OpenMP 6.0 API
構文リファレンス・ガイド

このドキュメントは、openmp.org で公開されている OpenMP 6.0 API Syntax Reference Guide と OpenMP 6.0 API Specification の参考訳です。openmp.org で公開されているシンタックス・リファレンス・ガイドは、カード形式ですが日本語版は見やすくするため文書形式にしています。

目次

はじめに

OpenMP API は、C/C++ および Fortran を使用するプログラマーに移植性の高い並列アプリケーションを開発するシンプルで柔軟なインターフェイスを提供するスケーラブルな並列プログラミング・モデルです。OpenMP は、マルチコアノードやチップ、NUMA システム、GPU、CPU に接続された各種デバイスで実行されるさまざまなアルゴリズムに適しています。

このガイドの読み方

このガイドには、C/C++ と Fortran 言語向けの OpenMP API の説明が含まれています。それぞれの言語に特化した説明は次のマークで示されます。

C/C++: C/C++ を対象とする説明
Fortran: Fortran を対象とする説明

[n.n.n]: 6.0 仕様の関連するセクション
[n.n.n]: 5.2 仕様の関連するセクション

OpenMP ディレクティブの形式

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（ディレクティブ名とそれに続くオプションの句）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/C++ ディレクティブは、プラグマのみ、または属性付きのプラグマで構成されます。
Fortran: Fortran ディレクティブは、自由形式および固定形式のソース（コード）内のコメントで構成されます。

ディレクティブ形式の詳細については [5] [5]を参照してください。

例:

C/C++ #pragma omp *ディレクティブ指定*
C++ [[omp :: directive(*ディレクティブ指定*)]]
C++ [[using omp : directive(*ディレクティブ指定*)]]
For !$omp *ディレクティブ指定*
For !$omp *ディレクティブ指定*
!$omp end *ディレクティブ名*

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。 が示されない限り、ディレクティブ名にはアンダースコアが必要です（アンダースコアで指定されている場合を除く）。

- 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 PURE プロシージャー内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 1 つの出口があります。
- 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。
- 緩い構造化されたブロックは、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++* のスカラー int 型、または *Fortran* のスカラー integer 型です。
- omp 論理式は、*C/C++* のスカラー式、または *Fortran* の論理式です。

データ環境ディレクティブ

threadprivate [7.3] [5.2]

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で区切ったリスト。

For 名前付き変数と名前付き共通ブロックのカンマで区切ったリスト。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 [7.6.4] [5.5.11]

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***) [***節* **[[,] ***節***]**
For	**!$omp declare reduction (***リダクション識別子* **:** *タイプ名リスト***) [***節* **[[,] ***節***]**

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演算子:

+、*、.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 タイプ名リスト: CLASS(*) または抽象タイプのタイプ指定子のリスト

節:

コンバイナー(*コンバイナー式*)

コンバイナー式:
C/C++ 関数呼び出しを含む式
For 代入文またはサブルーチン名と引数リスト。
omp_out を結果として使用し、omp_in と omp_out をリダクション演算の 2 つの入力オペランドとして使用します。

初期化子(*初期化子式*)

初期化子式: omp_priv = 初期化子、または関数名(引数リスト)

declare_induction [7.6.17]

induction 節で使用できる*インダクション識別子*を宣言します。

C/C++	#pragma omp declare_induction (*インダクション識別子* : *タイプ指定子リスト*) *節* [,] *節*
For	!$omp declare_induction (*インダクション識別子* : *タイプリスト*) *節* [,] *節*

節:
collector 節と inductor 節の両方を指定する必要があります。

collector(*コレクター式*)

コレクター式: ループカウンターには omp_idx を、ループ・インクリメントには omp_step を使用します。

inductor(*インダクター式*)

インダクター式: omp_var = *式*。インダクター変数には omp_var を、ループ・インクリメンには omp_step を使用します。

scan [7.7] [5.6]

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 *init_complete 節*]

	構造化ブロックシーケンス **#pragma omp scan** *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 *[* **!$omp scan** *init_complete 節* *]* 構造化ブロックシーケンス **!$omp scan** *scan 節* 構造化ブロックシーケンス

節:

scan 節

exclusive (*リスト*)

inclusive (*リスト*)

init_complete 節

init_complete (*omp 定数論理式*)

declare_mapper [7.9.10] [5.8.8]

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*:] *タイプ var*) [*節*[[,] *節*] …]
For	**!$omp declate_mapper** ([*マッパー識別子*:] *タイプ var*) [節[[,] 節] …]

節:

map ([[*マップ修飾子*, [*マップ修飾子*,…]] *マップタイプ*:] *リスト*) C

マップタイプ: ストレージ、**from**、**to**、**tofrom**

マップ修飾子: **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 C

タイプ: 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

groupprivate [7.13]

リスト項目を、競合するグループがそれぞれコピーを受け取るように配布してください。

C/C++	**#pragma omp groupprivate** *[節]*
For	**!$omp declate groupprivate** *[節]*

節:

device_type (**host** | **nohost** | **any**)

プロシージャーまたは変数のバージョンを利用可能にするデバイスのタイプを指定します。

メモリー管理ディレクティブ

メモリー空間 [8.1] [6.1]

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 [8.5] [6.6]

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 (*アロケーター*)

アロケーター:

C/C++ omp_allocator_handle_t タイプ

For omp_allocator_handle_kind カインド

allocators [8.7] [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	**!$omp allocators** *[節]* *アロケート文* *[!$omp end allocators]*

節:

allocate C

アロケート文: Fortran の ALLOCATE 文。

バリアント・ディレクティブ

[begin] metadirective [9.4.3-4] [7.4.3-4]

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(コンテキスト・セレクター指定 : [*ディレクティブ・バリアント*])

ディレクティブ・バリアント条件付きで選択。

otherwise ([*ディレクティブ・バリアント*])

条件付きでバリアント・ディレクティブを選択します。

otherwise は以前のバージョンで **default** と呼ばれていました。

[begin] declare_variant [9.6.4-5] [7.5.4-5]

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 directive_variant** (*バリアント関数 id*) *[節 [[,] 節] …]* [**#pragma omp directive_variant** (*バリアント関数 id*) *[節 [[,] 節] …]*] *関数定義または宣言* または **#pragma omp begin directive_variant** *マッチ節* *シーケンスの宣言定義* **#pragma omp end directive_variant**
For	**!$omp directive variant** ([*ベース・プロシージャー名*:] *バリアント・プロシージャー名*) *節 [[[,] 節] …]*

節:

adjust_args (*アジャスト操作* : *引数リスト*)

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。

アジャスト操作: **need_device_addr**、**need_device_ptr**、**nothing**。

append_args (*アペンド操作 [[, アペンド操作]…]*)

アペンド操作: interop(*interop タイプ [[, interop タイプ] …]*)

match(*コンテキスト・セレクター指定*)

置換対象としてバリアント関数が選択される条件を指定します。

match 節:

match(*コンテキスト・セレクター指定*)

必須。上記と同じ。

C/C++ バリアント関数 id

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。

For ベース・プロシージャー名

ベース言語の識別子である関数バリアント名。

dispatch [9.7] [7.6]

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 (*[依存関係修飾子,] 依存関係タイプ : ロケーターリスト*) C

device (*omp 整数式*) C

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

interop (*interop 変数リスト*)

is_device_ptr (*リスト*)

*リスト*はデバイスポインターです。

has_device_addr (*リスト項目*)

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 **dispatch** 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 C

declare_simd [9.8] [7.7]

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** *[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。

linear (*リニアリスト [: リニアステップ]*) C

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 [9.9.1-2] [7.8.1-2]

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** (*拡張リスト*) *または* **#pragma omp declare_target** *節 [[,] 節 …]* *または* **#pragma omp begin declare_target** *[節 [[,] 節] …]* *宣言関数定義シーケンス* **#pragma omp end declare_target**
For	**!$omp declare_target** (*拡張リスト*) *または* **!$omp declare_target** *[節 [[,] 節] …]*

節:

device_type (**host** | **nohost** | **any**)

プロシージャーまたは変数のバージョンを利用可能にするデバイスの種類を指定します。

enter (*拡張リスト*)

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。

local (*リスト項目*)

*リスト項目*への参照は、デバイス上の*リスト項目*のローカルコピーを参照するように指定します。

- **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** 節である必要があります。
- **begin declare target** の場合、**indirect** と **device type** 節のみが許可されます。

情報とユーティリティー・ディレクティブ

requires [10.5] [8.2]

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**)

実装では、**memory-order** 節を指定しないアトミック構造に対して、指定されたメモリー順序を使用することが求められます。

device_safesync

safesync 節が指定されていない限り、チーム内の異なるスレッドの同期は、ホスト以外のデバイス上で進行できる必要があります。

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）。

self_maps

マップ入力ディレクティブのすべての **map** 節には、暗黙の **self** 修飾子が必要であることを要求します。

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。

assume, [begin]assumes [10-6.2-4] [8.3.2-4]

最適化に使用できる、実装に対する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*
For	**!$omp assumes** *節 [[[,] 節] …]* *または* **!$omp assume** *節 [[[,] 節] …]* *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 (*ディレクティブ名 [[, ディレクティブ名] …]*)
スコープ内に存在しないディレクティブを指定します。

containes (*ディレクティブ名 [[, ディレクティブ名] …]*)
スコープ内に含まれるディレクティブを指定します。

no_openmp
no_openmp_constructs と **no_openmp_routines** の仮定を組み合わせたものと同等です。

no_openmp_constructs
スコープ内で OpenMP 構造が含まれていないことを示します。

no_openmp_routines
スコープ内で OpenMP ランタイム・ライブラリーの呼び出しがないことを示します。

no_parallelism
スコープ内で OpenMP task や SIMD 構造が実行されないことを示します。

nothing [10.7] [8.4]

意図的に効果を持たないことを明示的に示します。

C/C++	**#pragma omp nothing** *[節]*
For	**!$omp nothing** *[節]*

節:

apply (*適用されるディレクティブ*) C

nothing ディレクティブがループ変換構造を形成する場合にのみ指定できます。

error [10.1] [8.5]

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**)

ループ変換構造

fuse [11.3]

複数のループを 1 つのループに統合し、そのループの各反復処理で、各入力ループの 1 回の反復処理を実行します。

C/C++	**#pragma omp fuse** *節* *ループの入れ子シーケンス*
For	**!$omp fuse** *節* *ループの入れ子シーケンス* *[* **!$omp end fuse** *]*

節:

apply (*適用されるディレクティブ*) C
looprange (*入れ子ループのリスト*)

interchange [11.4]

ループの入れ子構造内のループの入れ子順序を変更します。

C/C++	**#pragma omp interchange** *[節 [[,] 節] …]* *ループの入れ子*
For	**!$omp interchange** *[節 [[,] 節] …]* *ループの入れ子* *[* **!$omp end interchange** *]*

節:

apply (*適用されるディレクティブ*) C
permutation (*順列リスト*)

reverse [11.5]

ループの反復処理を逆順で実行します。

C/C++	**#pragma omp reverse** *[節 [[,] 節] …]* *ループの入れ子*
For	**!$omp reverse** *[節 [[,] 節] …]* *ループの入れ子* *[* **!$omp end reverse** *]*

節:

apply (*適用されるディレクティブ*) C

split [11.6]

ループを複数のループに分割し、それぞれが連続する反復処理の一部を実行するようにします。

C/C++	**#pragma omp split** *[節 [[,] 節] …]* *ループの入れ子*
For	**!$omp split** *[節 [[,] 節] …]* *ループの入れ子* *[* **!$omp end split** *]*

節:

apply (*適用されるディレクティブ*) C

counts (*カウントリスト*)

stripe [11.7]

sizes 節によって決定されるパーティションごとに、ループの反復処理を交互に実行します。

C/C++	**#pragma omp stripe** *[節 [[,] 節] …]* *ループの入れ子*
For	**!$omp stripe** *[節 [[,] 節] …]* *ループの入れ子* *[* **!$omp end stripe** *]*

節:

apply (*適用されるディレクティブ*) C

sizes (*サイズリスト*)

tile [11.8] [9.1]

1 つ以上のループをタイル化します。

C/C++	**#pragma omp tile** *[節 [[,] 節] …]* *ループの入れ子*
For	**!$omp tile** *[節 [[,] 節] …]*

	ループの入れ子 *[* **!$omp end tile** *]*

節:

apply (*適用されるディレクティブ*) C
sizes (*サイズリスト*)

unroll [11.9] [9.2]

ループ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 [[,] 節] …]* *ループの入れ子*
For	**!$omp unroll** *[節]* *ループの入れ子* *[* **!$omp end unroll** *]*

節:

apply (*適用されるディレクティブ*) C
full
partial *[*(*アンロール係数*)*]*

並列構造

parallel [12.1] [10.1]

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] …]* *緩い構造化ブロック*
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 C
copyin(*リスト*)
default (*データ共有属性*) C
firstprivate (*リスト*) C
if (*[***parallel:** *]* *omp 論理式*) C
num_threads (*スレッド数*)
実行するスレッド数を指定します。

private (*リスト*) C
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* プレース間で *T* スレッドのチームのスパース分布を行います。
reduction C
safesync (*幅*)
severity (**fatal** | **warning**)
shared (*リスト*) C

teams [12.2] [10.2]

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	**!$omp teams** *[節 [[,] 節] …]* *緩い構造化ブロック* **!$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 C
default (*データ共有属性*) C
firstprivate (*リスト*) C
if (*[***teams :** *] omp 論理式*) C
num_teams (*[下限:] 上限*)
private (*リスト*) C
reduction C
shared (*リスト*) C
thread_limit (*omp 整数式*)

simd [12.4] [10.4]

ループに適用され、そのループが SIMD ループに変換可能であることを示します。

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*) C

if (*[*simd:*] omp 論理式*) C

induction C

lastprivate (*[lastprivate 修飾子:] リスト*) C

linear (*リスト [: リニアステップ]*) C

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**) C

order 修飾子: **reproduce** または **unconstrained**

private (*リスト*) C

reduction C

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 [12.5] [10.5]

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]* *緩い構造化ブロック*
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 (*スレッド数*)

構造化ブロックを実行するスレッド数を指定します。

ワークシェア構造

single [13.1] [11.1]

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 end single** *[end 節 [[,] end 節] ···]]*

節:

allocate C
copyprivate (*リスト*)
firstprivate (*リスト*) C
nowait C
private (*リスト*) C

end 節:

copyprivate (*リスト*)
nowait C

scope [13.2] [11.2]

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 C
firstprivate (*リスト*) C

nowait C
private (*リスト*) C
reduction C

section と sections [13.3-13.3.1] [11.3-11.3.1]

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 end sections *[* nowait *]*

節:

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

worksgare [13.4] [11.4]

囲まれた構造化ブロックの実行を個別のワーク単位に分割し、各ワーク単位はチーム内の 1 つのスレッドによって一度だけ実行されます。

For	!$omp worksharte *[節]* *緩い構造化ブロック* !$omp end workshare *[節]* *または* !$omp worksharte *[節]* *厳密な構造化ブロック]* *[*!$omp end workshare *[節]]*

節:

nowait C

workdistribute [13.5]

囲まれた構造化ブロックの実行を個別のワーク単位に分割し、各ワーク単位はリーグ内の各初期スレッドによって一度だけ実行されます。

For	!$omp workdistribute *緩い構造化ブロック* !$omp end distribute *または* !$omp workdistribute *厳密な構造化ブロック]* *[* !$omp end distribute *]*

do と for [13.6.1-2] [11.5.1-2]

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 C
collapse (*n*) C
firstprivate (*リスト*) C
induction (*リスト*)
lastprivate (*[lastprivate 修飾子:] リスト*) C
linear (*リスト [: リニアステップ]*) C
nowait C
order (*[order 修飾子:]* concurrent) C
order 修飾子: reproducible または 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*) C
reduction C
schedule (*[修飾子 [,修飾子]:] kind[,chunk_size]*)

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新しい *chunk_size* は、[*chunk_size*/ *simd_width*] * *simd_width* となります(*simd_width* は実装依存です)

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**: **dynamic** と同等ですが、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* ICV から取得されます。

distribute [13.7] [11.6]

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 C
collapse (*n*) C
dist_schedule (*kind [, chunk_size]*)
firstprivate (*リスト*) C
induction (*リスト*) C
lastprivate (*リスト*) C
order (*[order 修飾子 :]* **concurrent**) C

order 修飾子: reproducible または unconstrained

private (*リスト*) C

loop [13.8] [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 (*バインド*)

バインド: teams、parallel、thread

collapse (*n*) C

lastprivate (*リスト*) C

order (*[order 修飾子 :]* concurrent) C

order 修飾子: reproducible または unconstrained

private (*リスト*) C

reduction C

タスク構造

task [14.1] [12.5]

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:*] *ロケーターリスト*)

aff 修飾子: iterator (*イテレーター定義*) C

allocate C

default (*データ共有属性*) C
depend ([*依存関係修飾子*,] *依存関係タイプ* : *ロケーターリスト*) C
detach (*イベントハンドル*)
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)。
イベントハンドル:
C/C++ **omp_event_handle_t** タイプ
For kind **omp_event_handle_kind**
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*) C
if ([**task**:] *omp 論理式*) C
in_reduction (*リダクション識別子*: *リスト*) C
mergeable
priority (*優先度の値*) C
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*) C
replayable C
shared (*リスト*) C
threadset (**omo_pool** | **omp_team**) C
transparent (*impex タイプ*)
impex タイプ は、**omp_import**、**omp_export**、**omp_impex**、**omp_not_impex** のいずれかです。
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 [14.2] [12.6]

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 C
collapse (*n*) C
default (*データ共有属性*) C
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*) C
grainsize (*[*strict :*]* *グレインサイズ*)
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*) C
in_reduction (*リダクション識別子*: *リスト*) C
induction C
lastprivate (*[lastprivate 修飾子 :]* *リスト*) C
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*) C
private (*リスト*) C
reduction C
replayable C
shared (*リスト*) C
threadset
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。

task_iteration [14.2.3]

関連する taskloop 構造によって生成されるタスクの、反復ごとのタスク実行属性を制御します。taskloop 構造によって生成されるタスクに、タスクの依存関係とタスクのアフィニティーを適用できるようにします。

C/C++	**#pragma omp task_iteration** *節 [[[,] 節] …]*
For	**!$omp task_iteration** *節 [[[,] 節] …]*

節:
affinity C
depend C
if (*[*task_iteration : *]* *omp 論理式*)

taskyield [14.12] [12.7]

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**

taskgraph [14.3]

構造化ブロックを実行し、生成されたタスクの順序と依存関係を記録して、後で再生できるようにします。

C/C++	**#pragma omp taskgraph** *[節 [[,] 節] …]*
For	**!$omp taskgraph** *[節 [[,] 節] …]*

節:

graph_id (*グラフ id*)

記録と再生のためグラフを区別する数値の識別子。

graph_reset (*omp 論理条件*)

条件が true と評価されたときに、グラフ id を使用して記録されたグラフをリセットします。

if (*[taskgraph :] omp 論理式*) C

nogroup

暗黙的な taskgroup 領域の作成を禁止します。

デバイス・ディレクティブと構造

target data [15.7] [13.5]

指定された領域範囲の変数をデバイスデータ環境にマッピングします。

C/C++	**#pragma omp target_data** *節 [[[,] 節] …]* *構造化ブロック*
For	**!$omp omp target_data** *節 [[[,] 節] …]* *緩い構造化ブロック* **!$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:

affinity C

allocate

default (*データ共有属性*) C

depend C

detach

device (*omp 整数式*) C

firstprivate (*リスト*) C

if (*[target_data :] omp 論理式*) C

in_reduction

map (*[[マップ修飾子, [マップ修飾子, ...]] マップタイプ :] リスト*) C

mergeable
nogroup
暗黙的な **taskgroup** 領域の生成を禁止します。
nowait C
private (*リスト*) C
priority (*優先順位値*) C
shared (*リスト*) C
transparent (*impex タイプ*)
*impex タイプ*は、**omp_import**、**omp_export**、**omp_impex**、**omp_not_impex** のいずれかです。
use_device_ptr (*リスト*)
use_device_addr (*リスト*)

target_enter_data [15.5] [13.6]

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* : *ロケーターリスト*) C
device (*omp 整数式*) C
if ([target data :] *omp 論理式*) C
map ([[*マップ修飾子*, [*マップ修飾子*, ...]] *マップタイプ*:] *リスト*) C
nowait C
priority C
replayable C

target_exit_data [15.6] [13.7]

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 で利用できる節。

target [15.8] [13.8]

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] …]*

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

節:

allocate C
default C
defaultmap (*暗黙の動作 [: 変数カテゴリー]*)
暗黙の動作: storage、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 : ロケーターリスト*) C
device (*[デバイス修飾子:] omp 整数式*) C
デバイス修飾子: ancestor、device_num
firstprivate (*リスト*) C
has_device_addr (*リスト*)
*リスト*項目には既にデバイスアドレスが割り当てられているため、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。
if (*[* target : *] omp 論理式*) C
in_reduction (*リダクション識別子 : リスト*) C
is_device_ptr (*リスト*)
*リスト*項目はデバイスポインターであることを示します。
map (*[[マップ修飾子, [マップ修飾子, ...]] マップタイプ :] リスト*) C
nowait C
private (*リスト*) C
priority C
replayable C
thread_limit (*omp 整数式*)
uses_allocators (*[[alloc-mod ,] alloc-mod]: アロケーター*)
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 [15.9] [13.9]

モーション節で指定される、元のリスト項目とデバイスデータ環境のリスト項目を一致させます。

C/C++	#pragma omp target_update *節 [[[,] 節] …]*
For	!$omp target_update *節 [[[,] 節] …]*

節:

nowait C

depend (*[依存関係修飾子,] 依存関係タイプ : ロケーターリスト*) C

device (*omp 整数式*) C

from (*[モーション修飾子[,][モーション修飾子[,]…] :] ロケーターリスト*)

モーション修飾子: present、mapper (*マッパー識別子*)、iterator (*イテレーター定義*)

if (*[* target update *:] omp 論理式*) C

priority C

replayable C

to (*[モーション修飾子[,][モーション修飾子[,]…] :] ロケーターリスト*)

モーション修飾子: present、mapper (*マッパー識別子*)、iterator (*イテレーター定義*)

相互運用構造

interop [16.1] [14.1]

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*) C

depend (*[依存関係修飾子,] 依存関係タイプ : ロケーターリスト*) C

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 C

use (*interop-var*)

同期構造

critical [17.2] [15.2]

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** *[(名前) [[,] hint(ヒント式)]]* *構造化ブロック*
For	**!$omp critical** *[(名前) [[,] hint(ヒント式)]]* *緩い構造化ブロック* **!$omp end critical** *[(名前)]* *または* **!$omp critical** *[(名前) [[,] hint(ヒント式)]]* *厳密な構造化ブロック* *[* **!$omp end critical** *[(名前)]]*

ヒント式:

omp_sync_hint_contended
omp_sync_hint_none
omp_sync_hint_nonspeculative
omp_sync_hint_speculative
omp_sync_hint_uncontended

barrier [17.3] [15.3.1]

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

	[!$omp end taskgroup *]*

節:

allocate C

task_reduction (*リダクション識別子* : *リスト*)

リダクション識別子: reduction を参照 C

taskwait [17.5] [15.5]

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* : *ロケーターリスト*) C

nowait C

replayable C

atomic [17.8.5] [15.8.4]

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] …]* *文*
For	!$omp atomic *[節 [[,] 節] …]* *文* *[* !$omp end atomic *]* *または* !$omp atomic *[節 [[,] 節] …]* capture *[[,] 節 [[[,] 節] …]]* *文* *キャプチャー文* *[* !$omp end atomic *]* *または* !$omp atomic *[節 [[,] 節] …]* capture *[[,] 節 [[[,] 節] …]]* *キャプチャー文* *文* *[* !$omp end atomic *]*

節:

アトミック節 : read、write、update

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

拡張アトミック : 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)

memscope (**device** | **cgroup** | **all**)

アトミック操作の影響を受けるスレッドセットを決定します（**device**: 現在のデバイス上のすべてのスレッド、**cgroup**: 現在の競合グループ内のすべてのスレッド、**all**: システム内のすべてのスレッド）。

C/C++ 文:

アトミック節	式文
read	*v = x;*
write	*x = expr;*
update	*x++; x--; ++x; --x;*
	x binop= expr; x = x binop expr;
	x = expr binop x;
compare がある場合	*cond-update-stmt:*
	if *(expr ordopx) { x = expr; }*
	if *(x ordopexpr) { x = expr; }*
	if *(x == e) { x = d; }*
capture がある場合	*v = expr-stmt*
	{ v = x; expr-stmt}
	{ expr-stmtv = x; }
	(ここで、*expr-stmt*is は *write-expr-stmt*、*update-expr-stmt*、または *cond-expr-stmt* のいずれか)
compare と **capture** がある場合	*{ v = x; cond-update-stmt}*
	{ cond-update-stmtv = x; }
	if *(x == e) { x = d; }* **else** *{ v = x; }*
	{ r = x == e; **if** *(r) { x = d; } }*
	{ r = x == e; **if** *(r) { x = d; }* else *{ v = x; } }*

For キャプチャー文: *v = x* 形式

アトミック節	式文
read	*v= x*
write	*x = expr*
update	*x = x operator expr*
	x = expr operator x
	x = intrinsic_procedure_name(x, expr-list)
	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。
capture が存在し、文の前後に **capture** 文がある場合　*x* = *expr*, その他許可されているものに加えて
compare がある場合　**if** (*x* == *e*) **then**
x = *d*
end if
if (*x* == *e*) *x* = *d*
compare と **capture** があり、文の前後に **capture** 文がない場合　**if** (*x* == *e*) **then**
x = *d*
else *v* = *x*
end if

flush [17.8.6] [15.8.5]

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** *[メモリー順序節] [(リスト)]*
For	**!$omp flush** *[メモリー順序節] [(リスト)]*

メモリー順序節:
seq_cst、**acq_rel**、**release**、**acquire**、**relaxed**
memscope (**device** | **cgroup** | **all**)
アトミック操作の影響を受けるスレッドセットを決定します（**device**：現在のデバイス上のすべてのスレッド、**cgroup**：現在の競合グループ内のすべてのスレッド、**all**：システム内のすべてのスレッド）。

depobj [17.9.3] [15.9.4]

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 (*依存オブジェクト*)
init (*依存関係タイプ* (*リスト項目*) : *依存オブジェクト*)
*依存オブジェクト*の依存するオブジェクトを初期化します。
依存関係タイプ : **in**、**out**、**inout**、**inoutset**、**mutexinoutset**
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**

ordered [17.10] [15.10.2]

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* : [*ベクトル*]）

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、反復間の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、*ベクトル*引数はオプションです。*ベクトル*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、*ベクトル*は依存関係を満たす反復をしします。*ベクトル*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 [18.2] [16.1]

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 [18.3] [16.2]

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 *構造タイプ節*

構造タイプ節:

parallel
sections
taskgroup
C/C++ for
For do

結合構造

次の結合構造とディレクティブは、OpenMP API バージョン 6.0 仕様の 19 章および付録 D で定義される規則に従って作成されています。ここに示されている結合ディレクティブは、一般的に有用なディレクティブの組み合わせ例であり、完全なリストではありません。

distribute parallel do と distribute parallel for [19] [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	**!$omp distribute parallel for** *[節 [[,] 節] …]* *入れ子のループ* *[* **!$omp end distribute parallel for** *]*

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

distribute parallel do simd と distribute parallel for simd [19] [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	**!$omp distribute parallel for simd** *[節 [[,] 節] …]* *入れ子のループ* *[* **!$omp end distribute parallel for simd** *]*

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

distribute simd [19] [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** または **simd** と同一の節が適用されます。

do simd と for simd [19] [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** または **do** と同一の節が適用されます。

masked taskloop [19] [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** 構造を指定するショートカット。

C/C++	**#pragma omp masked taskloop** *[節 [[,] 節] …]* *入れ子のループ*
For	**!$omp masked taskloop** *[節 [[,] 節] …]* *入れ子のループ* *[* **!$omp end masked taskloop** *]*

節: **taskloop** または **masked** と同一の節が適用されます。

masked taskloop simd [19] [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 masked taskloop simd** *]*

節: **taskloop simd** または **masked** と同一の節が適用されます。

parallel do と parallel for [19] [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** と同一の節が適用されます。

parallel do simd と parallel for simd [19] [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** 構造を指定するショート

カット。

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: (*C/C++* では **nowait** 節を除く) **parallel**、**for simd** または **do simd** と同一の節が適用されます。

parallel loop [19] [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** 構造を指定するショートカット。

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** と同一の節が適用されます。

parallel masked [19] [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** 構造を指定するショートカット。

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** と同一の節が適用されます。

parallel masked taskloop [19] [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** 構造を指定するショートカット。

C/C++	**#pragma omp parallel masked taskloop** *[節 [[,] 節] …]* *構造化ブロック*

For	**!$omp parallel masked taskloop** *[節 [[,] 節] …]* *構造化ブロック* *[***!$omp end parallel masked taskloop** *]*

節: **parallel** または **masked taskloop** と同一の節が適用されます。

parallel masked taskloop simd [19] [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** 構造を指定するショートカット。

C/C++	**#pragma omp parallel masked taskloop simd** *[節 [[,] 節] …]* *構造化ブロック*
For	**!$omp parallel masked taskloop simd** *[節 [[,] 節] …]* *構造化ブロック* *[***!$omp end parallel masked taskloop simd** *]*

節: **parallel** または **masked taskloop simd** と同一の節が適用されます。

parallel target [19]

target 構造のみを含む **parallel** 構造を指定します。

C/C++	**#pragma omp parallel target** *[節 [[,] 節] …]* *構造化ブロック*
For	**!$omp parallel target** *[節 [[,] 節] …]* *緩い構造化ブロック* **!$omp end parallel target** *または* **!$omp parallel target** *[節 [[,] 節] …]* *厳密な構造化ブロック* *[***!$omp end parallel target** *]*

target loop [19] [2.16.15]

正規ループネストを持ち、その他のステートメントを含まない **loop** 構造を含む **target** 領域を指定するショートカット。

C/C++	**#pragma omp target loop** *[節 [[,] 節] …]* *入れ子のループ*
For	**!$omp target loop** *[節 [[,] 節] …]* *入れ子のループ* *[***!$omp end target loop** *]*

節: **teams** または **loop** と同一の節が適用されます。

target teams [19] [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** 構造を指定するショートカット。

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** と同一の節が適用されます。

target teams distribute [19] [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** 構造を指定するショートカット。

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** と同一の節が適用されます。

target teams distribute simd [19] [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** 構造を指定するショートカット。

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** と同一の節が適用されます。

target teams loop [19] [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** 構造を指定するショートカット。

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** と同一の節が適用されます。

target teams distribute parallel do と target teams distribute parallel for [19] [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** 構造を指定するショートカット。

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: **target**、**teams distribute parallel do** または **teams distribute parallel for** 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 [19] [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 構造を指定するショートカット。

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: **target**、**teams distribute parallel do simd** または **teams distribute parallel for simd** と同一の節が適用されます。

taskloop simd [19] [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** または **taskloop** と同一の節が適用されます。

teams distribute [19] [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** 構造を指定するショートカット。

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** と同一の節が適用されます。

teams distribute parallel do と teams distribute parallel for [19] [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** 構造を指定するショートカット。

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**、**distribute parallel for** または **distribute parallel do** と同一の節が適用されます。

teams distribute parallel do simd と teams distribute parallel for simd [19] [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** 構造を指定するショートカット。

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**、**distribute parallel for simd** または **distribute parallel do simd** と同一の節が適用されます。

teams distribute simd [19] [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** 構造を指定するショートカット。

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** と同一の節が適用されます。

節

修飾子: イテレーター [5.2.6] [3.2.6]

iterator (*イテレーター定義*)

節の中で複数の値に展開される識別子。

イテレーター定義 :
　イテレーター指定子 [, *イテレーター定義*]
*イテレーター*指定子 :
　[*イテレーター・タイプ*] *識別子* = *レンジ指定子*
識別子 : ベース言語の識別子
レンジ指定子 : [*begin* : *end* [: *step*]
　begin, *end*: 型を*イテレーター・タイプ*に変換できる式
　step : 整数式
イテレーター・タイプ : *C/C++* タイプ名。*For* タイプ指定子。

使用される節: **affinity**、**depend**、**from**、**map**、**to**

修飾子: デバイス名 [5.4]

デバイス名修飾子

各節には、その節を受け入れることができるディレクティブまたは複合ディレクティブの名前に対応する、オプションのディレクティブ名修飾子を含めることができます。
例:
C/C++
　#pragma parallel private (parallel: リスト)
　リスト内の項目が parallel 領域専用であることを指定します（この例ではデフォルト設定）。
For
　$!omp target teams distribute parallel do if (*target* : *条件 1*) **if** (*parallel* : *条件 2*)
　条件 1 は *target* 構成要素ディレクティブに適用され、*条件 2* は *parallel* 構成要素ディレクティブに適用されることを指定します。

使用される節: すべての節

affinity 節 [14.10] [12.5.1]

affinity (*ロケーターリスト*)

生成されたタスクの*ロケーターリスト*で指定されたデータのアフィニティーを示すヒントを指定します。

使用される節: target_data、task、task_iteration 節

allocate 節 [8.6] [6.6]

allocate (*[アロケーター :] リスト*)

allocate (*アロケーター修飾子 [, アロケーター修飾子]* : *リスト*)

アロケーター修飾子 :

allocator (*アロケーター*)

アロケーター : 次の表現

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 :正の整数乗（2 のべき乗）定数。

使用される節: allocators、distribute、do と for、parallel、scope、sections、single、target、target_data、task、taskgroup、taskloop、teams

apply 節 [11.1]

apply (*[ループ修飾子:] apply が適用されたディレクティブ*)

ループ変換構造を適用した後、*ループ修飾子*によって識別される変換後のループの前に適用済みディレクティブを挿入します。

使用される節: fuse、interchange、nothing、reverse、split、stripe、tile、unroll

collapse 節 [4.4.5] [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。

使用される節: distribute、do と for、loop、simd、taskloop

default 節 [7.5.1] [5.4.1]

default (*属性*)

デフォルトのデータ共有属性は無効になっています。構造内のすべての変数は、構造内で宣言するか、データ共有属性節に記述する必要があります。

属性 : shared、firstprivate、private、none

使用される節: parallel、target、target_data、task、taskloop、teams

depend 節 [17.9.5] [15.9.5]

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* : *ロケーターリスト*)

依存関係修飾子 : iterator (*イテレーター定義*)
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 *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。
- 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 *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、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

使用される節: dispatch、interop、target、target_enter_data、target_exit_data、target_update、task、task_iteration、taskwait

device 節 [15.2] [13.2]

device (*[ターゲットデバイス :] デバイス記述*)

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。
ターゲットデバイス : ancestor、device_num
デバイス記述 : 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 。

使用される節: dispatch、interop、target、target_data、target_enter_data、target_exit_data、

target_update

firstprivate 節 [7.5.4] [5.4.4]

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に固有のものとして宣言し、その構造が検出された時点での元の変数の値をそれらに割り当てます。

使用される節: distribute、do と for、parallel、scope、sections、single、target、target_data、task、taskloop、teams

if 節 [5.5] [3.4]

if (*[ディレクティブ名修飾子 :] omp 論理式*)

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。

使用される節: cancel、parallel、sim、target、target_data、target_enter_data、target_exit_data、target_update、task、task_iteration、taskgraph、taskloop、teams

induction 節 [7.6.13]

induction (*インダクション識別子 [, インダクション修飾子] [, ステップ修飾子] : リスト*)

ループの反復回数に対する式の閉形式を計算します。

インダクション識別子 : ID 式、または以下のいずれかの演算子 : +、*

インダクション修飾子 : strict、relaxed

strict: 各反復処理におけるインダクション値を計算し、リスト内の元の項目に割り当てます。

relaxed: 実装においては、各反復処理においてリスト内の元の項目に値を代入することなく、最後に閉形式の値が計算されることを前提としている場合があります。

ステップ修飾子 : step (*インダクション・ステップ*)

閉形式を計算する際、反復ごとに適用する増分。

使用される節: distribute、do と for、simd、taskloop

in_reduction 節 [7.6.12] [6.5.11]

in_reduction (*リダクション識別子 : リスト*)

タスクを、task_reduction 節または task を*リダクション識別子*として持つ reduction 節に現れる一致

するリスト項目の囲み領域によって定義されるタスク・リダクションの参加者として定義します。

C++ *リダクション識別子*:
ID 式または次の演算子のいずれか: +、*、&、|、^、&&、||
C *リダクション識別子*:
*識別子*または次の演算子のいずれか: +、*、&、|、^、&&、||

For *リダクション識別子*:
基本言語識別子、ユーザー定義演算子、または以下のいずれかの演算子:
+、*、.and.、.or.、.eqv.、.neqv.
または以下のいずれかの組み込みプロシージャー名:
max、min、iand、ior、ieor

使用される節: target、target_data、task、taskloop

lastprivate 節 [7.5.5] [5.4.5]

lastprivate (*[lastprivate 修飾子 :] リスト*)

*リスト*項目は、影響を受けるループのループ反復変数でなければなりません。private 節と同じ効果があり、領域を抜けた後、各変数がループの最後の反復の終了時に持っていた値を元の変数に代入します。

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。

使用される節: distribute、do と for、loop、sections、simd、taskloop

linear 節 [7.5.6] [5.4.6]

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: omp 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。

使用される節: declare simd、do と for、simd

map 節 [7.9.6] [5.8.3]

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 :] ロケーターリスト)

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 (*イテレーター定義*)

使用される節: declare_mapper、target、target_data、target_enter_data、target_exit_data

nowait 節 [17.6] [15.6]

nowait (*[値]*)

引数が指定されていない場合、デフォルト値は true になります。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。

使用される節: dispatch、do と for、interop、scope、sections、single、target、target_data、target_enter_data、target_exit

order 節 [12.3] [10.3]

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。

使用される節: distribute、do と for、loop、simd

priority 節 [14.9] [12.4]

priority (*優先度値*)

*優先度値*引数で、それが指定される構成要素のタスク優先度を指定します。

使用される節: target、target_data、target_enter_data、target_exit_data、target_update、

task、taskgraph、taskloop

private 節 [7.5.3] [5.4.3]

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。

使用される節: distribute、do と for、loop、parallel、scope、sections、simd、single、target、target_data、task、taskloop、teams

reduction 節 [7.6.10] [5.5.8]

reduction (*[リダクション修飾子,] リダクション識別子* : *リスト*)

*リダクション識別子*と 1 つ以上の*リスト*項目を指定します。

C++ リダクション識別子:

ID 式または次の演算子を指定します:

+, *, &, |, ^, &&, ||

C リダクション識別子:

*識別子*または次の演算子を指定します:

+, *, &, |, ^, &&, ||

For リダクション識別子:

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

使用される節: do と for、loop、parallel、scope、sections、simd、taskloop、teams

replayable 節 [14.6]

replayable

生成されたタスクを、taskgraph 構造の記録および再生対象としてマークします。

使用される節: target、target_enter_data、target_exit_data、target_update、task、taskloop、taskwait

shared 節 [7.5.2] [5.4.2]

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。

使用される節: parallel、target_data、task、taskloop、teams

task_reduction 節 [7.6.11, 7.6.7] [5.5.9]

task_reduction (*リダクション識別子* : *リスト*)

in_reduction 節を用いて、タスクまたは SIMD レーンによって*リダクション識別子*が計算される領域を定義する削減スコープ節。*リダクション識別子*の詳細については、reduction 節を参照してください。

使用される節: taskgroup

threadset 節 [14.8]

threadset (*セット*)

この節は、出現する構造によって生成されたタスクを実行できるスレッドの*セット*を指定します。遭遇するタスクが最終タスクである場合、threadset 節は無視されます。
セット: omp_team、omp_pool

使用される節: task、taskloop

ランタイム・ライブラリー・ルーチン

並列領域サポートルーチン

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 [21.2] [18.2.2]

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++	**void omp_get_num_threads (vioid);**
For	**integer function omp_get_num_threads ()**

omp_get_thread_num [21.3] [18.2.4]

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 (vioid);**
For	**integer function omp_get_thread_num ()**

omp_get_max_threads [21.4] [18.2.3]

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 (vioid);**
For	**integer function omp_get_max_threads ()**

omp_get_thread_limit [21.5] [18.2.13]

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 (vioid);**
For	**integer function omp_get_thread_limit ()**

omp_in_parallel [21.6] [18.2.5]

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 [21.7] [18.2.6]

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++	**int omp_set_dynamic (int** *dynamic_threads*);
For	**subroutine omp_set_dynamic (***dynamic_threads*) **logical** *dynamic_threads*

omp_get_dynamic [21.8] [18.2.7]

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++*）または +（*For*）を使用して、修飾子 *omp_sched_monotonic*（[20.5.1][18.2.11] を参照）と組み合わせます。

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

omp_set_schedule [21.9] [18.2.11]

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 (omp_sched_kind)** *kind* **integer** *chunk_size*

omp_set_schedule および **omp_get_schedule** の *kind* は、実装定義のスケジュール、または以下のいずれかです:

omp_sched_static、**omp_sched_dynamic**、**omp_sched_guided**、**omp_sched_auto**

修飾子 **omp_sched_monotonic** を使用して *kind* を組み合わせるには、+ または | 演算子（*C/C++*）または + 演算子（*For*）を使用します。

omp_get_schedule [21.10] [18.2.12]

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_set_schedule (_kind, chunk_size_)** **integer (omp_sched_kind) _kind_** **integer _chunk_size_**

kind については、**omp_set_schedule** を参照してください。

omp_get_supported_active_levels [21.11] [18.2.14]

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 (vioid);**
For	**integer function omp_get_supported_active_levels ()**

omp_set_max_active_levels [21.12] [18.2.15]

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_ supported_active_levels (int _max_levels_);**
For	**subroutine omp_set_ supported_active_levels (_max_levels_)** **integer _max_levels_**

omp_get_max_active_levels [21.13] [18.2.16]

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 (vioid);**
For	**integer function omp_get_ max_active_levels ()**

omp_get_level [21.14] [16.2.17]

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並列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 (vioid);**
For	**integer function omp_get_level ()**

omp_get_ancestor_thread_num [21.15] [18.2.18]

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** *level***);**
For	**integer function omp_get_ancestor_thread_num (***level***)** **integer** *level*

omp_get_team_size [21.16] [18.2.19]

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 [21.17] [18.2.20]

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 ()**

チーム領域ルーチン

omp_get_num_teams [22.1] [18.4.1]

現在の **teams** 領域のチーム数を返します。

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

omp_set_num_teams [22.2] [18.4.3]

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_team_num [22.3] [18.4.2]

呼び出しスレッドのチーム番号 *team-num-var* を返します。これは現在のチーム番号であり、0 から現在のチーム領域内のチーム数より 1 少ない値までの整数です。

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

omp_get_max_teams [22.4] [18.4.4]

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_get_teams_thread_limit [22.5] [18.4.6]

teams 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 ()

omp_set_teams_thread_limit [22.6] [18.4.5]

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 (*thread-limit*) integer *thread-limit*

タスク・サポート・ルーチン

omp_get_max_task_priority [23.1.1] [18.5.1]

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_explicit_task (void) [23.1.2]

遭遇したタスクが明示的なタスク領域である場合は、*true* を返します。

C/C++	int omp_in_explicit_task (void);
For	logical function omp_in_explicit_task ()

omp_in_final [23.1.3] [18.5.3]

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_is_free_agent [23.1.4]

ルーチンが呼び出された時点で、フリーエージェント・スレッドが囲んでいるタスク領域を実行している場合は *true* を返します。

C/C++	int omp_is_free_agent (void);
For	logical function omp_is_free_agent ()

omp_ancesor_is_free_agent [23.1.5]

遭遇したスレッドの祖先スレッドがフリーエージェント・スレッドである場合は、*true* を返します。

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

環境ルーチン

環境ルーチンは、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 [23.2.1] [18.11.1]

event 引数に関連付けられたイベントを作成または破棄します。

C/C++	int omp_fulfill_event (omp_event_handle_t *event*);
For	subroutine omp_fulfill_event (*event*) integer (omp_event_handle_kind) *event*

デバイス情報ルーチン

omp_set_default_device [24.1] [18.7.2]

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* の値を割り当てます。

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 [24.2] [18.7.3]

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* の値を返します。

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

omp_get_num_devices [24.3] [18.7.4]

コードまたはデータをオフロードに使用可能な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 [22.4] [18.7.5]

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_get_num_procs [24.5] [18.7.1]

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_get_max_progress_width [24.6]

device_num で指定されたデバイス上のプログレス単位の最大サイズを、ハードウェア・スレッド数で返します。

C/C++	**int omp_get_max_progress_width (int** *device_num*);
For	**integer function omp_get_max_progress_width (***device_num*) **integer** *device_num*

omp_get_device_from_uid [24.7]

uid 内の一意の文字列に関連付けられたデバイス番号を返します。デバイスが存在しない場合は **omp_invalid_device** を返します。

C/C++	**int omp_get_device_from_uid (const char *** *uid*);
For	**integer function omp_get_device_from_uid (***uid*) **character (len=*), intent (in) ::** *uid*

omp_get_uid_from_device [24.8]

デバイス *device_num* を識別する一意の文字列を返します。*device_num* の値が **omp_invalid_device** の場合は NULL を返します。

C/C++	**constant char * omp_get_uid_from_device (int** *device_num*);
For	**character(:) function omp_get_uid_from_device (***device_num*) **pointer :: omp_get_uid_from_device** **integer, intent (in) ::** *device_num*

omp_is_initial_device [24.9] [18.7.6]

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	**logical function omp_is_initial_device ()**

omp_get_initial_device [24.10] [18.7.7]

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	**logical function omp_get_initial_device ()**

omp_get_device_num_teams [24.11]

num_teams 節が指定されていない場合、デバイス *device_num* のチーム領域に対して要求されるチーム数を返します。

C/C++	**int omp_get_device_num_teams (int** *device_num*);
For	**integer function omp_get_device_num_teams (**_device_num_**)** **integer** *device_num*

omp_set_device_num_teams [24.12]

num_teams 節が指定されていない場合、デバイス *device_num* のチーム領域に対して要求されるチーム数を設定します。

C/C++	**void omp_set_device_num_teams (int** *num_teams*, **int** *device_num*);
For	**subroutine omp_set_device_num_teams (**_num_teams_, _device_num_**)** **integer** *num_teams*, *device_num*

omp_get_device_teams_thread_limit [24.13]

teams 構造によって生成される各競合グループでタスクを実行するのに使用可能なスレッドの最大数を返します。

C/C++	**int omp_get_device_teams_thread_limit (int** *device_num*);
For	**integer function omp_get_device_teams_thread_limit (**_device_num_**)** **integer** *device_num*

omp_set_device_teams_thread_limit [24.14]

teams 構造によって生成される各競合グループでタスクを実行するのに使用できるスレッドの最大数を定義します。

C/C++	**void omp_set_device_teams_thread_limit (** **int** *thread_limit*, **int** *device_num*);
For	**subroutine omp_set_device_teams_thread_limit (** *thread_limit*, *device_num***)** **integer** *thread_limit*, *device_num*

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_is_present [25.2.1] [18.8.3]

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, intent (in) :: *ptr* integer (c_int), value :: *device_num*

omp_target_is_accessible [25.2.2] [18.8.4]

指定されたメモリーアドレスとサイズの *ptr* が、指定されたデバイスからアクセス可能であるかテストします。

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_int, c_ptr, c_size_t type (c_ptr), value, intent (in) :: *ptr* integer (c_size_t), value :: *size* integer (c_int), value :: *device_num*

omp_get_mapped_ptr [25.2.3] [18.8.11]

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, intent (in) :: *ptr* integer (c_int), value :: *device_num*

omp_target_alloc [25.3] [18.8.1]

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*

omp_target_free [25.4] [18.8.2]

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_associate_ptr [25.5] [18.8.9]

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_int, c_ptr, c_size_t type (c_ptr), value, intent (in) :: _host_ptr_, _device_ptr_ integer (c_size_t), value :: _size_, _device_offset_ integer (c_int), value :: _device_num_

omp_target_disassociate_ptr [25.6] [18.8.10]

ホスト _ptr_ のプレゼンステーブルから、デバイス _device_num_ に関連付けられたデバイスデータを削除します。

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, intent (in) :: _ptr_ integer (c_int), value :: _device_num_

omp_target_memcpy [25.7.1] [18.8.5]

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_int, c_ptr, c_size_t type (c_ptr), value :: *dst* type (c_ptr), value, intent (in) :: *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 [25.7.2] [18.8.6]

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_int, c_ptr, c_size_t type (c_ptr), value :: *dst* type (c_ptr), value, intent (in) :: *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 [25.7.3] [18.8.7]

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_memory_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_memory_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_int, c_ptr, c_size_t type (c_ptr), value :: *dst* type (c_ptr), value, intent(in) :: *src* integer (c_size_t), value :: *length*, *dst_offset*, *src_offset* integer (c_int), intent(in) :: *dst_device_num*, *src_device_num*, *depobj_count* integer (omp_depend_kind), optional :: *depobj_list*(*)

omp_target_memcpy_rect_async [25.7.4] [18.8.8]

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_int, c_ptr, c_size_t type (c_ptr), value :: *dst* type (c_ptr), value, intent(in) :: *src* integer (c_size_t), value :: *element_size* integer (c_int), value :: *num_dims*, *dst_device_num*, *src_device_num*, *depobj_count* integer (c_int), intent(in) :: *volume*(*), *dst_offsets*(*), *src_offsets*(*), *dst_dimensions*(*), *src_dimensions*(*) integer (omp_depobj_kind), optional :: *depobj_list*(*)

omp_target_memset [25.8.1]

device_num に関連付けられたデバイスのデバイス環境で、*ptr* で示される最初の *count* バイトを値 *val* で埋めます。

C/C++	int omp_target_memset (void *_ptr_, int _val_, size_t _count_, int _device_num_);
For	type (c_int) function omp_target_memset (*ptr*, *val*, *count*, *device_num*) bind (c) use, intrinsic :: iso_c_binding, only : c_int, c_ptr, c_size_t type (c_ptr), value :: *ptr* integer (c_int), value :: *val*, *device_num* integer (c_size_t), value :: *count*

omp_target_memset_async [25.8.2]

ptr で示される最初の *count* バイトを、*num_device* に関連付けられたデバイスのデバイス環境の値 *val* で非同期的にフィルします。

C/C++	int omp_target_memset (void *_ptr_, int _val_, size_t _count_, int _device_num_, int _depobj_count_, omp_depobj_t *_depobj_list_);
For	type(c_int) function omp_target_memset (*ptr*, *val*, *count*, *device_num*, *depobj_count*, *depobj_list*) bind (c) use, intrinsic :: iso_c_binding, only : c_int, c_ptr, c_size_t type (c_ptr), value :: *ptr* integer (c_int), value :: *val*, *device_num*, *depobj_count* integer (c_size_t), value :: *count* integer (omp_depobj_kind), optional :: *depobj_list*(*)

相互運用ルーチン

omp_get_num_interop_properties [26.1] [18.12.1]

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_);
For	integer function omp_get_num_interop_properties (*interop*) integer (omp_interop_kind), intent (in) :: *interop*

omp_get_interop_int [26.2] [18.12.2]

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++	int omp_get_interop_int (const omp_interop_t *interop*, omp_interop_property_t *property_id*, int *ret_code*);
For	integer (c_intptr_t) function omp_get_interop_int (*interop*, *property_id*, *ret_code*) use, intrinsic :: iso_c_binding, only : c_intptr_t integer (omp_interop_kind), intent (in) :: *interop* integer (omp_interop_property_kind) *property_id* integer (omp_interop_rc_kind), intent (out), optional :: *ret_code*

omp_get_interop_ptr [26.3] [18.12.3]

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*);
For	integer (c_ptr) function omp_get_interop_ptr (*interop*, *property_id*, *ret_code*) use, intrinsic :: iso_c_binding, only : c_ptr integer (omp_interop_kind), intent (in) :: *interop* integer (omp_interop_property_kind) *property_id* integer (omp_interop_rc_kind), intent (out), optional :: *ret_code*

omp_get_interop_str [26.4] [18.12.4]

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*);
For	character (:) function omp_get_interop_str (*interop*, *property_id*, *ret_code*) pointer :: omp_get_interop_str integer (omp_interop_kind), intent (in) :: *interop* integer (omp_interop_property_kind) *property_id* integer (omp_interop_rc_kind), intent (out), optional :: *ret_code*

omp_get_interop_name [26.5] [18.12.5]

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 (const omp_interop_t *interop*, omp_interop_property_t *property_id*);

For	character (:) function omp_get_interop_name (*interop*, *property_id*) pointer :: omp_get_interop_name integer (omp_interop_kind), intent (in) :: *interop* integer (omp_interop_property_kind) *property_id*

omp_get_interop_type_desc [26.6] [18.12.6]

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*);
For	character (:) function omp_get_interop_type_desc (*interop*, *property_id*) pointer :: omp_get_interop_type_desc integer (omp_interop_kind), intent (in) :: *interop* integer (omp_interop_property_kind) *property_id*

omp_get_interop_rc_desc [26.7] [18.12.7]

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*);
For	character (:) function omp_get_interop_rc_desc (*interop*, *ret_code*) pointer :: omp_get_interop_rc_desc integer (omp_interop_kind), intent (in) :: *interop* integer (omp_interop_rc_kind) *ret_code*

メモリー空間ルーチン

omp_get_devices_memspace [27.1.1]

devs 引数で指定されたデバイスのメモリー領域を解放します。

C/C++	omp_memspace_handle_t omp_get_devices_memspace (int *ndevs*, const int **devs*, omp_memspace_handle_t *memspace*);
For	character (:) function omp_get_devices_memspace (*ndevs*, *devs*, *memspace*) integer, intent (in) :: *ndevs*, *devs* (*) integer (omp_memspace_handle_kind), intent (in) :: *memspace*

omp_get_device_memspace [27.1.2]

選択されたデバイスは、*dev* 引数で指定されたデバイスです。

C/C++	omp_memspace_handle_t omp_get_device_memspace (int *dev*, omp_memspace_handle_t *memspace*);
For	integer (omp_memspace_handle_kind) function omp_get_device_memspace (*devs*, *memspace*) integer, intent (in) :: *dev* integer (omp_memspace_handle_kind), intent (in) :: *memspace*

omp_get_devices_and_host_memspace [27.1.3]

ホストデバイスおよび *devs* 引数で指定されたデバイスのメモリー領域を解放します。

C/C++	omp_memspace_handle_t omp_get_devices_and_host_memspace (int *ndevs*, const int **devs*, omp_memspace_handle_t *memspace*);
For	integer (omp_memspace_handle_kind) function omp_get_devices_and_host_memspace (*ndevs*, *devs*, *memspace*) integer, intent (in) :: *ndevs*, *devs* (*) integer (omp_memspace_handle_kind), intent (in) :: *memspace*

omp_get_device_and_host_memspace [27.1.4]

ホストデバイスと *dev* 引数で指定されたデバイスのメモリー領域を取得します。

C/C++	omp_memspace_handle_t omp_get_device_and_host_memspace (int *dev*, omp_memspace_handle_t *memspace*);
For	integer (omp_memspace_handle_kind) function omp_get_device_and_host_memspace (*dev*, *memspace*) integer, intent (in) :: *dev* integer (omp_memspace_handle_kind), intent (in) :: *memspace*

omp_get_devices_all_memspace [27.1.5]

利用可能なすべてのデバイスを選択するメモリー領域取得ルーチンです。

C/C++	omp_memspace_handle_t omp_get_devices_all_memspace (omp_memspace_handle_t *memspace*);
For	integer (omp_memspace_handle_kind) function omp_get_devices_all_memspace (*memspace*) integer (omp_memspace_handle_kind), intent (in) :: *memspace*

メモリー管理ルーチン

omp_get_memspace_num_resources [27.2]

memspace で表されるメモリー空間に関連付けられている、異なるストレージリソースの数を返します。

C/C++	omp_memspace_handle_t omp_get_memspace_num_resources (omp_memspace_handle_t *memspace*);
For	integer (omp_memspace_handle_kind) function omp_get_memspace_num_resources (*memspace*) integer (omp_memspace_handle_kind), intent (in) :: *memspace*

omp_get_memspace_pagesize [27.3]

memspace ハンドルが表すメモリー空間がサポートするページサイズを返します。

C/C++	omp_memspace_handle_t omp_get_memspace_pagesize (omp_memspace_handle_t *memspace*);
For	integer (c_size_t) function omp_get_memspace_pagesize (*memspace*) bind (c) use, intrinsic :: iso_c_binding, only : c_size_t integer (omp_memspace_handle_kind), intent (in) :: *memspace*

omp_get_submemspace [27.4]

元のメモリー空間のリソースのサブセットを含む新しいメモリー空間を返します。

C/C++	omp_memspace_handle_t omp_get_submemspace (omp_memspace_handle_t *memspace* int *num_resources*, const int **resources*);
For	integer (omp_memspace_handle_kind) function omp_get_submemspace (*memspace*, *num_resources*, *resources*) integer (omp_memspace_handle_kind), intent (in) :: *memspace* integer, intent (in) :: *num_resources*, *resources* (*)

omp_init_allocator [27.6] [18.13.2]

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_memspace_handle_t omp_init_allocator (omp_memspace_handle_t *memspace* int *ntraits*, const omp_alloctrait **traits*);
For	integer (omp_memspace_handle_kind) function omp_init_allocator (*memspace*, *ntraits*, *traits*) integer (omp_memspace_handle_kind), intent (in) :: *memspace* integer, intent (in) :: *ntraits* type (omp_alloctrait), intent (in) :: *traits* (*)

omp_destroy_allocator [27.7] [18.13.3]

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 (omp_allocator_handle_kind), intent (in) :: *allocator*

omp_set_default_allocator [27.9] [18.13.4]

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 (omp_allocator_handle_kind), intent (in) :: *allocator*

omp_get_default_allocator [27.10] [18.13.5]

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++	void omp_get_default_allocator (void);
For	integer (omp_allocator_handle_kind) function omp_get_default_allocator ()

メモリー・パーティショニング・ルーチン

omp_init_mempartitioner [27.5.1]

パーティション・オブジェクトが表すメモリー・パーティションを、存続期間（*lifetime*）、計算手順（*compute_proc*）、および解放手順（*release_proc*）で初期化します。

C/C++	void omp_init_mempartitioner (omp_mempartitioner_t **partitioner*, omp_mempartitioner_lifetime_t *lifetime*, omp_mempartitioner_compute_proc_t *compute_proc*, omp_mempartitioner_release_proc_t *release_proc*);
For	subroutine omp_init_mempartitioner (*partitioner*, *lifetime*, *compute_proc*, *release_proc*) integer (omp_mempartitioner_kind), intent (out) :: *partitioner* integer (omp_mempartitioner_lifetime_kind), intent (in) :: *lifetime* integer (omp_mempartitioner_compute_proc_t) *compute_proc* integer (omp_mempartitioner_release_proc_t) *release_proc*

omp_destroy_mempartitioner [27.5.2]

メモリー・パーティショナーの初期化を解除し、メモリー・パーティショナーの状態を初期化解除して、リソースを解放します。

C/C++	void omp_destroy_mempartitioner (const omp_mempartitioner_t **partitioner*);
For	subroutine omp_destroy_mempartitioner (*partitioner*) integer (omp_mempartitioner_kind), intent (in) :: *partitioner*

omp_init_mempartition [27.5.3]

nparts 個のパーツを持つメモリー・パーティション・オブジェクトを初期化し、*user_data* 引数をそれに関連付けます。

C/C++	void omp_init_mempartition (omp_mempartition_t **partition*, size_t *nparts*, const void **user_data*);
For	subroutine omp_init_mempartition (*partition*, *nparts*, *user_data*) bind (c) use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t integer (omp_mempartition_kind), intent (out) :: *partition*

	integer (c_size_t), intent (in) :: *nparts* type (c_ptr), intent (in) :: *user_data*

omp_destroy_mempartition [27.5.4]

メモリー・パーティション・オブジェクトの初期化を解除し、メモリー・パーティションとそのリソースを解放します。

C/C++	void omp_destroy_mempartition (const omp_mempartition_t *_partition_); const omp_mempartition_t *_partition_);
For	subroutine omp_destroy_mempartition (*partition*) integer (omp_mempartition_kind), intent (in) :: *partition*

omp_mempartition_set_part [27.5.5]

メモリー・パーティションの特定部分のサイズとリソースを定義します。

C/C++	int omp_mempartition_set_part (const omp_mempartition_t *_partition_, size_t *part*, int *resource*, size_t *size*);
For	integer function omp_mempartition_set_part (*partition*, *part*, *resource*, *size*) bind (c) use, intrinsic :: iso_c_binding, only : c_size_t integer (omp_mempartition_kind), intent (out) :: *partition* integer (c_size_t), intent (in) :: *part*, *size* integer, intent (in) :: *resource*

omp_mempartition_get_user_data [27.5.6]

メモリー・パーティションが作成された際に、そのパーティションに関連付けられていたユーザーデータを取得して返します。

C/C++	void *omp_mempartition_get_user_data (const omp_mempartition_t *_partition_);
For	type (c_ptr) function omp_mempartition_get_user_data (*partition*) bind (c) use, intrinsic :: iso_c_binding, only : c_ptr integer (omp_mempartition_kind), intent (in) :: *partition*

メモリー割り当て取得ルーチン

omp_get_devices_allocator [27.8.1]

devs 引数で指定されたデバイス用のメモリー割り当て取得ルーチン。

C/C++	omp_allocator_handle_t omp_get_devices_allocator (int *ndevs*, const int *devs*, omp_memspace_handle_t *memspace*);
For	integer (omp_allocator_handle_kind) function omp_get_devices_allocator (*ndevs*, *devs*, *memspace*) integer, intent (in) :: *ndevs*, *devs* (*) integer (omp_memspace_handle_kind), intent (in) :: *memspace*

omp_get_device_allocator [27.8.2]

デバイス *dev* 用のメモリー・アロケーターを取得します。

C/C++	omp_allocator_handle_t omp_get_device_allocator (int *dev*, omp_memspace_handle_t *memspace*);
For	integer (omp_allocator_handle_kind) function omp_get_device_allocator (*dev*, *memspace*) integer, intent (in) :: *dev* integer (omp_memspace_handle_kind), intent (in) :: *memspace*

omp_get_devices_and_host_allocator [27.8.3]

ホストデバイスと *devs* で指定されたデバイス用のメモリー割り当てを取得します。

C/C++	omp_allocator_handle_t omp_get_devices_and_host_allocator (int *ndevs*, const int *devs*, omp_memspace_handle_t *memspace*);
For	integer (omp_allocator_handle_kind) function omp_get_devices_and_host_allocator (*ndevs*, *devs*, *memspace*) integer, intent (in) :: *ndevs*, *devs* (*) integer (omp_memspace_handle_kind), intent (in) :: *memspace*

omp_get_device_and_host_allocator [27.8.4]

ホストデバイスとデバイス *dev* 用のメモリー割り当てを取得します。

C/C++	omp_allocator_handle_t omp_get_device_and_host_allocator (

	int *dev*, omp_memspace_handle_t *memspace*);
For	integer (omp_allocator_handle_kind) function omp_get_device_and_host_allocator (*dev*, *memspace*) integer, intent (in) :: *dev* integer (omp_memspace_handle_kind), intent (in) :: *memspace*

omp_get_devices_all_allocator [27.8.5]

利用可能なすべてのデバイス用のメモリー・アロケーターを取得します。

C/C++	omp_allocator_handle_t omp_get_devices_all_allocator (omp_memspace_handle_t *memspace*);
For	integer (omp_allocator_handle_kind) function omp_get_devices_all_allocator (*memspace*) integer (omp_memspace_handle_kind), intent (in) :: *memspace*

メモリー割り当てルーチン

omp_alloc と omp_aligned_alloc [27.11.1-2] [18.13.6]

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 *alignment*, size_t *size*, omp_allocator_handle_t *allocator*);
C++	void *omp_alloc (size_t *size*, omp_allocator_handle_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_calloc と omp_aligned_calloc [27.11.3-4] [18.13.8]

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 *size*, omp_allocator_handle_t *allocator*);
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 [27.11.5] [18.13.9]

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++	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*

omp_free [27.12] [18.13.7]

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 (c_size_t), value :: *size* integer (omp_allocator_handle_kind), value :: *allocator*

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（*svar*）と入れ子可能なロック（*nvar*）。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 [28.1.1-2] [18.9.1]

OpenMP ロックを初期化します。

C/C++	void omp_init_lock (omp_lock_t *svar); void omp_init_nest_lock (omp_nest_lock_t *nvar);
For	subroutine omp_init_lock (*svar*) integer (omp_lock_kind) *svar* subroutine omp_init_nest_lock (*nvar*) integer (omp_nest_lock_kind) *nvar*

ヒント付きのロックの初期化 [28.1.3-4] [18.9.2]

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 *svar, omp_sync_hint_t *hint*); void omp_init_nest_lock_with_hint (omp_nest_lock_t *nvar, omp_sync_hint_t *hint*);
For	subroutine omp_init_lock_with_hint (*svar*, *hint*)

	integer (omp_lock_kind) *svar* integer (omp_sync_hint_kind) *hint* subroutine omp_init_nest_lock_with_hint (*nvar*, *hint*) integer (omp_nest_lock_kind) *nvar* integer (omp_sync_hint_kind) *hint*

hint: 仕様の [20.9.5] [15.1] を参照してください。

ロックの破棄 [28.2] [18.9.3]

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 **svar*); void omp_destroy_nest_lock (omp_nest_lock_t **nvar*);
For	subroutine omp_destroy_lock (*svar*) integer (omp_lock_kind) *svar* subroutine omp_destroy_nest_lock (*nvar*) integer (omp_nest_lock_kind) *nvar*

ロックの設定 [28.3] [18.9.4]

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 **svar*); void omp_set_nest_lock (omp_nest_lock_t **nvar*);
For	subroutine omp_set_lock (*svar*) integer (omp_lock_kind) *svar* subroutine omp_set_nest_lock (*nvar*) integer (omp_nest_lock_kind) *nvar*

ロックの解除 [28.4] [18.9.5]

OpenMP ロックを解除するか、ネスト数を減らします。

C/C++	void omp_unset_lock (omp_lock_t **svar*); void omp_unset_nest_lock (omp_nest_lock_t **nvar*);

For	subroutine omp_unset_lock (*svar*) integer (omp_lock_kind) *svar* subroutine omp_unset_nest_lock (*nvar*) integer (omp_nest_lock_kind) *nvar*

ロックをテスト [28.5] [18.9.6]

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 **svar*); void omp_test_nest_lock (omp_nest_lock_t **nvar*);
For	subroutine omp_test_lock (*svar*) integer (omp_lock_kind) *svar* subroutine omp_test_nest_lock (*nvar*) integer (omp_nest_lock_kind) *nvar*

スレッド・アフィニティー・ルーチン

omp_get_proc_bind [29.1] [18.3.1]

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 (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 [29.2] [18.3.2]

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 [29.3] [18.3.3]

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 [29.4] [18.3.4]

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 [29.5] [18.3.5]

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 ()

omp_get_partition_num_places [29.6] [18.3.6]

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 [29.7] [18.3.7]

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 [29.8] [18.3.8]

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 [29.9] [18.3.9]

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_);
For	integer function omp_get_affinity_format (_buffer_) character (len=*), intent (out) :: _buffer_

omp_display_affinity [29.10] [18.3.10]

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 [29.11] [18.3.11]

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_

実行制御ルーチン

omp_get_cancellation [30.8] [18.2.8]

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	integer function omp_get_cancellation ()

omp_pause_resource [30.2.1] [18.6.1]

omp_pause_resource_all [30.2.2] [18.6.2]

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 (omp_pause_resource_kind)** *kind* **integer** *device_num* **integer function omp_pause_resource_all (***kind*) **integer (omp_pause_resource_kind)** *kind*

omp_display_env [30.4] [18.15]

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*

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 [30.3.1] [18.10.1]

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 [30.3.2] [18.10.2]

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 ()**

ツール・サポート・ルーチン

omp_control_tool [31.1] [18.14]

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++	**omp_control_tool_result_t omp_control_tool (** **omp_control_tool_t** *command*, **int** *modifier*, **void** ***arg**);**
For	**integer (omp_control_tool_result_kind) function omp_control_tool (** *command*, *modifier*) **integer (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* [4.3.5] [21.2.5]

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。*format* 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 21.2]。

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 *allocator* [4.4.1] [21.5.1]

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 8.1 6.1 に示します。
- アロケーターの特性を表 8.2 6.2 に示します。
- 事前定義されたアロケーターを表 8.3 6.3 に示します。

例

```
export OMP_ALLOCATOR = omp_high_bw_mem_alloc

export OMP_ALLOCATOR = “omp_large_cap_mem_space : alignment=16, pinned=true”

export OMP_ALLOCATOR = “omp_high_bw_mem_space : pool_size=1048576, ¥
fallback=allocator_fb, fb_data=omp_low_lat_mem_alloc”
```

表 8.1 6.1 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 8.2 6.2 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（デフォルトは false）
partition	environment、nearest、blocked、interleaved（デフォルトは environment）
pin_device	適合デバイス番号（デフォルトなし）
preferred_device	適合デバイス番号（デフォルトなし）
target_access	single、multiple（デフォルトは single）
atomic_scope	all、device（デフォルトは device）
part_data	正の整数値（デフォルトは実装定義）
partitioner	メモリー・パーティショナーのハンドル（デフォルトなし）
partitioner_arg	整数値（デフォルトは 0）

表 8.3 6.3 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

OMP_AVAILABLE_DEVICES *list* [4.3.7]

内部変数 *available-devices-var* を設定し、利用可能な非ホストデバイスとそのデバイス番号を決定しま

す。*list* の値は、特性仕様をカンマで区切ったリスト、または * である必要があります。* は、残りのすべてのデバイスを表します。各リスト項目は、サポートされアクセス可能で、かつそれ以前のリスト項目と重複しない、該当するデバイスのセットに展開されます。結果として得られるデバイスには、順番にデバイス番号が割り当てられます。

OMP_CANCELLATION *cancelstate* [4.3.6] [21.2.6]

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、キャンセルがアクティブになります。

OMP_DEBUG *debugstate* [4.6.1] [21.4.1]

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* [4.3.8] [21.2.7]

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* [4.3.4] [21.2.4]

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* [4.7] [21.7]

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* [4.1.2] [21.1.1]

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* [4.1.5] [21.1.4]

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* [4.3.11] [21.2.9]

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 [4.2.1] [21.6.1]

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 *list* [4.1.3] [21.1. 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* [4.1.6] [21.1.6]

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* [4.1.7] [21.1.7]

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で区切った **primary**、**close**、もしくは **spread** リストを指定できます。

OMP_SCHEDULE *[modifier:] kind [,chunk]* [4.3.1] [21.2.1]

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[*unit*]* [4.3.2] [21.2.2]

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。*unit* の値は測定単位を示しており、B はバイト、K はキロバイト、M はメガバイト、G はギガバイトを表します。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 *state* [4.3.9] [21.2.8]

state には *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 *number* [4.2.2] [21.6.2]

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* [4.1.4] [21.1.3]

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_THREADS_RESERVE *list* [4.3.10]

競合グループ内で予約する構造化スレッドとフリーエージェント・スレッドの数を制御するには、内部制御変数 *structured-thread-limit-var* と *free-agentthread-limit-var* を適切に設定します。*list* の値は、カンマで区切った予約リストです。予約は、**n**、**structured** *[(n)]*、または **free_agent** *[(n)]* で表され、*n* は非負の整数です。

OMP_TOOL *toolstate* [4.5.1] [21.3.1]

内部制御変数 *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* [4.5.2] [21.3.2]

内部制御変数 *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* は、絶対パスで指定されたコロンで区切ったダイナミック・リンク・ライブラリーのリストです。

OMP_TOOL_VERBOSE_INIT *value* [4.5.3] [21.3.3]

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。*value* は、ファイル名、**disabled**、**stdout**、または **stderr** のいずれかです。

OMP_WAIT_POLICY *policy* [4.3.3] [21.2.3]

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。デフォルトは実装定義です。

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 3.1-3] [表 2.1-3]

ICV	環境変数	初期値	値の変更	値の取得	スコープ
active-levels-var	（なし）	*ゼロ*	（なし）	omp_get_active_level()	データ環境
affinity-format-var	OMP_AFFINITY_FORMAT [4.3.5] [21.2.5]	実装定義	omp_set_affinity_format()	omp_get_affinity_format()	デバイス
available-devices-var	OMP_AVAILABLE_DEVICES [4.3.7]	サポート対象でアクセス可能なすべてのデバイスのセット。	（なし）	（なし）	グローバル
bind-var	OMP_PROC_BIND [4.1.7] [21.1.7]	実装定義	（なし）	omp_get_proc_bind()	データ環境
cancel-var	OMP_CANCELLATION [4.3.6] [21.2.6]	false	（none）	omp_get_cancellation()	グローバル
debug-var	OMP_DEBUG [4.6.1] [21.4.1]	無効	（なし）	（なし）	グローバル
def-allocator-var	OMP_ALLOCATOR [4.4.1] [21.5.1]	実装定義	omp_set_default_allocator()	omp_get_default_allocator()	暗黙のタスク
default-device-var	OMP_DEFAULT_DEVICE [4.3.8] [21.2.7]	実装定義	omp_set_default_device()	omp_get_default_device()	データ環境
device-num-var	（なし）	*ゼロ*	（なし）	omp_get_device_num()	デバイス
display-affinity-var	OMP_DISPLAY_AFFINITY [4.3. 4] [21.2.4]	false	（なし）	（なし）	グローバル
dyn-var	OMP_DYNAMIC [4.1.2] [21.1.1]	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
explicit-task-var	（なし）	false	（なし）	omp_in_explicit_task()	
final-task-var	（なし）	false	（なし）	omp_in_final()	データ環境

free-agent-thread-limit-var	OMP_THREAD_LIMIT [4.1.4] [21.1.3] OMP_THREADS_RESERVE [4.3.10]	*thread-limit-var* の初期値より 1 つ少ない値。	(なし)	(なし)	データ環境
free-agent-var	(なし)	false	(なし)	omp_is_free_agent()	データ環境
league-size-var	(なし)	1	(なし)	omp_get_num_teams()	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 [4.1.5] [21.1.4] OMP_NUM_THREADS [4.1.3] [21.1.2] OMP_PROC_BIND [4.1.7] [21.1.7]	実装定義	omp_set_max_active_levels()	omp_get_max_active_levels()	デバイス データ環境
max-task-priority-var	OMP_MAX_TASK_PRIORITY [4.3.11] [21.2.9]	*ゼロ*	(なし)	omp_get_max_task_priority()	グローバル
nteams-var	OMP_NUM_TEAMS [4.2.1] [21.6.1]	*ゼロ*	omp_set_num_teams()	omp_get_max_teams()	デバイス
nthreads-var	OMP_NUM_THREADS [4.1.3] [21.1.2]	実装定義	omp_set_device_num_teams() omp_set_num_threads()	omp_get_max_threads()	データ環境
num-devices-var	(なし)	実装定義	(なし)	omp_get_num_devices()	グローバル
num-procs-var	(なし)	実装定義	(なし)	omp_get_num_procs()	デバイス
place-assignment-var	(なし)	実装定義	(なし)	(なし)	暗黙のタスク
place-partition-var	OMP_PLACES [4.1.6] [21.1.6]	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()	実装タスク
run-sched-var	OMP_SCHEDULE [4.3.1] [21.2.1]	実装定義	omp_set_schedule()	omp_get_schedule()	データ環境
stacksize-var	OMP_STACKSIZE [4.3.2] [21.2.2]	実装定義	(なし)	(なし)	デバイス
structures-thread-limit-var	OMP_THREAD_LIMIT [4.1.4] [21.1.3] OMP_THREADS_RESERVE [4.3.10]	*thread-limit-var* の初期値。	(なし)	(なし)	データ環境
target-offload-var	OMP_TARGET_OFFLOAD [4.3.9] [21.2.8]	デフォルト	(なし)	(なし)	グローバル
team-generator-var	(なし)	*ゼロ*	(なし)	(なし)	データ環境

team-num-var	（なし）	*ゼロ*	（なし）	omp_get_team_num()	データ環境
team-size-var	（なし）	*1*	（なし）	omp_get_num_threads()	データ環境
teams-thread-limit-var	OMP_TEAMS_THREAD_LIMIT [4.2.2] [21.6.2]	*ゼロ*	omp_set_device_thread_limit() omp_set_teams_thread_limit()	omp_get_teams_thread_limit()	デバイス
thread-limit-var	OMP_THREAD_LIMIT [4.1.4] [21.1.3]	実装定義	target および teams 構造の thread_limit 節	omp_get_thread_limit()	データ環境
thread-num-var	（なし）	*ゼロ*	（なし）	omp_get_thread_num()	データ環境
tool-libraries-var	OMP_TOOL_LIBRARIES [4.5.2] [21.3.2]	空の文字列	（なし）	（なし）	グローバル
tool-var	OMP_TOOL [4.5.1] [21.3.1]	有効	（なし）	（なし）	グローバル
tool-verbose-init-var	OMP_TOOL_VERBOSE_INIT [4.5.3] [21.3.3]	無効	（なし）	（なし）	グローバル
wait-policy-var	OMP_WAIT_POLICY [4.3.3] [21.2.3]	実装定義	（なし）	（なし）	デバイス

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 ARB メンバーになるには

OpenMP アーキテクチャー・レビューボード（ARB）の強みは、その多様なメンバー構成にあります。

OpenMP API に依存する製品を持つ大小さまざまな企業、そして OpenMP API の進化に関心のある組織は、OpenMP ARB への参加を歓迎します。

OpenMP ARB は、メンバーに開発者向けリソース、API への影響力、API ロードマップへのアクセス、共同マーケティングの機会、OpenMP コミュニティーへの参加、そして OpenMP API 関連の製品、サービス、研究の広報に OpenMP 商標およびロゴの使用権を提供します。

詳細は openmp.org/join をご覧ください。

OpenMP API 6.0 リファレンス・ガイド索引