is_device_ptr** (リスト)
リスト項目はデバイスアドレスに含まれており、ターゲットデバイスから直接アクセスできることを示します。配列セクションが含まれる場合もあります。
**map**(*[[*マップ修飾子, *[*マップ修飾子, *...]]*
マップタイプ**:** *]* リスト) ⑨
**nowait** ⑨
**private** (リスト) ⑨
**thread_limit** (omp 整数式)
**uses_allocators** (*[[alloc-mod ,] alloc-mod]***:** *allocator*)
ディレクティブに関連する領域で、指定するアロケーターを使用できるようにします。
*alloc-mod:*
***memspace*** (*mem-space-handle*)
**traits** (*traits-array*)
*mem-space-handle:*
*C/C++* ***memspace_handle_t*** *タイプの変数*
*For* ***memspace_handle_kind*** *kind の整数*
*traits-array:* 各タイプの特性の定数配列
*C/C++* **omp_alloctrait_t**
*For* **type** (**omp_alloctrait**)

### target update [13.9] [2.14.6]
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:
**nowait** ⑨
**depend** (*[*依存関係修飾子, *]* 依存関係タイプ **:** *locator* リスト) ⑨
**device** (omp 整数式) ⑨
**from** (*[*モーション修飾子*[,][*モーション修飾子*[,]…]:]*
*locator* リスト)
モーション修飾子: **present**、
**mapper** (マッパー識別子)、**iterator** (*iterators* 定義)
**if** (*[* **target update** : *]* omp 論理式) ⑨
**device** (スカラー整数式)
**to** (*[*モーション修飾子*[,][*モーション修飾子*[,]…]:]*
*locator* リスト)
モーション修飾子: **present**、
**mapper** (マッパー識別子)、**iterator** (*iterators* 定義)

## 相互運用構造

### interop [14.1] [2.15.1]
OpenMP 実装から相互運用プロパティーを取得して、外部実行コンテキストとの相互運用を有効にします。

| | |
|---|---|
| C/C++ | **#pragma omp interop** 節 *[[[, ]節] ...]* |
| For | **!$omp interop** 節 *[[[, ]節] ...]* |

節:
**device** (omp 整数式) ⑨
**depend** (*[*依存関係修飾子*,]* 依存関係タイプ *: locator* リスト) ⑨
**destroy** (*interop-var*)
**init** (*[interop* 修飾子*,] [interop* タイプ*,] interop* タイプ*: interop-var*)
interop 修飾子:
**prefer_type** (プリファレンス・リスト)
interop タイプ: **target**、**targetsync**
interop タイプは 2 つしかありません。
**nowait** ⑨
**use** (*interop-var*)

## 同期構造

### critical [15.2] [2.19.1]
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)]*** *[[,]* **¥**<br>**hint** *(hint* 式*)]]*<br>構造化ブロック |
| For | **!$omp critical** *[(name)] [[,]* **hint** *(hint* 式*)]]*<br>緩い構造化ブロック<br>**!$omp end critical *[(name)]***<br>- または -<br>**!$omp critical** *[(name)] [[,]* **hint** *(hint* 式*)]]*<br>厳密な構造化ブロック<br>**!$omp end critical *[(name)]*** |

*hint* 式:
**omp_sync_hint_uncontended**
**omp_sync_hint_contended**
**omp_sync_hint_speculative**
**omp_sync_hint_nonspeculative**

### barrier [15.3.1] [2.19.2]
チーム内のすべてのスレッドがバリアに到達するまでチーム内のどのスレッドもバリアの先に進めないように明示的なバリアを指定します。

| | |
|---|---|
| C/C++ | **#pragma omp barrier** |
| For | **!$omp barrier** |

### taskgroup [15.4] [2.19.6]
領域の動的スコープ内で生成されたタスクの子孫がすべて完了するまで待機することを指示します。

| | |
|---|---|
| C/C++ | **#pragma omp taskgroup** *[節[ [,] 節] ⋯]*<br>構造化ブロック |
| For | **!$omp taskgroup** *[節[ [,] 節] ⋯]*<br>緩い構造化ブロック<br>**!$omp end taskgroup**<br>- または -<br>**!$omp taskgroup** *[節[ [,] 節] ⋯]*<br>厳密な構造化ブロック<br>*[***!$omp end taskgroup***]* |

節:
**allocate** ⑨
**task_reduction** (*reduction* 識別子 : リスト)
*reduction* 識別子 : **reduction** を参照 ⑨

### taskwait [15.5] [2.19.5]
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** (*[*依存関係修飾子, *]* 依存関係タイプ **:** *locator* リスト) ⑨
**nowait** ⑨

### flush [15.8.5] [2.19.8]
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*] [(*リスト*)]* |
| For | **!$omp flush** *[*メモリーオーダー節*] [(*リスト*)]* |

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

### depobj [15.9.4] [2.19.10.1]
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 (**依存オブジェクト**)** 節 |
| For | **!$omp depobj (**依存オブジェクト**)** 節 |

節*:*
**depend** (依存関係タイプ: *locator*) ⑨
**destroy** (依存オブジェクト)
**update** (タスク依存関係タイプ)
OpenMP 依存オブジェクトの依存関係タイプをタスク依存関係タイプに設定します。
タスク依存関係タイプ:
**in**、**out**、**inout**、**inoutset**、**mutexinoutset**

# ディレクティブと構造 (続き)

## atomic [15.8.4] [2.19.7]

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。

C/C++

```
#pragma omp atomic [節[[,] 節] ...]
    構造化ブロック
```

For

```
!$omp atomic [節[[[,] 節] ... ]]
    文
[!$omp end atomic]
 - または -
!$omp atomic [節[[[,] 節] ... ] [,]] capture &
      [[,] 節 [[[,] 節] ...]]
    文
    capture 文
[!$omp end atomic]
 - または -
!$omp atomic [節[[[,] 節] ... ] [,]] capture &
      [[,] 節 [[[,] 節] ...]]
    capture 文
    文
[!$omp end atomic]
```

節:

*atomic* 節: : **read**、**write**、**update**

メモリーオーダー節:

**seq_cst**、**acq_rel**、**release**、**acquire**、**relaxed**、**acquire**、**relaxed**

拡張 *atomic:* **capture**、**compare**、**fail**、**weak**

**capture:** アトミックに更新される変数値をキャプチャーします。

**compare**: 条件付きアトミック更新を行います。

**fail** (*seq_cst* | *acquire* | *relaxed*): 失敗した条件付きアトミック更新で比較されるメモリー順序要件を指定します。引数は、他に指定されるメモリー順序をオーバーライドします。

**weak**: 条件付きアトミック更新が行われる比較は、値が等しい場合でも等しくないと評価され、偽の失敗をもたらす可能性があることを指定します。

***hint*** (ヒント式)

式文: C/C++

| 節 | 式文 |
|---|---|
| **read** | *v = x;* |
| **write** | *x = expr;* |
| **update** | *x++; x--; ++x; --x;*<br>*x binop= expr; x = x binop expr;*<br>*x = expr binop x;* |
| **compare** | *cond-expr-stmt:*<br>*x = expr ordop x ? expr : x;*<br>*x = x ordop expr ? expr : x;*<br>*x = x == e ? d : x;*<br>*cond-update-stmt:*<br>***if**(expr ordop x) { x = expr; }*<br>***if**(x ordop expr) { x = expr; }*<br>***if**(x == e) { x = d; }* |
| **capture** | *v = expr-stmt*<br>*{ v = x; expr-stmt }*<br>*{ expr-stmt v = x; }*<br>(where *expr-stmt* is either *write-expr-stmt, update-expr-stmt, or cond-expr-stmt.*) |
| **compare** と **capture** | *{ v = x; cond-update-stmt }*<br>*{ cond-update-stmt v = x; }*<br>*if(x == e) { x = d; } else { v = x; }*<br>*{ r = x == e; if(r) { x = d; } }*<br>*{ r = x == e; if(r) { x = d; } else { v = x; } }* |

## atomic の続き

capture 文: *v = x*。 For

式文: For

| 節 | 式文 |
|---|---|
| **read** | *v = x;* |
| **write** | *x = expr;* |
| **update** | *x = x operator expr*<br>*x = expr operator x*<br>*x = intrinsic_procedure_name (x, expr-list)*<br>*x = intrinsic_procedure_name (expr-list, x)* |
| *intrinsic_procedure_name :* **MAX**、**MIN**、**IAND**、**IOR**、**IEOR** | |
| *operator* は、**+**、*****、**/**、**.AND.**、**.OR.**、**.EQV.**、**.NEQV** のいずれか。 | |
| **compare** がある場合 | **if** (*x* == *e*) **then**<br>  *x = d*<br>**end if**<br><br>**if** (*x* == *e*) *x = d* |
| **compare** 節と **capture** 節があり、文の前後に capture 文がない場合 | **if** (*x* == *e*) **then**<br>  *x = d*<br>**else** *v = x*<br>**end if** |

## ordered [15.10.2] [2.19.9]

並列ループ内でループの反復順に実行される構造化ブロックを指定したり、doacross 入れ子のループで反復間の依存関係を指定します。

C/C++

```
#pragma omp ordered [節[[, ] 節]]
    構造化ブロック
    または
#pragma omp ordered  節 [[[, ] 節] ...]
```

For

```
!$omp ordered [節[[, ]節]]
    緩い構造化ブロック
!$omp end ordered]
    または
!$omp ordered  [節[[, ]節]]
    厳密な構造化ブロック
[!$omp end ordered]
    または
!$omp ordered 節[[[,] 節] ...]
```

節 (構造化ブロック形式のみ):

**threads**

**simd**

**threads** または **simd** は、構造に関連付ける並列化レベルを指定します。

節 (スタンドアロン形式のみ):

**doacross** (依存関係タイプ: [vector])

ループ反復のスケジュールに追加の制約があることを意味する doacross (相互反復依存関係) を識別します。

依存関係タイプ:

**source**

現在の反復から生じる相互反復依存を指定します。**source** が指定される場合、***vector*** 引数はオプションです。***vector*** が省略されると **omp_cur_iteration** と見なされます。依存関係タイプそして **source** を使用するディレクティブでは、最大 1 つの **doacross** 節を指定できます。

**sink**

相互反復依存関係を指定します。ここで、***vector*** は依存関係を満たす反復をします。***vector*** が反復空間で発生しない場合、**doacross** 節は無視されます。ordered 構造のすべての **doacross** 節が無視されると、その構造は無視されます。

## キャンセル構造

### cancel [16.1] [2.20.1]

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]
```

if 節: **if** ([ **cancel** : ] *omp* 論理式)

構造タイプ節:

*C/C++* **parallel**、**sections**、**taskgroup**、**for**

*For* **parallel**、**sections**、**taskgroup**、**do**

### cancellation point [16.2] [2.20.2]

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:

**parallel**

**sections**

**taskgroup**

*C/C++* **for**

*For* **do**

## 結合構造とディレクティブ

次の結合構造とディレクティブは、OpenMP API バージョン 5.2 仕様の 17 章で定義されるパラメーターに従って作成され、以前のバージョンでは明示的に定義されていました。

### do simd と for simd [17] [2.11.5.2]

関連するループの反復がチーム内のスレッドで並列に実行でき、各スレッドで実行される反復が SIMD 命令を使用して同時に実行できるループに適用します。

C/C++

```
#pragma omp for simd [節[ [, ]節] ...]
    入れ子のループ
```

For

```
!$omp do simd [節[ [, ]節] ...]
    入れ子のループ
[!$omp end do simd [nowait]]
```

節: **simd** 、**for** もしくは **do** ディレクティブと同一の節が適用されます。

### distribute simd [17] [2.11.6.2]

チーム領域のプライマリー・スレッドに分散され、SIMD 命令を使用して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] ...]
    入れ子のループ
```

For

```
!$omp distribute simd [節[ [, ]節] ...]
    入れ子のループ
[!$omp end distribute simd]
```

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

### distribute parallel do と distribute parallel for [17] [2.11.6.3]

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] ...]
    入れ子のループ
```

For

```
!$omp distribute parallel do [節[ [, ]節] ...]
    入れ子のループ
[!$omp end distribute parallel do]
```

節: **distribute**、**parallel for** および **parallel do** と同一の節が適用されます。

## ディレクティブと構造 (続き)

### distribute parallel do simd と distribute parallel for simd [17] [2.11.6.4]

複数のチームのメンバーである複数のスレッドで SIMD 命令を使用し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 simd ¥
                                    [節[ [, ]節] ...]
   入れ子のループ
```

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

節: **distribute**、**parallel for simd** および **parallel do simd** ディレクティブと同一の節が適用されます。

### taskloop simd [17] [2.12.3]

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に実行できることを示します。

C/C++
```
#pragma omp taskloop simd [節[ [, ]節] ...]
   入れ子のループ
```

For
```
!$omp taskloop simd [節[ [, ]節] ...]
   入れ子のループ
[!$omp end taskloop simd]
```

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

### parallel do と parallel for [17] [2.16.1]

1 つ以上の関連するループだけを含み、その他の文を含まないワークシェア・ループ構造を含む **parallel** 構造を指定します。

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

For
```
!$omp parallel do [節[ [, ]節] ...]
   入れ子のループ
[!$omp end parallel do]
```

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

### parallel loop [17] [2.16.2]

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] ...]
   入れ子のループ
```

For
```
!$omp parallel loop [節[ [, ]節] ...]
   入れ子のループ
[!$omp end parallel loop]
```

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

### parallel sections [17] [2.16.3]

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
   ...
   }
```

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が適用されます [C/C++ では **nowait** 節を除きます]。

### parallel workshare [17] [2.16.4]

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] ...]
   緩い構造化ブロック
!$ omp end parallel workshare
 - または -
!$omp parallel workshare [節[ [, ]節] ...]
   厳密な構造化ブロック
[!$omp end parallel workshare]
```

節: **parallel** ディレクティブと同一の節が適用されます。

### parallel for simd と parallel do simd [17] [2.16.5]

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

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

For
```
!$omp parallel do simd [節[ [, ]節] ...]
   入れ子のループ
[!$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が適用されます (C/C++ では **nowait** 節を除きます)。

### parallel masked [17] [2.16.6]

1 つの **masked** 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 masked [節[ [, ]節] ...]
   構造化ブロック
```

For
```
!$omp parallel masked [節[ [, ]節] ...]
   緩い構造化ブロック
!$omp end parallel masked
 - または -
!$omp parallel masked [節[ [, ]節] ...]
   厳密な構造化ブロック
[!$omp end parallel masked]
```

節: **parallel** または **masked** ディレクティブと同一の節が適用されます。

### masked taskloop [17] [2.16.7]

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

C/C++
```
#pragma omp masked taskloop [節[ [, ]節] ...]
   入れ子のループ
```

For
```
!$omp master taskloop [節[ [, ]節] ...]
   入れ子のループ
[!$omp end masked taskloop]
```

節: **masked** または **taskloop** ディレクティブと同一の節が適用されます。

### masked taskloop simd [17] [2.16.8]

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

C/C++
```
#pragma omp masked taskloop simd ¥
                                    [節[ [, ]節] ...]
   入れ子のループ
```

For
```
!$omp masked taskloop simd [節[ [, ]節] ...]
   入れ子のループ
[!$omp end master taskloop simd]
```

節: **masked** または **taskloop simd** ディレクティブと同一の節が適用されます。

### parallel masked taskloop [17] [2.16.9]

1 つの **masked 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 masked taskloop ¥
                                    [節[ [, ]節] ...]
   入れ子のループ
```

For
```
!$omp parallel masked taskloop [節[ [, ]節] ...]
   入れ子のループ
[!$omp end parallel masked taskloop]
```

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

### parallel masked taskloop simd [17] [2.16.10]

1 つの **masked 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 masked taskloop simd ¥
                                    [節[ [, ]節] ...]
   入れ子のループ
```

For
```
!$ omp parallel masked taskloop simd
                                    [節[ [, ]節] ...]
   入れ子のループ
[!$ omp end parallel masked taskloop simd]
```

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

### teams distribute [17] [2.16.11]

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] ...]
   入れ子のループ
```

For
```
!$omp teams distribute [節[ [, ]節] ...]
   入れ子のループ
[!$omp end teams distribute ]
```

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

### teams distribute simd [17] [2.16.12]

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] ...]
   入れ子のループ
```

For
```
!$omp teams distribute simd [節[ [, ]節] ...]
   入れ子のループ
[!$omp end teams distribute simd ]
```

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

### teams distribute parallel do と teams distribute parallel for [17] [2.16.13]

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] ...]
   入れ子のループ
```

For
```
!$omp teams distribute parallel do
                                    [節[ [, ]節] ...]
   入れ子のループ
[!$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が適用されます。

### teams distribute parallel do simd と teams distribute parallel for simd [17] [2.16.14]

1 つの **distribute parallel for simd** 構造または **distribute parallel do simd** 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] ...]
   入れ子のループ
```

For
```
!$omp teams distribute parallel do simd
                                    [節[ [, ]節] ...]
   入れ子のループ
[!$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が適用されます。

### teams loop [17] [2.16.15]

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] ...]
   入れ子のループ
```

For
```
!$omp teams loop [節[ [, ]節] ...]
   入れ子のループ
[!$omp end teams loop]
```

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

## ディレクティブと構造 (続き)

### target parallel [17] [2.16.16]

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
```
!$omp target parallel [節[ [, ]節] ...]
    緩い構造化ブロック
!$omp end target parallel
```
- または -
```
!$omp target parallel [節[ [, ]節] ...]
    厳密な構造化ブロック
[!$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が適用されます。

### target parallel do と target parallel for [17] [2.16.17]

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] ...]
    入れ子のループ
```

For
```
!$omp target parallel do [節[ [, ]節] ...]
    入れ子のループ
[!$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が適用されます。

### target parallel do simd と target parallel for simd [17] [2.16.18]

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] ...]
    入れ子のループ
```

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

節: **copyin** 節を除く、**target** または **parallel for/do simd** ディレクティブと同一の節が適用されます。

### target parallel loop [17] [2.16.19]

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] ...]
    入れ子のループ
```

For
```
!$omp target parallel loop [節[ [, ]節] ...]
    入れ子のループ
[!$omp end target parallel loop]
```

節: **copyin** 節を除く、**target** または **parallel loop** ディレクティブと同一の節が適用されます。

### target simd [17] [2.16.20]

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] ...]
    入れ子のループ
```

For
```
!$omp target simd [節[ [, ]節] ...]
    入れ子のループ
[!$omp end target simd]
```

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

### target teams [17] [2.16.21]

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
```
!$omp target teams [節[ [, ]節] ...]
    緩い構造化ブロック
!$omp end target teams
```
- または -
```
!$omp target teams [節[ [, ]節] ...]
    厳密な構造化ブロック
[!$omp end target teams]
```

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

### target teams distribute [17] [2.16.22]

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] ...]
    入れ子のループ
```

For
```
!$omp target teams distribute [節[ [, ]節] ...]
    入れ子のループ
[!$omp end target teams distribute]
```

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

### target teams distribute simd [17] [2.16.23]

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] ...]
    入れ子のループ
```

For
```
!$omp target teams distribute simd
      [節[ [, ]節] ...]
    入れ子のループ
[!$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が適用されます。

### target teams loop [17] [2.16.24]

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] ...]
    入れ子のループ
```

For
```
!$omp target teams loop [節[ [, ]節] ...]
    入れ子のループ
[!$omp end target teams loop]
```

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

### target teams distribute parallel do と target teams distribute parallel for [17] [2.16.25]

**teams distribute parallel do または teams distribute parallel for** 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] ...]
    入れ子のループ
```

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

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

### target teams distribute parallel do simd と target teams distribute parallel for simd [17] [2.16.26]

**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] ...]
    入れ子のループ
```

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

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

メモ

# 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。

## データ共有節 [5.4] [2.21.4]

追加のデータ共有属性は、is_device_ptr、use_device_ptr、has_device_addr、および use_device_addr です。

これらの節については、使用可能なディレクティブで説明されています。

**default** (**shared | none | private | none**)
デフォルトのデータ共有属性は無効化されています。構造内のすべての変数は、構造内で宣言するかデータ共有属性に含める必要があります。

関連: parallel (3), task (4), taskloop (4), teams (3)

**shared** (リスト)
リスト項目の変数は、構造を実行するスレッドまたは明示的なテスク間で共有されます。

関連: parallel (3), task (4), taskloop (4), teams (3)

**private** (リスト)
各スレッドまたは明示的なタスクにプライベートであるリスト項目の変数を作成します。プライベート変数には初期値は割り当てられません。

関連: distribute (4), do と for (4), loop (4), parallel (3), scope (3), section (4), simd (3), single (3), target (5), task (4), taskloop (4), teams (3)

**firstprivate** (リスト)
リスト項目を各スレッドまたは明示的な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項目の値で初期化します。

関連: distribute (4), do と for (4), parallel (3), scope (3), section (4), simd (3), target (5), task (4), taskloop (4), teams (3)

**lastprivate** (*[lastprivate* 修飾子*:]* リスト)
最後のループが終了すると、リスト項目の変数がプライマリー・スレッドにコピーされます。
lastprivate 修飾子: **conditional**
**conditional** は、インデックスの反復回数が最も多いスレッドの値を使用します。

関連: distribute (4), do と for (4), loop (4), section (4), simd (3), taskloop (4)

**linear** (リニアリスト *[:* リニアステップ*]*)
**linear** (リニアリスト *[:* リニア修飾子*] [,*リニア修飾子*]])*
各リニアリストの項目が、ループ反復空間に対しリニア値またはアドレスを持つように宣言します。
リニアリスト: リスト (または declare simd 引数リスト)
リニア修飾子: **step** (リニアステップ)、リニアタイプ修飾子
リニアステップ: OpenMP 整数式 (デフォルトは 1)
リニアタイプ修飾子: **ref**、**val**、**uval** (デフォルトは **val** )
**val**: 値はリニア
**ref**: アドレスはリニア (C++ と Fortran *のみ*)
**uval**: 値はリニアで変更不可 (C++ と Fortran *のみ*)

**ref** および **uval** 修飾子は、**declare simd** ディレクティブの **linear** 節、および参照によって渡される引数に対してのみ指定できます。

関連: declare simd (2), distribute (4), do と for (4), simd (3)

## allocate 節 [6.6] [2.13.4]

**allocate** (*[allocator:]* リスト)
**allocate** (*allocate* 修飾子 *[, allocate* 修飾子 *:]* リスト)
*allocate* 修飾子*:*
***allocator*** *(allocator)*
*allocator* は次のいずれか:
*C/C++* **omp_allocator_handle_t** タイプ
*For* **omp_allocator_handle_kind** *kind*
**align** (アライメント)
アライメントは 2 の正の整数乗の定数。

関連: distribute (4), do と for (4), parallel (3), scope (3), section (4), single (3), target (5), task (4), taskloop (4), teams (3)

## collapse 節 [4.4.3]

**collapse**(*n*)
構造に関連付けるループの入れ子数を指定する正の定数整数式。

関連: distribute (4), do と for (4), loop (4), simd (3), taskloop (4)

## depend 節 [15.9.5] [2.19.11]

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**(*[*依存関係修飾子*,]* 依存関係タイプ **:** *locator* リスト)

依存関係修飾子*:* **iterator** (iterators 定義)
依存関係タイプ*:* **in**、**out**、**inout**、**mutexinoutset**、**inoutset**、**depobj**

- **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**、**inout** または **inoutset** 依存関係タイプリストの 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**、または **inoutset** *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** 依存関係タイプを持つ **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。
- **inoutset:** リスト項目の少なくとも 1 つのストレージの場所が、兄弟タスクが以前に生成したコンテキストの **in**、**out**、**inout**、または **mutexinoutset** 依存関係タイプを持つ **depend** 節に指定されるリスト項目と一致する場合、生成されたタスクはその兄弟タスクの依存関係タスクになります。
- **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

関連: depobj (5), dispatch (2), interop (5), target (5), target enter data (5), target exit data (5), target update (5), task (4), taskwait (5)

## device 節 [13.2]

**device** (*[***ancestor** | **device_num***:]* *device-description*)
デバイス構造に関連するターゲットデバイスを識別します。

devcice-description: デバイス番号を参照する整数タイプの式、または **ancestor** 修飾子がある場合は 1 。

関連: dispatch (2), interop (5), target (5), target data (4), target enter data (5), target exit data (5), target update (5)

## if 節 [3.4] [2.18]

**if** 節の作用は適用される構造に依存します。結合もしくは複合構造においては、ディレクティブ名修飾子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** (*[*ディレクティブ名修飾子 *:]* *omp* 論理式)

関連: cancel (6), parallel (3), simd (3), target (5), target data (4), target enter data (5), target exit data (5), target update (5), taskloop (4), teams (3)

## map 節 [5.8.3] [2.21.7.1]

**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 *: ]* *locator* リスト)

タスク環境からデバイス環境へデータをマップします。

マップタイプ*:* **alloc**, **to**, **from**, **tofrom**, **release**, **delete**

**target** または **target data** ディレクティブ:
マップタイプ*:* **alloc**, **to**, **from**, **tofrom**, **release**

**target enter data** ディレクティブ:
マップタイプ*:* **alloc**, **to**, **from**, **tofrom**

**target exit data** ディレクティブ:
マップタイプ*:* **to**, **from**, **tofrom**, **release**, **delete**

マップタイプ修飾子: **always**, **close**, **present**, **mapper** (マッパー識別子), **iterator** (iterators 定義)

関連: declare mapper (1), target (5), target data (4), target enter data (5), target exit data (5)

## order 節 [10.3] [2.11.3]

**order** (*[order* 修飾子 **: ]** **concurrent**)

*order* 修飾子*:* **reproducible**, **unconstrained**

ループ関連ディレクティブの関連するループ反復で予測される実行順序を指定します。

関連: distribute (4), do と for (4), loop (4), simd (3)

## nowait 節 [15.6]

**nowait**

構造の最後で発生する同期をオーバーライドします。また、相互運用性要件セットに nowait プロパティーが含まれるよう指定できます。構造に暗黙のバリアがある場合、nowait 節はバリアを無視します。

関連: dispatch (2), do と for (4), interop (5), scope (3), section (4), single (3), target (5), target enter data (5), target exit data (5), target update (5), taskwait (5), workshare (3)

## reduction 節 [5.5.8] [2.21.5.4]

**reduction** (*[reduction* 修飾子 *,] reduction* 識別子 **:** リスト)
*reduction* 識別子と 1 つ以上のリスト項目を指定します。

*C++ reduction* 識別子:
*id-expression* または次の演算子を指定します:
**+**, *****, **&**, **|**, **^**, **&&**, **||**
*C reduction* 識別子:
*identifier* または次の演算子を指定します:
**+**, *****, **&**, **|**, **^**, **&&**, **||**
*For reduction* 識別子:
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**

関連: do と for (4), loop (4), parallel (3), scope (3), section (4), simd (3), taskloop (4), teams (3)

## イテレーター [3.2.6] [2.1.6]

節の中で複数の値に展開される識別子。
**iterator** (*iterators* 定義)
*iterators* 定義*:*
*iterator* 指定子 *[, iterators* 定義 *]*
*iterators* 指定子*:*
*[ iterator* タイプ *]* *identifier* **=** *range* 指定子
*identifier*: ベース言語の識別子
*range* 指定子: *begin* **:** *end [: step]*
*begin*, *end*: 型を *iterator* タイプに変換できる式
*step*: 整数式
*iterator* タイプ: *C/C++* タイプ名。*For* タイプ指定子。

# ランタイム・ライブラリー・ルーチン

## スレッド・チーム・ルーチン

**omp_set_num_threads [18.2.1] [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 [18.2.2] [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 [18.2.3] [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 [18.2.4] [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_in_parallel [18.2.5] [3.2.5]**
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** ()

**omp_set_dynamic [18.2.6] [3.2.6]**
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 [18.2.7] [3.2.7]**
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 [18.2.8] [3.2.8]**
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_schedule [18.2.11] [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 [18.2.12] [3.2.12]**
**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次のいずれかです:
**omp_sched_static**
**omp_sched_dynamic**
**omp_sched_guided**
**omp_sched_auto**
+ 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 [18.2.13] [3.2.13]**
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 [18.2.14] [3.2.14]**
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_active_levels** ()

**omp_set_max_active_levels [18.2.15] [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_max_active_levels** (*max_levels*)
**logical** *nested*

**omp_get_max_active_levels [18.2.16] [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_max_active_levels** ()

**omp_get_level [18.2.17] [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 [18.2.18] [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_num** (*levels*)
**integer** *levels*

**omp_get_team_size [18.2.19] [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 [18.2.20] [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** ()

## スレッド・アフィニティー・ルーチン

**omp_get_proc_bind [18.3.1] [3.3.1]**
**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_primary, omp_proc_bind_close, omp_proc_bind_spread**

**omp_get_num_places [18.3.2] [3.3.2]**
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 [18.3.3] [3.3.3]**
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_num_procs** (*place_num*)
**integer** *place_num*

**omp_get_place_proc_ids [18.3.4] [3.3.4]**
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 [18.3.5] [3.3.5]**
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** ()

# ランタイム・ライブラリー・ルーチン (続き)

### omp_get_partition_num_places [18.3.6] [3.3.6]
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 ()
```

### omp_get_partition_place_nums [18.3.7] [3.3.7]
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 [18.3.8] [3.3.8]
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 [18.3.9] [3.3.9]
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 [18.3.10] [3.3.10]
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 [18.3.11] [3.3.11]
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
```

## チーム領域ルーチン

### omp_get_num_teams [18.4.1] [3.4.1]
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 [18.4.2] [3.4.2]
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_set_num_teams [18.4.3] [3.4.3]
現在のタスクの内部制御変数 nteams-var 値を設定します。**num_teams** 節を指定しない後続のチーム領域のスレッド数に影響します。

C/C++
```
void omp_set_num_teams (int num_teams);
```
For
```
subroutine omp_set_num_teams (num_teams)
integer num_teams
```

### omp_get_max_teams [18.4.4] [3.4.4]
このルーチンから戻った後に到達する **num_teams** 節を持たない **teams** 構造で生成されるチーム数の上限を取得します。

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

### omp_set_teams_thread_limit [18.4.5][3.4.5]
内部制御変数 team-thread-limit-var 値を設定することで、**teams** 構造によって生成される各競合グループに参加できる OpenMP スレッドの最大数を設定します。

C/C++
```
void omp_set_teams_thread_limit
                        (int thread_limit);
```
For
```
subroutine omp_set_teams_thread_limit &
                        (num_ thread_limit)
integer thread_limit
```

### omp_get_teams_thread_limit [18.4.6][3.4.6]
チーム構造によって生成された各競合グループに参加できる OpenMP スレッドの最大数を取得します。

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

## タスクルーチン

### omp_get_max_task_priority [18.5.1] [3.5.1]
**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_in_final [18.5.3] [3.5.2]
**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 ()
```

## リソース解放ルーチン

### omp_pause_resource [18.6.1] [3.6.1]
### omp_pause_resource_all [18.6.2] [3.6.2]
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解放することを許可します。有効な kind 値には、**omp_pause_soft** と **omp_pause_hard** が含まれます。

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
```

## デバイス情報ルーチン

### omp_get_num_procs [18.7.1] [3.7.1]
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_set_default_device [18.7.2] [3.7.2]
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 [18.7.3] [3.7.3]
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 [18.7.4] [3.7.4]
ホスト以外の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 [18.7.5] [3.7.5]
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_is_initial_device [18.7.6] [3.7.6]
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 [18.7.7] [3.7.7]
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 ()
```

## 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 [18.8.1] [3.8.1]
デバイスデータ環境にメモリーを割り当てて、そのメモリーへのデバイスポインターを返します。

C/C++
```
void *omp_target_alloc (
         size_t size, int device_num);
```
For
```
type(c_ptr) function omp_target_alloc ( &
   size, device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
   c_size_t, c_int
integer (c_size_t), value :: size
integer (c_int), value :: device_num
```

# ランタイム・ライブラリー・ルーチン (続き)

**omp_target_free [18.8.2] [3.8.2]**
**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_target_free (
        void *device_ptr, int device_num);
```

For
```
subroutine omp_target_free ( &
  device_ptr, device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
  c_int
type (c_ptr), value :: device_ptr
integer (c_int), value :: device_num
```

**omp_target_is_present [18.8.3] [3.8.3]**
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_target_is_present (
        const void *ptr, int device_num);
```

For
```
integer(c_int) function omp_target_is_present &
  (ptr, device_num) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_int
type(c_ptr), value :: ptr
integer(c_int), value :: device_num
```

**omp_target_is_accessible [18.8.4] [3.8.4]**
特定のデバイスからホストメモリーにアクセスできるかテストします。

C/C++
```
int omp_target_is_accessible (const void *ptr,
    size_t size, int device_num);
```

For
```
integer(c_int) function &
  omp_target_is_accessible( &
  ptr, size, device_num) bind(c)
use, intrinsic :: iso_c_binding, only : &
  c_ptr, c_size_t, c_int
type(c_ptr), value :: ptr
integer(c_size_t), value :: size
integer(c_int), value :: device_num
```

**omp_target_memcpy [18.8.5] [3.8.5]**
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);
```

For
```
integer(c_int) function omp_target_memcpy(&
  dst, src, length, dst_offset, src_offset, &
  dst_device_num, src_device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
  c_int, c_size_t
type(c_ptr), value :: dst, src
integer(c_size_t), value :: length, dst_offset, &
  src_offset
integer(c_int), value :: dst_device_num, &
  src_device_num
```

**omp_target_memcpy_rect [18.8.6] [3.8.6]**
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);
```

For
```
integer(c_int) function &
omp_target_memcpy_rect( &
  dst, src ,element_size, num_dims, volume, &
  dst_offsets, src_offsets, dst_dimensions, &
  src_dimensions, dst_device_num, &
  src_device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
  c_int, c_size_t
type(c_ptr), value :: dst, src
integer(c_size_t), value :: element_size
integer(c_int), value :: num_dims, &
  dst_device_num, src_device_num
integer(c_size_t), intent(in) :: volume(*), &
  dst_offsets(*), src_offsets(*), &
  dst_dimensions(*), src_dimensions(*)
```

**omp_target_memory_async [18.8.7] [3.8.7]**
任意の組み合わせのホストポインターとデバイスポインター間で、非同期にコピーを実行します。

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

For
```
integer(c_int) function &
omp_target_memcpy_async( &
  dst, src, length, dst_offset, src_offset, &
  dst_device_num, src_device_num, &
  depobj_count, depobj_list) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
  c_int, c_size_t
type(c_ptr), value :: dst, src
integer(c_size_t), value :: length, dst_offset, &
  src_offset
integer(c_int), value :: dst_device_num, &
  src_device_num, depobj_count
integer(omp_depend_kind), &
  optional :: depobj_list(*)
```

**omp_target_memcpy_rect_async [18.8.8] [3.8.8]**
ホストポインターとデバイスポインターの任意の組み合わせで非同期にコピーを実行します。

C/C++
```
int omp_target_memcpy_rect_async (
    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, int depobj_count,
    omp_depend_t *depobj_list);
```

For
```
integer(c_int) function &
omp_target_memcpy_rect_async ( &
  dst, src, element_size, num_dims, volume, &
  dst_offsets, src_offsets, dst_dimensions, &
  src_dimensions, dst_device_num, &
  src_device_num, depobj_count, &
  depobj_list) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_int, c_size_t
  type(c_ptr), value :: dst, src
  integer(c_size_t), value :: element_size
  integer(c_int), value :: num_dims, &
  dst_device_num, src_device_num, depobj_count
integer(c_size_t), intent(in) :: volume(*), &
  dst_offsets(*), src_offsets(*), &
  dst_dimensions(*), src_dimensions(*)
integer(omp_depobj_kind), &
  optional :: depobj_list(*)
```

**omp_target_associate_ptr [18.8.9] [3.8.9]**
**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);
```

For
```
integer(c_int) function &
omp_target_associate_ptr(&
  host_ptr, device_ptr, size, device_offset, &
  device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
  c_size_t, c_int
type(c_ptr), value :: host_ptr, device_ptr
integer(c_size_t), value :: size, device_offset
integer(c_int), value :: device_num
```

**omp_target_disassociate_ptr [18.8.10] [3.8.10]**
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);
```

For
```
integer(c_int) function &
omp_target_disassociate_ptr(&
  ptr, device_num) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_int
type(c_ptr), value :: ptr
integer(c_int), value :: device_num
```

**omp_get_mapped_ptr [18.8.11] [3.8.11]**
指定されたデバイスのホストポインターに関連付けられているデバイスポインターを返します。

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

For
```
type(c_ptr) function omp_get_mapped_ptr( &
  ptr, device_num) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_int
type(c_ptr), value :: ptr
integer(c_int), value :: device_num
```

## 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 [18.9.1] [3.9.1]**
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 [18.9.2] [3.9.2]**
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:* [15.1] [2.19.12] を参照

**ロックの破棄 [18.9.3] [3.9.3]**
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 [18.9.4] [3.9.4]**
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 [18.9.5] [3.9.5]**
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 [18.9.6] [3.9.6]**
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
```
subroutine omp_test_lock (svar)
integer (kind=omp_lock_kind) svar

subroutine 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 [18.10.1] [3.10.1]**
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 [18.10.2] [3.10.2]**
**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 [18.11.1] [3.11.1]**
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
```

## 相互運用ルーチン

**omp_get_num_interop_properties [18.12.1] [3.12.1]**
**omp_interop_t** オブジェクトで使用可能な実装定義のプロパティー数を取得します。

C/C++
```
int omp_get_num_interop_properties (
  omp_interop_t interop);
```

**omp_get_interop_int [18.12.2] [3.12.2]**
**omp_interop_t** オブジェクトから整数プロパティーを取得します。

C/C++
```
omp_intptr_t omp_get_interop_int (
  const omp_interop_t interop,
  omp_interop_property_t property_id,
  int *ret_code );
```

**omp_get_interop_ptr [18.12.3] [3.12.3]**
**omp_interop_t** オブジェクトからポインター・プロパティーを取得します。

C/C++
```
void *omp_get_interop_ptr (
  const omp_interop_t interop,
  omp_interop_property_t property_id,
  int *ret_code );
```

**omp_get_interop_str [18.12.4] [3.12.4]**
**omp_interop_t** オブジェクトから文字列プロパティーを取得します。

C/C++
```
const char* omp_get_interop_str (
  const omp_interop_t interop,
  omp_interop_property_t property_id,
  int *ret_code );
```

**omp_get_interop_name [18.12.5] [3.12.5]**
**omp_interop_t** オブジェクトからプロパティー名を取得します。

C/C++
```
const char* omp_get_interop_name (
  omp_interop_t interop,
  omp_interop_property_t property_id);
```

**omp_get_interop_type_desc [18.12.6] [3.12.6]**
**omp_interop_t** オブジェクトに関連付けられているプロパティーのタイプの説明を取得します。

C/C++
```
const char* omp_get_interop_type_desc (
  omp_interop_t interop,
  omp_interop_property_t property_id);
```

**omp_get_interop_rc_desc [18.12.7] [3.12.7]**
**omp_interop_t** オブジェクトに関連付けられている戻りコードの説明を取得します。

C/C++
```
const char* omp_get_interop_rc_desc (
  omp_interop_t ret_code);
```

## メモリー管理ルーチン

**メモリー管理タイプ [18.13.1] [3.13.1]**
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:

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

*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。

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

*omp_atv_X*。ここで X は false、true、default、contended、uncontended、serialized、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 [18.13.2] [3.13.2]**
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 [18.13.3] [3.13.3]**
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 [18.13.4] [3.13.4]**
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 [18.13.5] [3.13.5]**
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 と omp_aligned_alloc [18.13.6] [3.13.6]**
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);

void *omp_aligned_alloc (size_t size,
  omp_allocator_handle_t allocator);
```

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

void *omp_aligned_alloc (size_t size,
  size_t alignment,
  omp_allocator_handle_t allocator
  = omp_null_allocator);
```

For
```
type(c_ptr) function omp_alloc &
  (size, allocator) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_size_t
integer(c_size_t), value :: size
integer(omp_allocator_handle_kind), &
  value :: allocator
type(c_ptr) function omp_aligned_alloc ( &
  alignment, size, allocator) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_size_t
integer(c_size_t), value :: alignment, size
integer(omp_allocator_handle_kind), &
  value :: allocator
```

**omp_free [18.13.7] [3.13.7]**
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);
```

For
```
subroutine omp_free (ptr, allocator) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr
type(c_ptr), value :: ptr
integer(omp_allocator_handle_kind), &
  value :: allocator
```

## ランタイム・ライブラリー・ルーチン (続き)

### omp_calloc と omp_aligned_calloc [18.13.8] [3.13.8]

メモリー・アロケーターにゼロで初期化されたメモリー割り当てを要求します。

C
```
void *omp_calloc (size_t nmemb, size_t size,
  omp_allocator_handle_t allocator);
void *omp_aligned_calloc (size_t alignment,
  size_t nmemb, size_t size,
  omp_allocator_handle_t allocator);
```

C/C++
```
void *omp_calloc (size_t nmemb, size_t size,
  omp_allocator_handle_t allocator
  =omp_null_allocator);

void *omp_aligned_calloc (size_t alignment,
  size_t nmemb, size_t size,
  omp_allocator_handle_t allocator
  = omp_null_allocator);
```

For
```
type(c_ptr) function omp_calloc (&
  nmemb, size, allocator) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_size_t
integer(c_size_t), value :: nmemb, size
integer(omp_allocator_handle_kind), &
  value :: allocator
type(c_ptr) function omp_aligned_calloc ( &
  alignment, nmemb, size, allocator) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_size_t
integer(c_size_t), value :: alignment,  &
  nmemb, size
integer(omp_allocator_handle_kind), &
  value :: allocator
```

### omp_realloc [18.13.9] [3.13.9]

割り当て済みのメモリー割り当てを解放し、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_realloc (void *ptr, size_t size,
  omp_allocator_handle_t allocator,
  omp_allocator_handle_t free_allocator);
```

C/C++
```
void *omp_realloc (void *ptr, size_t size,
  omp_allocator_handle_t allocator
  = omp_null_allocator,
  omp_allocator_handle_t free_allocator
  =omp_null_allocator);
```

For
```
type(c_ptr) function omp_realloc ( &
  ptr, size, allocator, free_allocator) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_size_t
type(c_ptr), value :: ptr
integer(c_size_t), value :: size
integer(omp_allocator_handle_kind), &
  value :: allocator, free_allocator
```

### ツール制御ルーチン

**omp_control_tool [18.14] [3.14]**
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。

### 環境表示ルーチン

**omp_display_env [18.15] [3.15]**
OpenMP のバージョン番号と環境変数に関連付けられている内部制御変数の値を表示します。

C/C++
```
void omp_display_env (int verbose);
```

For
```
subroutine omp_display_env (verbose)
logical, intent(in) :: verbose
```

## 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_AFFINITY_FORMAT *format* [21.2.5] [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文字または名前です [表 21.2] [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_ALLOCATOR [21.5.1] [6.22]

OpenMP メモリー・アロケーターを使用して、割り当て要求を行うことができます。この環境変数は、アロケーターを指定しない割り当て呼び出し、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。値は、事前定義されたアロケーターまたはメモリー空間であり、オプションで 1 つ以上のアロケーター特性を指定できます。

- 事前定義されたメモリー空間を表 6.1 2.8 に示します。
- アロケーターの特性を表 6.2 2.9 に示します。
- 事前定義されたアロケーターを表 6.3 2.10 に示します。

**例:**

```
setenv OMP_ALLOCATOR omp_high_bw_mem_alloc

setenv OMP_ALLOCATOR ¥
  omp_large_cap_mem_space : alignment=16, ¥
  pinned=true

setenv OMP_ALLOCATOR ¥
  omp_high_bw_mem_space : pool_size=1048576, ¥
  fallback=allocator_fb, ¥
  fb_data=omp_low_lat_mem_alloc
```

**表 6.1 2.8 メモリー空間名**

| |
|---|
| **omp_default_mem_space, omp_large_cap_mem_space, omp_const_mem_space , omp_high_bw_mem_space, omp_low_lat_mem_space** |

**表 6.2 2.9 アロケーター特性と許容値**

| | |
|---|---|
| **sync_hint** | **contended**、**uncontended**、**serialized**、**private** |
| **alignment** | **1 バイト**; 2 の累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** |

**表 6.3 2.10 事前定義アロケーター (メモリー空間と特性値)**

| | |
|---|---|
| **omp_default_mem_alloc** | **omp_default_mem_space** **fallback:null_fb** |
| **omp_large_cap_mem_alloc** | **omp_large_cap_mem_space** (なし) |
| **omp_const_mem_alloc** | **omp_const_mem_space** (なし) |
| **omp_high_bw_mem_alloc** | **omp_high_bw_mem_space** (なし) |
| **omp_low_lat_mem_alloc** | **omp_low_lat_mem_space** (なし) |
| **omp_cgroup_mem_alloc** | 実装定義 **access:cgroup** |
| **omp_pteam_mem_alloc** | 実装定義 **access:pteam** |
| **omp_thread_mem_alloc** | 実装定義 **access:thread** |

## 環境変数

**OMP_CANCELLATION** [21.2.6] [6.11]
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** [21.4.1] [6.21]
内部制御変数 *debug-var* を設定します。*var* は **enabled** または **disabled** 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* [21.2.7] [6.15]
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* [21.2.4] [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。*var* は、**true** または **false** の場合があります。

**OMP_DISPLAY_ENV** *var* [21.7] [6.12]
*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* [21.1.1] [6.3]
内部制御変数 *dyn-var* を設定します。*var* は **true** または **false** です。**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* [21.1.4] [6.8]
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* [21.2.9] [6.16]
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_NUM_TEAMS** [21.6.1] [6.23]
内部制御変数 nteams-var を設定することで、**teams** 構造で生成されるチームの最大数を設定します。

**OMP_NUM_THREADS** リスト [21.1.2] [6.2]
**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* [21.1.6] [6.5]
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**、**ll_caches**、**numa_domains**)、または中括弧で区切られた数値の各場所が、デバイス上の順序付けされていないプロセッサーのセットである順序付きリストです。

**OMP_PROC_BIND** *policy* [21.1.7] [6.4]
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 **primary**、**close**、もしくは **spread** リストを指定できます [5.1 では、**master** は **primary** に置き換えられました]。

**OMP_SCHEDULE** *[modifier:] kind [,chunk]* [21.2.1] [6.1]
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。

**OMP_STACKSIZE** *size[***B** | **K** | **M** | **G***]* [21.2.2] [6.6]
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* [21.2.8] [6.17]
*target-offload-var* の初期値を設定します。引数は、**mandatory**、**disabled**、または **default** のいずれかす。

**OMP_TEAMS_THREAD_LIMIT** [21.6.2] [6.24]
内部制御変数 *teams-thread-limit-var* を設定することで、teams 構造で作成される各競合グループで使用する OpenMP スレッドの最大数を設定します。

**OMP_THREAD_LIMIT** *limit* [21.1.3] [6.10]
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**) [21.3.1] [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* リスト [21.3.2] [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* リスト 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_TOOL_VERBOSE_INIT** [21.3.3] [6.20]
内部制御変数 *tool-verbose-init-var* を設定します。これは、OpenMP 実装がツールの登録に関する詳細なログを記録するかどうか制御できます。値は、ファイル名、**disabled**、**stdout**、または **stderr** のいずれかです。

**OMP_WAIT_POLICY** *policy* [21.2.3] [6.7]
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* を設定します。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。

## 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。特定の環境変数は、*<ENV_VAR>*_**DEV**[_ *<device_num>*] でデバイス固有の環境変数を拡張できます。デバイス固有の環境変数は、グローバルスコープで ICV を初期化する環境変数に対応させてはなりません。

### ICV 初期値の表 および ICV 値の変更と取得方法 [表 2.1-3] [2.1-3]

| ICV | 環境変数 | 初期値 | 値の変更 | 値の取得 | スコープ | 参照 |
|---|---|---|---|---|---|---|
| *active-levels-var* | (なし) | *zero* | (なし) | **omp_get_active_level()** | データ環境 | --- |
| *affinity-format-var* | **OMP_AFFINITY_FORMAT** | 実装定義 | **omp_set_affinity_format()** | **omp_get_affinity_format()** | デバイス | [21.2.5] [6.14] |
| *bind-var* | **OMP_PROC_BIND** | 実装定義 | (なし) | **omp_get_proc_bind()** | データ環境 | [21.1.7] [6.4] |
| *cancel-var* | **OMP_CANCELLATION** | false | (none) | **omp_get_cancellation()** | グローバル | [21.2.6] [6.11] |
| *debug-var* | **OMP_DEBUG** | 無効 | (なし) | (なし) | グローバル | [21.4.1] [6.21] |
| *def-allocator-var* | **OMP_ALLOCATOR** | 実装定義 | **omp_set_default_allocator()** | **omp_get_default_allocator()** | 暗黙のタスク | [21.5.1] [6.22] |
| *def-sched-var* | (なし) | 実装定義 | (なし) | (なし) | デバイス | --- |
| *default-device-var* | **OMP_DEFAULT_DEVICE** | 実装定義 | **omp_set_default_device()** | **omp_get_default_device()** | データ環境 | [21.2.7] [6.15] |

## 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 | スコープ | 参照 |
|---|---|---|---|---|---|---|
| *display-affinity-var* | **OMP_DISPLAY_AFFINITY** | false | (なし) | (なし) | グローバル | [21.2.4] [6.13] |
| *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()** | データ環境 | [21.1.1] [6.3] |
| *explicit-task-var* | (なし) | false | | **omp_in_explicit_task()** | | |
| *final-task-var* | (なし) | False | (なし) | **omp_in_final()** | データ環境 | --- |
| *levels-var* | (なし) | ゼロ | (なし) | **omp_get_level()** | データ環境 | --- |
| *max-active-levels-var* | **OMP_MAX_ACTIVE_LEVELS**、**OMP_NUM_THREADS**、**OMP_PROC_BIND** | 実装定義 | **omp_set_max_active_levels()** | **omp_get_max_active_levels()** | デバイス<br>データ環境 | [21.1.4] [6.8]<br>[21.1.2] [6.9]<br>[21.1.7] [6.2] |
| *max-task-priority-var* | **OMP_MAX_TASK_PRIORITY** | ゼロ | (なし) | **omp_get_max_task_priority()** | グローバル | [21.2.9] [6.16] |
| *nteams-var* | **OMP_NUM_TEAMS** | ゼロ | **omp_set_num_teams()** | **omp_get_max_teams()** | デバイス | [21.6.1][6.23] |
| *nthreads-var* | **OMP_NUM_THREADS** | 実装定義 | **omp_set_num_threads()** | **omp_get_max_threads()** | データ環境 | [21.1.2][6.2] |
| *num-procs-var* | (なし) | 実装定義 | (なし) | **omp_get_num_procs()** | デバイス | --- |
| *place-partition-var* | **OMP_PLACES** | 実装定義 | (なし) | **omp_get_partition_num_places()**<br>**omp_get_partition_place_nums()**<br>**omp_get_place_num_procs()**<br>**omp_get_place_proc_ids()** | 実装タスク | [21.1.6][6.5] |
| *run-sched-var* | **OMP_SCHEDULE** | 実装定義 | **omp_set_schedule()** | **omp_get_schedule()** | データ環境 | [21.2.1][6.1] |
| *stacksize-var* | **OMP_STACKSIZE** | 実装定義 | (なし) | (なし) | デバイス | [21.2.2] [6.6] |
| *target-offload-var* | **OMP_TARGET_OFFLOAD** | デフォルト | (なし) | (なし) | グローバル | [21.2.8] [6.17] |
| *team-size-var* | (なし) | 1 | (なし) | **omp_get_num_threads()** | チーム | --- |
| *teams-thread-limit-var* | **OMP_TEAMS_THREAD_LIMIT** | ゼロ | **omp_set_teams_thread_limit()** | **omp_get_teams_thread_limit()** | デバイス | [21.6.2] [6.24] |
| *thread-limit-var* | **OMP_THREAD_LIMIT** | 実装定義 | **target** および **teams** 構造 | **omp_get_thread_limit()** | データ環境 | [21.1.3] [6.10] |
| *thread-num-var* | (なし) | ゼロ | (なし) | **omp_get_thread_num()** | 実装タスク | --- |
| *tool-libraries-var* | **OMP_TOOL_LIBRARIES** | 空の文字列 | (なし) | (なし) | グローバル | [21.3.2] [6.19] |
| *tool-var* | **OMP_TOOL** | 有効 | (なし) | (なし) | グローバル | [21.3.1] [6.18] |
| *tool-verbose-init-var* | **OMP_TOOL_VERBOSE_INIT** | 無効 | (なし) | (なし) | グローバル | [21.3.3] [6.20] |
| *wait-policy-var* | **OMP_WAIT_POLICY** | 実装定義 | (なし) | (なし) | デバイス | [21.2.3] [6.7] |

## OpenMP ツールを使用する

ツールは、**ompt_start_tool** からの戻り値として、OpenMP 実装に **ompt_start_tool_result_t** 構造体への **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** 定義を提供するには、次の 3 つの方法があります。

- 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。
- 内部制御変数 *tool-libraries-var* を使用して (**omp_tool_libraries** を介して)、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。

**omp_tool_verbose_init** を使用すると、ツールのロードまたはアクティブ化に関連する問題を理解するのに役立ちます。このランタイム・ライブラリー・ルーチンは、OpenMP 実装がツールの登録を冗長に記録するか制御する内部制御変数 *tool-verbose-init-var* を設定します。

## OpenMP API 5.2 リファレンス・ガイド索引

**節 - 9**

**ディレクティブと構造 - 1-8**
allocate、allocators - 1
assume、assumes - 2
atomic - 6
barrier - 5
begin assumes - 2
begin declare target - 2
cancel - 6
Cancellation 構造
  cancel - 6
  cancellation point - 6
cancellation point - 6
結合形式 (Combined forms) - 6–8
critical - 5
データ環境ディレクティブ - 1
  declare mapper - 1
  declare reduction - 1
  scan - 1
  threadprivate - 1
declare mapper - 1
declare reduction - 1
declare simd - 2
declare target - 2
declare variant - 2
depobj - 5
デバイス・ディレクティブと構造 - 4
  target - 5
  target data - 4
  target enter data - 5
  target exit data - 5
  target update - 5
dispatch - 2
distribute - 4
do - 4
flush - 5
for - 4
情報とユーティリティー・ディレクティブ - 2
  assume, [begin ]assumes - 2
  error - 3
  nothing - 3
  requires - 2
interop - 5
loop - 4
ループ変換構造 - 3
  tile - 3
  unroll - 3
masked - 3
メモリー管理ディレクティブ - 1
  allocate - 1
  allocators - 1
  メモリー空間 - 1
metadirective - 1
nothing - 3
ordered - 6
parallel - 3
並列構造 - 3
  masked - 3
  parallel - 3
  simd - 3
  teams - 3
requires - 2
scan - 1
scope - 3
section、sections - 4
simd - 3
single - 3
同期構造 - 5
  atomic - 6
  barrier - 5
  critical - 5
  depobj - 5
  flush - 5
  ordered - 6
  taskgroup - 5
  taskwait - 5
target - 5
target data - 4
target enter data - 5
target exit data - 5
target update - 5
task - 4
taskgroup - 5
タスク構造 - 4
  task - 4
  taskloop - 4
  taskyield - 4
taskloop - 4
taskwait - 5
taskyield - 4
teams - 3
threadprivate - 1
tile - 3
unroll l -3
バリアント・ディレクティブ - 1
  [begin ]declare target - 2
  declare simd - 2
  declare variant - 2
  dispatch - 2
  metadirective - 1
ワーク分散構造 - 3
  distribute - 4
  do - 4
  for - 4
  loop - 4
  scope - 3
  section、sections - 4
  single - 3
  workshare - 3
workshare - 3

**環境変数 - 14–15**

**内部制御変数 (ICV) - 15–16**

**ランタイム・ライブラリー・ルーチン - 10–14**
デバイス情報ルーチン - 11
デバイス・メモリー・ルーチン - 11–12
環境表示ルーチン - 14
イベントルーチン - 13
相互運用性ルーチン - 13
ロックルーチン - 12–13
メモリー管理ルーチン - 13
リソース解放ルーチン - 11
タスクルーチン - 11
チーム領域ルーチン - 11
スレッド・アフィニティー・ルーチン - 10–11
スレッド・チーム・ルーチン - 10
タイミングルーチン - 13
ツール制御ルーチン - 14