OpenMP 5.1 API シンタックス・クイック・リファレンス・カード

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

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

C/C++	C/C++ 向け	For	Fortran 向け

OpenMP 5.1 で追加/変更された機能はこの色で示し、OpenMP 5.0 の機能はこの色で示します。

[n.n.n] は 5.1 のセクションを示し、 **[n.n.n]** は 5.0 のセクションを示します。
● は、5.1 では非標準となった機能、● は 5.0 で非標準となった機能を示します。

ディレクティブと構造

OpenMP 実行ディレクティブは、後続の構造化ブロックに適用されます。構造化ブロックは、OpenMP 構造、または単一の入口と出口を持つ実行文のブロックです。SIMD と **declare** ディレクティブを除く OpenMP ディレクティブは、**PURE** プロシージャー内では使用できません。

バリアント・ディレクティブ

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

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

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

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

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

C/C++
```
#pragma omp declare variant(variant-func-id) 節
[#pragma omp declare variant(variant-func-id) 節]
    [...]
    関数定義または宣言
- または -
#pragma omp declare variant 節
    宣言定義シーケンス
#pragma omp end declare variant
```

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

節:
match(*context-selector-specification*)
adjust_args (*adjust* 操作 : 引数リスト)
append_args (*append* 操作 [[, *append* 操作] ...])

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

dispatch [2.3.6]
特定の呼び出しに対してバリアント置換を行うか制御します。

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

For
```
!$ omp dispatch [節 [[,] 節] ...]
    文
```

節:
depend ([*depend* 修飾子,] 依存関係タイプ : *locator* リスト)
nowait
is_device_ptr (リスト)
C/C++ **device** (整数式)
novariants (スカラー式)
C/C++ **nocontext** (スカラー式)
For **device** (スカラー整数式)
For **novariants** (スカラー論理式)
For **nocontext** (スカラー論理式)

情報とユーティリティー・ディレクティブ

requires [2.5.1] [2.4]
コードをコンパイルして正しく実行するために実装が提供しなければならない機能を指定します。

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

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

節:
reverse_offload
unified_address
unified_shared_memory
atomic_default_mem_order(&
seq_cst | acq_rel | relaxed)
dynamic_allocators
ext_実装定義の要件

assumes と assume [2.5.2]
最適化に使用可能な実装の不変条件を提供します。

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

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

節:
assumption 節
ext_実装定義の要件
assumption 節:
absent (*directive* 名 [[,*directive* 名]...])
contains (*directive* 名 [[,*directive* 名]...])
no_openmp
no_openmp_routines
no_parallelism
C/C++ **holds** (スカラー式)
For **holds** (スカラー論理式)

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

C/C++
```
#pragma omp nothing
```

For
```
!$ omp nothing
```

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

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

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

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

並列構造

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

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

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

節:
default (データ共有属性)、**private**(リスト)、**firstprivate**(リスト)、**shared**(リスト)
copyin(リスト)
reduction([*reduction* 修飾子,] *reduction* 識別子: リスト)
proc_bind(**primary**|●**master** |**close** | **spread**)
allocate ([*allocator:*] リスト)
C/C++ **if** ([***parallel:***] スカラー式)
C/C++ **num_threads** (整数式)
For **if** ([***parallel:***] スカラー論理式)
For **num_threads** (スカラー整数式)

チーム構造

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

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

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

節:
private(リスト)、**firstprivate**(リスト)、**shared**(リスト)
reduction([**default**,] *reduction* 識別子: リスト)
allocate ([*allocator:*] リスト)
default (データ共有属性)
num_teams([下限:] 上限)
C/C++ **thread_limit**(整数式)
For **thread_limit**(スカラー整数式)

ディレクティブと構造 (続き)

マスク構造

masked [2.8] [2.16]

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。[5.0 ではこれは、master 構造であり、master は masked を置き換えます。]

C/C++	**#pragma omp masked** *[**filter** (整数式)]* 構造化ブロック
For	**!$ omp masked** *[**filter** (スカラー整数式)]* 緩い構造化ブロック **!$ omp end masked** - または – **!$ omp masked** *[**filter** (スカラー整数式)]* 厳密な構造化ブロック *[**!$ omp end masked**]*

スコープ構造

scope [2.9]

チーム内のすべてのスレッドで実行され、追加で 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:

private(リスト)
reduction(*[reduction 修飾子,]* & *reduction 識別子:* リスト)
C/C++ **nowait**

ワークシェア構造

sections [2.10.1] [2.8.2]

チーム内のスレッドに分散され実行される一連の構造化ブロックを含む非反復型のワークシェア構造です。

C/C++	**#pragma omp sections** *[節[[,] 節] ...]* **{** *[**#pragma omp section**]* 構造化ブロック *[**#pragma omp section**]* 構造化ブロック] … **}**
For	**!$ omp sections** *[節[[,] 節] ...]* **{** *[**!$ omp section**]* 構造化ブロック *[!$ **omp section**]* 構造化ブロック] … **!$ omp and sections** *[**nowait**]*

節:

private(リスト)、**firstprivate**(リスト)
lastprivate(*[lastprivate 修飾子:]* リスト)
reduction(*[reduction 修飾子,]* & *reduction 識別子:* リスト)
allocate (*[allocator:]* リスト)
C/C++ **nowait**

single [2.10.2] [2.8.3]

指定する構造化ブロックは、チーム内の 1 つのスレッドでのみ実行されます。

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

節:

private(リスト)、**firstprivate**(リスト)
allocate (*[allocator:]* リスト)
C/C++ **copyprivate**(リスト)、**nowait**

end 節:

For **copyprivate**(リスト)、**nowait**

workshare [2.10.3] [2.8.3]

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

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

ワークシェア・ループ構造

for / do [2.11.4] [2.9.2]

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] ...]* 構造化ブロック
For	**!$ omp do** *[節[[,]節] ...]* 構造化ブロック **!$ omp end do** *[**nowait**]*

節:

private(リスト) 、**firstprivate**(リスト)、
lastprivate(*[lastprivate 修飾子:]* リスト)、
linear(リスト *[: リニアステップ]*)、
schedule(*[修飾子 [,修飾子]:] kind[,チャンクサイズ]*)、
collapse(*n*)、ordered*[(n)]*、
allocate(*[allocator:]* リスト)、
order([order 修飾子] **concurrent)**、
reduction(*[reduction 修飾子,]* *reduction 識別子 :*リスト)
C/C++ **nowait**

***schedule** kind の値:*

- **static:** 反復は同一のチャンクサイズに分割され、そのチャンクはラウンドロビン方式 (総当り) でスレッド番号の順番にチーム内のスレッドへ振り分けられます。
- **dynamic:** スレッドは反復チャンクを実行して、実行が終わったら、次のチャンクを要求します。チャンクがなくなるまで、これを繰り返します。
- **guided:** スレッドは反復チャンクを実行して、実行が終わったら、次のチャンクを要求します。割り当てるチャンクがなくなるまで、これを繰り返します。チャンクサイズはチャンクごとに異なり、後になるほど小さくなります。
- **auto:** スケジュールの決定権は、コンパイラーやランタイムに任せられています (実装依存)。
- **runtime:** スケジュール・タイプとチャンクサイズは、内部制御変数 ***run-sched-var*** から取得されます。

***schedule** modifier の値:*

- **monotonic:** 各スレッドは、論理的な反復順序の昇順で割り当てられたチャンクを実行します。**schedule(static)** 節または **order** 節は、monotonic であることを意味します。
- **nonmonotonic:** チャンクはスレッドに任意の順番で割り当てられ、チャンクの実行順序に依存するアプリケーションの動作は未定義です。
- **simd:** ループが SIMD 構造に関連付けられていない場合は無視されますが、それ以外では最初と最後のチャンクを除き *new_chunk_size* は、[*chunk_size/simd_width*] * *simd_width* となります (*simd_width* は実装依存です)。

SIMD ディレクティブと構造

simd [2.11.5.1] [2.9.3.1]

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

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

節:

safelen(レングス)、simdlen(レングス)、
linear(リスト *[: リニアステップ]*) 、
aligned(リスト *[: アライメント]*) 、
nontemporal(リスト)、
private(リスト) 、
lastprivate(*[lastprivate 修飾子:]* リスト)、
reduction(*[reduction 修飾子,]* *reduction 識別子:*リスト)、
order([order 修飾子] **concurrent)**、
collapse(*n*)
C/C++ **if**(*[simd:]* スカラー式)
For **if**(*[simd:]* スカラー論理式)

Order 修飾子:

reproduce、**unconstrained**

for simd と do simd [2.11.5.2] [2.9.3.2]

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

C/C++	**#pragma omp for simd** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp do simd** *[節[[,]節] ...]* 入れ子のループ *[**!$ omp end do simd** [**nowait**]]*

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

declare simd [2.11.5.3] [2.9.3.3]

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

C/C++	**#pragma omp declare simd** *[節[[,]節] ...]* *[**#pragma omp declare simd** [節[[,]節] ...]]* 関数定義または宣言
For	**!$ omp declare simd** *[(proc-name)][節[[,]節] ...]*

節:

simdlen(レングス)、
linear(リスト *[: リニアステップ]*)、
aligned(引数リスト *[: アライメント]*)、
uniform(引数リスト)、
inbranch、
notinbranch

ディレクティブと構造 (続き)

distribute loop 構造

distribute [2.11.6.1] [2.9.4.1]
初期チームで実行されるループを指定します。

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

節:
private(リスト)、**firstprivate**(リスト)、
lastprivate(リスト)、**collapse**(*n*)、
dist_schedule(*kind [, チャンクサイズ]*)、
allocate(*[allocator :]* リスト)、
order ([order 修飾子 : **] concurrent)**
Ouder 修飾子:
reproduce、**unconstrained**

distribute simd [2.11.6.2] [2.9.4.2]
チーム領域のプライマリー・スレッドに分散され、SIMD 命令を使用して同時に実行されるループを指定します。

C/C++	**#pragma omp distribute simd** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp distribute simd** *[節[[,]節] ...]* 入れ子のループ ***[!$ omp end distribute simd]***

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

distribute parallel for と distribute parallel do [2.11.6.3] [2.9.4.3]
複数のチームのメンバーである複数のスレッドによって同時に実行できるループを指定します。

C/C++	**#pragma omp distribute parallel for ¥** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp distribute parallel do** *[節[[,]節] ...]* 入れ子のループ ***[!$ omp end distribute parallel do]***

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

distribute parallel for simd と distribute parallel do simd [2.11.6.4] [2.9.4.4]
複数のチームのメンバーである複数のスレッドで SIMD 命令を使用して同時に実行できるループを指定します。

C/C++	**#pragma omp distribute parallel for simd ¥** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp distribute parallel do simd** *[節[[,]節] ...]* 入れ子のループ ***[!$ omp end distribute parallel do simd]***

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

loop 構造

loop [2.11.7] [2.9.5]
複数のスレッドで関連するループ反復を並列に実行できることを示します。

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

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

スキャン・ディレクティブ

scan [2.11.8] [2.9.6]
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 節
	構造化ブロックシーケンス
	}

For	構造化ブロックシーケンス **!$omp scan** 節 構造化ブロックシーケンス

節:
inclusive(リスト)、**exclusive**(リスト)

loop 変換構造

tile [2.11.9.1]
1 つ以上の loop のタイル化します。

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

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

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

節:
full、**partial** *[(アンロール係数)]*

タスク構造

task [2.12.1] [2.10.1]
明示的にタスクを定義します。タスクのデータ環境は、**task** 構造のデータ共有属性節、データごとの環境 ICV、および適用されるデフォルトに従って作成されます。

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

節:
untied、**mergeable**、**private**(リスト)、
firstprivate(リスト)、**shared**(リスト)、
in_reduction(*reduction* 識別子: リスト)
depend(*[depend* 修飾子*,] dependence-type: locator*-リスト)
priority(プライオリティー値)
allocate(*[allocator:]* リスト)
affinity(*[aff* 修飾子*:] locator*-リスト)
- *aff* 修飾子は **iterator**(イテレーター定義)
detach(*event* ハンドル)
- *event* ハンドルは:
C/C++ **omp_event_handle** タイプ
For **kind omp_event_handle_kind**
default(データ共有属性)
C/C++ **if**([**task**:] スカラー式)
C/C++ **final**(スカラー式)
For **if**([**task**:] スカラー論理式)
For **final**(スカラー論理式)

taskloop [2.12.2] [2.10.2]
1 つ以上の関連するループ反復を、OpenMP タスクを使用して並列に実行します。

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

節:
shared(リスト)、**private**(リスト)、
firstprivate(リスト)、**lastprivate**(リスト)
reduction(*[default,][reduction* 識別子*:* リスト*)]*
in_reduction(*reduction* 識別子*:* リスト)
grainsize(*[strict:]* グレインサイズ)、
num_tasks(*[strict:]* タスク数)
collapse(*n*)、**priority**(プライオリティー値)
untied、**mergeable**、**nogroup**
allocate(*[allocator:]* リスト)
default(データ共有属性)
C/C++ **if**(*[***taskloop***:]* スカラー式)
C/C++ **final**(スカラー式)
For **if**(*[***taskloop***:]* スカラー論理式)
For **final**(スカラー論理式)

taskloop simd [2.12.3] [2.10.3]
ループが SIMD 命令を使用して同時に実行可能であり、またその反復は OpenMP タスクを使用して並列時実行できることを示します。

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

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

taskyield [2.12.4] [2.10.4]
現在のタスクを中断し、別のタスクの実行を優先することを許可します。

C/C++	**#pragma omp taskyield**
For	**!$ omp taskyield**

メモリー管理ディレクティブ

メモリー空間 [2.13.1] [2.11.1]
定義済みメモリー空間 **[表 2.8]** は、変数の格納と検索向けのストレージリソースを表します。

メモリー空間	ストレージ用途
omp_default_mem_space	システムのデフォルト
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 [2.13.3] [2.11.3]
変数の割り当て方法を指定します。

C/C++	**#pragma omp allocate** (リスト) *[節 [[,] 節] …]*
For	**!$ omp allocate** (リスト) *[節 [[,] 節] …]* - または - **!$ omp allocate** *[(リスト)] [節 [[,] 節] …]* ***[!$ omp allocate*** (リスト) *[節 [[,] 節] …] […]]*** 割り当て文

ディレクティブと構造 (続き)

節:
allocator (*allocator*)
- *allocator* は次の表現:
C/C++ **omp_allocator_handle_t タイプ**
For *kind* **omp_allocator_handle_kind**
align (アライメント)
- アライメントは 2 の整数乗
For 割り当て文: Fortran の ALLOCATE 文

デバイス・ディレクティブと構造

target data [2.14.2] [2.12.2]
領域範囲のデバイスデータ環境を作成します。

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

節:
map([[マップタイプ修飾子 [,] [マップタイプ修飾子 [,] …] マップタイプ:] *locator* リスト)、
use_device_ptr(リスト)、
use_device_addr(リスト)、
C/C++ **if**([**target data**:] スカラー式)
C/C++ **device**(整数式)
For **if**([**target data**:] スカラー論理式)
For **device**(スカラー整数式)

target enter data [2.14.3] [2.12.3]
変数をデバイスのデータ環境にマップします。

C/C++
```
#pragma omp target enter data [節[ [, ]節] …]
```
For
```
!$ omp target enter data [節[ [, ]節] …]
```

節:
map([[マップタイプ修飾子 [,] [マップタイプ修飾子 [,] …] マップタイプ:] *locator* リスト)、
depend([*depend* 修飾子,] 依存関係タイプ : *locator* リスト)、
nowait、
C/C++ **if**([**target enter data**:] スカラー式)
C/C++ **device**(整数式)
For **if**([**target enter data**:] スカラー論理式)
For **device**(スカラー整数式)

target exit data [2.14.4] [2.12.4]
変数をデバイスのデータ環境からアンマップします。

C/C++
```
#pragma omp target exit data [節[ [, ]節] …]
```
For
```
!$ omp target exit data [節[ [, ]節] …]
```

節:
map([[マップタイプ修飾子 [,] [マップタイプ修飾子 [,] …] マップタイプ:] *locator* リスト)、
depend([*depend* 修飾子,] 依存関係タイプ: *locator* リスト)、
nowait、
C/C++ **if**([**target exit data**:] スカラー式)
C/C++ **device**(整数式)
For **if**([**target exit data**:] スカラー論理式)
For **device**(スカラー整数式)

target [2.14.5] [2.12.5]
デバイスのデータ環境に変数をマップし、デバイス上で構造を実行します。

C/C++
```
#pragma omp target [節[ [, ]節] …]
    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:
private (リスト)、**firstprivate** (リスト)、
in_reduction (*reduction* 識別子 : リスト)、
map([[マップタイプ修飾子 [,] [マップタイプ修飾子 [,] …] マップタイプ:] *locator* リスト)、
is_device_ptr (リスト)、**has_device_addr** (リスト)、
defaultmap (*implicit-behavior*[:*variable-category*])、
nowait、
depend([*depend* 修飾子,] 依存関係タイプ : *locator* リスト)、
allocate ([*allocator* :] リスト)、
uses_allocators (*allocator*[(*allocator-traits-array*)] [,*allocator*[(*allocator-traits-array*)] …])
C/C++ **if** ([**target** :] スカラー式)
C/C++ **device**([*device* 修飾子 :] 整数式)
For **if** ([**target** :] スカラー論理式)
For **device** ([*device* 修飾子:] スカラー整数式)

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

target update [2.14.7] [2.12.7]
モーション節で指定される、元のリスト項目と一致するデバイスデータ環境のリスト項目を作成します。

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

節: モーション節または次のいずれか:
nowait
depend ([*depend* 修飾子,] 依存関係タイプ : *locator* リスト)
C/C++ **if** ([**target update** :] スカラー式)
C/C++ **device** (整数式)
For **if** ([**target update** :] スカラー論理式)
For **device** (スカラー整数式)

モーション節:
to ([モーション修飾子[,][モーション修飾子[,]…]:] *locator* リスト)
from ([モーション修飾子[,][モーション修飾子[,]…]:] *locator* リスト)
モーション修飾子: **present**、**mapper** (マッパー識別子)、**iterator** (*iterators* 定義)

declare target [2.14.7] [2.12.7]
デバイスにマップされる変数、関数およびサブルーチンを指定します。

C/C++
```
#pragma omp declare target
  宣言定義シーケンス
#pragma omp end declare target
 - または -
#pragma omp declare target (extended リスト)
 - または -
#pragma omp declare target 節[ [, ]節 …]
 - または -
#pragma omp begin declare target ¥
                        [節[ [, ]節] …]
  宣言定義シーケンス
#pragma omp end declare target
```
For
```
!$ omp declare target (extended リスト)
 - または -
!$ omp declare target [節[ [, ]節] …]
```

節:
to (*extended* リスト)、**link** (リスト)、
device_type (host | nohost | any)
indirect [(*invoked-by-fptr*)]

extended リスト: 名前付き変数、プロシージャー名、および名前付き共通ブロックをカンマで区切ったリスト。
invoked-by-fptr:
C/C++ 定数ブール式
For 論理式

相互運用構造

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

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

節:
アクション節、
device (整数式)、
depend ([*depend* 修飾子,] 依存関係タイプ : *locator* リスト)、

アクション節:
init ([*interop* 修飾子,] *interop* タイプ [[, *interop* タイプ] …] : *interop-var*)、
destroy (*interop-var*)、**use** (*interop-var*)、**nowait**
interop タイプ: **target**、**targetsync**
interop 修飾子: **prefer_type** (プリファレンス・リスト)

結合構造

parallel for と parallel do [2.16.1] [2.13.1]
1 つ以上の関連するループだけを含み、その他の文を含まないワークシェア・ループ構造を含む **parallel** 構造を指定します。

C/C++
```
#pragma omp parallel for [節[ [, ]節] …]
    入れ子のループ
```
For
```
!$ omp parallel do [節[ [, ]節] …]
    入れ子のループ
[!$ omp end parallel do]
```

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

parallel loop [2.16.2] [2.13.2]
1 つ以上の関連するループの **loop** 構造だけを含み、その他の文を含まない **parallel** 構造を簡潔に指定します。

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

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

parallel sections [2.16.3] [2.13.3]
1 つの **sections** 構造だけを含み、その他の文を含まない **parallel** 構造を簡潔に指定します。

C/C++
```
#pragma omp parallel sections [節[ [, ]節] …]
  {
  [#pragma omp section]
    構造化ブロック
  [#pragma omp section]
    構造化ブロック
  …
  }
```
For
```
!$ omp parallel sections [節[ [, ]節] …]
[#pragma omp section]
  構造化ブロック
[#pragma omp section]
  構造化ブロック
…
!$ omp end parallel sections
```

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

ディレクティブと構造 (続き)

parallel workshare [2.16.4] [2.13.4]

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

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

節: **parallel** ディレクティブと同一の節が適用されます。

parallel for simd と parallel do simd [2.16.5] [2.13.5]

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

C/C++	**#pragma omp parallel for simd** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp parallel do simd** *[節[[,]節] ...]* 入れ子のループ *[**!$ omp end parallel do simd**]*

節: **parallel** または **for/do simd** ディレクティブと同一の節が適用されます (C/C++ では nowait 節を除きます)。

parallel masked [2.16.6]

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

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

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

masked taskloop [2.16.7]

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

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

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

masked taskloop simd [2.16.8]

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

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

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

parallel masked taskloop [2.16.9]

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

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

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

parallel masked taskloop simd [2.16.10]

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

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

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

teams distribute [2.16.11] [2.13.11]

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

C/C++	**#pragma omp teams distribute** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp teams distribute** *[節[[,]節] ...]* 入れ子のループ *[**!$ omp end teams distribute**]*

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

teams distribute simd [2.16.12] [2.13.12]

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

C/C++	**#pragma omp teams distribute simd ¥** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp teams distribute simd** *[節[[,]節] ...]* 入れ子のループ *[**!$ omp end teams distribute simd**]*

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

teams distribute parallel for と teams distribute parallel do [2.16.13] [2.13.13]

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

C/C++	**#pragma omp teams distribute parallel for ¥** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp teams distribute parallel do** *[節[[,]節] ...]* 入れ子のループ *[**!$ omp end teams distribute parallel do**]*

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

teams distribute parallel for simd と teams distribute parallel do simd [2.16.14] [2.13.14]

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

C/C++	**#pragma omp teams distribute parallel for ¥** **simd** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp teams distribute parallel do simd** *[節[[,]節] ...]* 入れ子のループ *[**!$ omp end teams distribute parallel do simd**]*

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

teams loop [2.16.15] [2.13.15]

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

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

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

target parallel [2.16.16] [2.13.16]

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

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

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

target parallel for と target parallel do [2.16.17] [2.13.17]

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

C/C++	**#pragma omp target parallel for** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp target parallel do** *[節[[,]節] ...]* 入れ子のループ *[**!$ omp end target parallel do**]*

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

target parallel for simd と target parallel do simd [2.16.18] [2.13.18]

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

C/C++	**#pragma omp target parallel for simd** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp target parallel do simd** *[節[[,]節] ...]* 入れ子のループ *[**!$ omp end target parallel do simd**]*

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

target parallel loop [2.16.19] [2.13.19]

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

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

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

target simd [2.16.20] [2.13.20]

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

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

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

ディレクティブと構造 (続き)

target teams [2.16.21] [2.13.21]

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

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

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

target teams distribute [2.16.22] [2.13.22]

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

C/C++	**#pragma omp target teams distribute ¥** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp target teams distribute** *[節[[,]節] ...]* 入れ子のループ *[***!$ omp end target teams distribute***]*

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

target teams distribute simd [2.16.23] [2.13.23]

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

C/C++	**#pragma omp target teams distribute simd ¥** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp target teams distribute simd** *[節[[,]節] ...]* 入れ子のループ *[***!$ omp end target teams distribute simd***]*

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

target teams loop [2.16.24] [2.13.24]

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

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

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

target teams distribute parallel for と target teams distribute parallel do [2.16.25] [2.13.25]

teams distribute parallel for または teams distribute parallel do 構造を含み、その他の文を含まない **target** 構造を簡潔に指定します。

C/C++	**#pragma omp target teams distribute ¥** **parallel for** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp target teams distribute parallel do &** *[節[[,]節] ...]* 入れ子のループ *[***!$ omp end target teams distribute parallel do***]*

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

target teams distribute parallel for simd と target teams distribute parallel do simd [2.16.26] [2.13.26]

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

C/C++	**#pragma omp target teams distribute ¥** **parallel for simd** *[節[[,]節] ...]* 入れ子のループ
For	**!$ omp target teams distribute parallel &** **do simd** *[節[[,]節] ...]* 入れ子のループ *[***!$ omp end target teams distribute parallel &** **do simd***]*

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

同期構造

critical [2.19.1] [2.17.1]

指定する構造化ブロックの実行を一度に 1 つのスレッドに制限します。

C/C++	**#pragma omp critical *[(name)*** *[[,]* **¥** **hint** *(hint 式)]]* 構造化ブロック
For	**!$ omp critical** *[(name) [[,]* **hint** *(hint 式)]]* 緩い構造化ブロック **!$ omp end critical *[(name)]*** - または – **!$ omp critical** *[(name) [[,]* **hint** *(hint 式)]]* 厳密な構造化ブロック **!$ omp end critical *[(name)]***

hint 式: C/C++
有効な hint に評価される整数定数式
hint 式: For
kind **omp_sync_hint_kind** のスカラー値、および有効な同期 hint 値に評価される定数式

barrier [2.19.2] [2.17.2]

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

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

taskwait [2.19.5] [2.17.5]

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 (*[depend 修飾子,] dependence-type* : *locator リスト*) 、**nowait**

taskgroup [2.19.6] [2.17.6]

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:
task_reduction (*reduction 識別子 : リスト*)
allocate (*[allocator:]リスト*)

atomic [2.19.7] [2.17.7]

特定のストレージの位置がアトミックにアクセスされることを保証します。

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

節: *atomic* 節、メモリーオーダー節、または次のいずれか: **capture**、**compare**、**weak**、**hint** *(hint 式)*、**fail** *(seq_cst | acquire | relaxed)*
atomic 節: **read**、**write**、**update**
メモリーオーダー節:
seq_cst、**acq_rel**、**release**、**acquire**、**relaxed**

式文: C/C++

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	*cond-expr-stmt:* *x = expr ordop x* **?** *expr* **:** *x;* *x = x ordop expr* **?** *expr* **:** *x;* *x = x == e* **?** *d* **:** *x;* *cond-update-stmt:* ***if(****expr ordop x****) { x = expr; }*** ***if(****x ordop expr****) { x = expr; }*** ***if(****x == e****) { x = d; }***
capture	*v = expr-stmt* **{** *v = x;* *expr-stmt* **}** **{** *expr-stmt v = x;* **}** (where *expr-stmt* is either *write-expr-stmt, update-expr-stmt, or cond-expr-stmt*.)
compare と capture	*{ v = x; cond-update-stmt }* *{ cond-update-stmt v = 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; } }*

capture 文: *v = x*。For

式文: For

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*は、**MAX**、**MIN**、**IAND**、**IOR**、**IEOR**
operator は、**+**、*****、**-**、**/**、**.AND.**、**.OR.**、**.EQV.**、**.NEQV.**

ディレクティブと構造 (続き)

capture が存在して、文の前後に capture 文がある場合	許可されるものに加えて、*X = expr*

compare	***if (x == e) then***
	x = d
	end if
	if (x == e) x = d

compare と **capture** 節の両方が存在し、文の前後に capture 文がない場合	**if (*x* == *e*) then**
	x = d
	else
	v = x
	end if

flush [2.19.8] [2.17.8]
スレッドのテンポラリー・メモリー・ビューとメモリーの一貫性を保ち、変数のメモリー操作の順番を強制します。

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

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

ordered [2.19.9] [2.17.9]
並列ループ内でループの反復順に実行される構造化ブロックを指定したり、入れ子のループで反復間の依存関係を指定します。

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

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

depobj [2.19.10.1] [2.17.10.1]
OpenMP depend オブジェクトを初期化、更新、または破棄します。

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

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

キャンセル構造

cancel [2.20.1] [2.18.1]
指定したタイプの最内領域のキャンセル要求を行います。

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

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

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

cancellation point [2.20.2] [2.18.2]
タスクが、指定されたタイプの最内領域のキャンセルが要求されたかどうかをチェックするユーザー定義のキャンセルポイントを定義します。

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

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

データ環境ディレクティブ

threadprivate [2.21.2] [2.19.2]
各スレッドが独自のコピーを持つように、変数を複製します。**threadprivate** 変数のそれぞれのコピーは、最初に参照される前に一度だけ初期化されます。

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

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

declare reduction [2.21.5.7] [2.19.5.7]
reduction 節で使用できるリダクション演算子を宣言します。

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

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

declare mapper [2.21.7.4] [2.19.7.3]
与えられたタイプに対するユーザー定義マッパーを宣言し、**map** 節で使用するマッパー識別子を定義します。

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

マッパー識別子: ベース言語の識別子または **default**

type: スコープ内で有効なタイプ
var: 有効なベース言語の識別子

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

メモ

ランタイム・ライブラリー・ルーチン

スレッド・チーム・ルーチン

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

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

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

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

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

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

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

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

omp_in_parallel [3.2.5] [3.2.6]
内部制御変数 *active-levels-var* がゼロより大きい場合 *true* を返します。それ以外は *false* を返します。

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

omp_set_dynamic [3.2.6] [3.2.7]
スレッド数の動的調整が有効か無効かを示す内部制御変数 *dyn-var* の値を設定します。

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

omp_get_dynamic [3.2.7] [3.2.8]
このルーチンは内部制御変数 *dyn-var* の値を返します。スレッド数の動的調整が現在のタスクに対し有効であれば値は *true* です。

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

omp_get_cancellation [3.2.8] [3.2.9]
内部制御変数 *cancel-var* の値を返します。キャンセルが有効である場合は *true*、そうでなければ *false* を返します。

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

●● **omp_set_nested [3.2.9] [3.2.10]**
入れ子構造の並列処理を有効または無効にする、内部制御変数 *max-active-levels-var* を設定します。

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

●● **omp_get_nested [3.2.10] [3.2.11]**
内部制御変数 *max-active-levels-var* の値に応じて、入れ子構造の並列処理が有効または無効かを示す値を返します。

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

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

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

omp_get_schedule の *kind* を参照。

omp_get_schedule [3.2.12] [3.2.13]
runtime スケジュールに適用される内部制御変数 *run-sched-var* の値を返します。

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

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

omp_get_thread_limit [3.2.13] [3.2.14]
利用可能な OpenMP スレッドの最大数を示す、内部制御変数 *thread-limit-var* の値を返します。

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

omp_get_supported_active_levels [3.2.14] [3.2.15]
利用可能なアクティブな並列処理レベル数を返します。

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

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

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

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

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

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

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

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

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

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

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

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

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

スレッド・アフィニティー・ルーチン

omp_get_proc_bind [3.33.1] [3.2.23]
proc_bind 節を指定しない後続の入れ子になった **parallel** 領域に適用するスレッド・アフィニティー・ポリシーを返します。

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

次のいずれかを返します:
omp_proc_bind_false
omp_proc_bind_true
omp_proc_bind_primary (5.0 では primary は master)
omp_proc_bind_close
omp_proc_bind_spread

omp_get_num_places [3.3.2] [3.2.24]
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** ()

ランタイム・ライブラリー・ルーチン (続き)

omp_get_place_num_procs [3.3.3] [3.2.25]
指定された位置の実行環境で使用可能なプロセッサー数を返します。

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

omp_get_place_proc_ids [3.3.4] [3.2.26]
指定された位置の実行環境で使用可能なプロセッサーの数値識別子を返します。

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

omp_get_place_num [3.3.5] [3.2.27]
遭遇したスレッドがバインドされている位置のプレース番号を返します。

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

omp_get_partition_num_places [3.3.6] [3.2.28]
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 [3.3.7] [3.2.29]
最も内側の暗黙のタスクの内部制御変数 *place-partition-var* 内の位置に対応するプレース番号のリスト返します。

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

omp_set_affinity_format [3.3.8] [3.2.30]
デバイスで使用されるアフィニティー形式を設定する内部制御変数 *affinity-format-var* の値を設定します。

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

omp_get_affinity_format [3.3.9] [3.2.31]
デバイスの内部制御変数 *affinity-format-var* の値を返します。

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

omp_display_affinity [3.3.10] [3.2.32]
提供された書式で OpenMP スレッド・アフィニティー情報を表示します。

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

omp_capture_affinity [3.3.11] [3.2.33]
提供された書式で、OpenMP スレッド・アフィニティー情報をバッファーに出力します。

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

チーム領域ルーチン

omp_get_num_teams [3.4.1] [3.2.38]
現在の **teams** 領域のチーム数を返します。**teams** 領域外から呼び出された場合 1 を返します。

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

omp_get_team_num [3.4.2] [3.2.39]
呼び出しスレッドのチーム番号を返します。チーム番号は、0 から **omp_get_num_teams** で返される値より 1 つ少ない整数値です。

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

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

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

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

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

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

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

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

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

タスクルーチン

omp_get_max_task_priority [3.5.1] [3.2.42]
priority 節で指定できる最大値を返します。

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

omp_in_final [3.5.2] [3.2.22]
final タスク領域内で実行された場合、*true* を返します。そうでない場合は、*false* を返します。

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

リソース解放ルーチン

omp_pause_resource [3.6.1] [3.2.43]
omp_pause_resource_all [3.6.2] [3.2.44]
ランタイムが、指定されたデバイス上の OpenMP によって使用されるリソースを解放することを許可します。有効な kind 値には、omp_pause_soft と omp_pause_hard が含まれます。

C/C++	**int omp_pause_resource** (**omp_pause_resource_t** *kind*, **int** *device_num*); **int omp_pause_resource_all** (**omp_pause_resource_t** *kind*);
For	**integer function omp_pause_resource** (*kind device_num*) **integer** (**kind=omp_pause_resource_kind**) *kind* **integer** ***device_num*** **integer function omp_pause_resource_all** (*kind*) **integer** (**kind=omp_pause_resource_kind**) *kind*

デバイス情報ルーチン

omp_get_num_procs [3.7.1] [3.2.5]
ルーチンが呼び出されたときに、デバイスで利用可能なプロセッサー数を返します。

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

omp_set_default_device [3.7.2] [3.2.34]
デフォルトのターゲットデバイスを決定する内部制御変数 *default-device-var* を設定します。

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

omp_get_default_device [3.7.3] [3.2.35]
デフォルトのターゲットデバイスを示す、内部制御変数 *default-device-var* 値を返します。

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

ランタイム・ライブラリー・ルーチン (続き)

omp_get_num_devices [3.7.4] [3.2.36]
ターゲットデバイス数を返します。

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

omp_get_device_num [3.7.5] [3.2.37]
呼び出しスレッドが実行しているデバイスのデバイス番号を返します。

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

omp_is_initial_device [3.7.6] [3.2.40]
現在のタスクがホストデバイス上で実行している場合 *true* を返します。それ以外は、*false* を返します。

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

omp_get_initial_device [3.7.7] [3.2.41]
ホストデバイスのデバイス番号を返します。

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

デバイス・メモリー・ルーチン

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

omp_target_alloc [3.8.1] [3.6.1]
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 [3.8.2] [3.6.2]
omp_target_alloc ルーチンで割り当てられたデバイスメモリーを開放します。

C/C++
```
void omp_target_free (
        void *device_ptr, int device_num);
```
For
```
subroutine omp_target_free ( &
  device_ptr, device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
  c_int
type (c_ptr), value :: device_ptr
integer (c_int), value :: device_num
```

omp_target_is_present [3.8.3] [3.6.3]
ホストポインターが、指定されたデバイス上のデバイスバッファーに関連付けられているか確認します。

C/C++
```
int omp_target_is_present (
        const void *ptr, int device_num);
```
For
```
integer(c_int) function omp_target_is_present &
  (ptr, device_num) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_int
type(c_ptr), value :: ptr
integer(c_int), value :: device_num
```

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

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

omp_target_memcpy [3.8.5] [3.6.4]
ホストとデバイスポインターの任意の組み合わせでメモリーをコピーします。

C/C++
```
int omp_target_memcpy (void *dst,
    const void *src, size_t length,
    size_t dst_offset, size_t src_offset,
    int dst_device_num, int src_device_num);
```
For
```
integer(c_int) function omp_target_memcpy(&
  dst, src, length, dst_offset, src_offset, &
  dst_device_num, src_device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
  c_int, c_size_t
type(c_ptr), value :: dst, src
integer(c_size_t), value :: length, dst_offset, &
  src_offset
integer(c_int), value :: dst_device_num, &
  src_device_num
```

omp_target_memcpy_rect [3.8.6] [3.6.5]
多次元配列からほかの多次元配列へ矩形サブボリュームをコピーします。

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

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

C/C++
```
int omp_target_memcpy_async (void *dst,
  const void *src, size_t length, size_t dst_offset,
  size_t src_offset, int dst_device_num,
  int src_device_num, int depobj_count,
  omp_depend_t *depobj_list);
```
For
```
integer(c_int) function &
omp_target_memcpy_async( &
  dst, src, length, dst_offset, src_offset, &
  dst_device_num, src_device_num, &
  depobj_count, depobj_list) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
  c_int, c_size_t
type(c_ptr), value :: dst, src
integer(c_size_t), value :: length, dst_offset, &
  src_offset
integer(c_int), value :: dst_device_num, &
  src_device_num, depobj_count
integer(omp_depend_kind), &
  optional :: depobj_list(*)
```

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

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

omp_target_associate_ptr [3.8.9] [3.6.6]
omp_target_alloc もしくは実装定義のランタイムルーチンから返されるデバイスポインターをホストポインターにマップします。

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

omp_target_disassociate_ptr [3.8.10] [3.6.7]
ホストポインターから、指定されたデバイスに関連するポインターを削除します。

C/C++
```
int omp_target_disassociate_ptr (
    const void *ptr, int device_num);
```
For
```
integer(c_int) function &
omp_target_disassociate_ptr(&
  ptr, device_num) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_int
type(c_ptr), value :: ptr
integer(c_int), value :: device_num
```

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

C/C++
```
void *omp_get_mapped_ptr (const void *ptr,
int device_num);
```
For
```
type(c_ptr) function omp_get_mapped_ptr( &
  ptr, device_num) bind(c)
use, intrinsic :: iso_c_binding, &
  only : c_ptr, c_int
type(c_ptr), value :: ptr
integer(c_int), value :: device_num
```

ロックルーチン

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

ロックの初期化 [3.9.1] [3.3.1]

OpenMP ロックを初期化します。

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

void omp_init_nest_lock (omp_nest_lock_t *lock);
```

For
```
subroutine omp_init_lock (svar)
integer (kind=omp_lock_kind) svar

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

ヒント付きのロックの初期化 [3.9.2] [3.3.2]

ヒント付きで OpenMP ロックを初期化します。

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

void omp_init_nest_lock_with_hint (
    omp_nest_lock_t *lock,
    omp_sync_hint_t hint);
```

For
```
subroutine omp_init_lock_with_hint (svar, hint)
integer (kind=omp_lock_kind) svar
integer (kind=omp_sync_hint_kind) hint

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

hint: [2.17.12] [2.18.12] を参照

ロックの破棄 [3.9.3] [3.3.3]

OpenMP ロックを破棄します。ロックを非初期化状態に戻します。

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

void omp_destroy_nest_lock (omp_nest_lock_t *lock);
```

For
```
subroutine omp_destroy_lock (svar)
integer (kind=omp_lock_kind) svar

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

ロックの設定 [3.9.4] [3.3.4]

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

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

void omp_set_nest_lock (omp_nest_lock_t *lock);
```

For
```
subroutine omp_set_lock (svar)
integer (kind=omp_lock_kind) svar

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

ロックの解除 [3.9.5] [3.3.5]

OpenMP ロックを解除します。

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

void omp_unset_nest_lock (omp_nest_lock_t *lock);
```

For
```
subroutine omp_unset_lock (svar)
integer (kind=omp_lock_kind) svar

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

ロックをテスト [3.9.6] [3.3.6]

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

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

void omp_test_nest_lock (omp_nest_lock_t *lock);
```

For
```
subroutine omp_test_lock (svar)
integer (kind=omp_lock_kind) svar

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

タイミングルーチン

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

omp_get_wtime [3.10.1] [3.4.1]

ウォールクロックの経過時間を秒単位で返します。

C/C++
```
double omp_get_wtime (void);
```

For
```
double precision function omp_get_wtime ()
```

omp_get_wtick [3.10.2] [3.4.2]

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

C/C++
```
double omp_get_wtick (void);
```

For
```
double precision function omp_get_wtick ()
```

イベントルーチン

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

omp_fulfill_event [3.11.1] [3.5.1]

OpenMP イベントを作成または破棄します。

C/C++
```
void omp_fulfill_event (
        omp_event_handle_t event);
```

For
```
subroutine omp_fulfill_event (event)
integer (kind=omp_event_handle_kind) event
```

相互運用ルーチン

omp_get_num_interop_properties [3.12.1]

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

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

omp_get_interop_int [3.12.2]

omp_interop_t オブジェクトから整数プロパティーを取得します。

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

omp_get_interop_ptr [3.12.3]

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

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

omp_get_interop_str [3.12.4]

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

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

omp_get_interop_name [3.12.5]

omp_interop_t オブジェクトからプロパティー名を取得します。

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

omp_get_interop_type_desc [3.12.6]

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

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

omp_get_interop_rc_desc [3.12.7]

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

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

メモリー管理ルーチン

メモリー管理タイプ [3.13.1] [3.7.1]

C/C++ の **omp_alloctrait_t** 構造体と Fortran の **omp_alloctrait** 型は、次の型と値を持つメンバー名 *key* と *value* を定義します:

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

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

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

omp_atv_X。ここで X は false、true、default、contended、uncontended、serialized、sequential、private、all、thread、pteam、cgroup、default_mem_fb、null_fb、abort_fb、allocator_fb、environment、nearest、blocked、interleaved のいずれかです [5.1 では sequential は非推奨となりました]。

omp_init_allocator [3.13.2] [3.7.2]

アロケーターを初期化してメモリー空間に関連付けます。

C/C++
```
omp_allocator_handle_t omp_init_allocator (
    omp_memspace_handle_t memspace,
    int ntraits, const omp_alloctrait_t traits[]);
```

For
```
integer (kind=omp_allocator_handle_kind) &
    function omp_init_allocator (&
    memspace, ntraits, traits)
integer (kind=omp_memspace_handle_kind),&
    intent (in) :: memspace
integer, intent (in) :: ntraits
type (omp_alloctrait), intent (in) :: traits (*)
```

omp_destroy_allocator [3.13.3] [3.7.3]

アロケーター・ハンドルによって使用されたすべてのリソースを解放します。

C/C++
```
void omp_destroy_allocator (
    omp_allocator_handle_t allocator);
```

For
```
subroutine omp_destroy_allocator (allocator)
integer (kind=omp_allocator_handle_kind), &
    intent (in) :: allocator
```

ランタイム・ライブラリー・ルーチン (続き)

omp_set_default_allocator [3.13.4] [3.7.4]

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

C/C++
```
void omp_set_default_allocator (
    omp_allocator_handle_t allocator);
```

For
```
subroutine omp_set_default_allocator ( &
    allocator)
integer (kind=omp_allocator_handle_kind), &
    intent (in) :: allocator
```

omp_get_default_allocator [3.13.5] [3.7.5]

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

C/C++
```
omp_allocator_handle_t
  omp_get_default_allocator (void);
```

For
```
integer (kind=omp_allocator_handle_kind)
function omp_get_default_allocator ()
```

omp_alloc と omp_aligned_alloc [3.13.6] [3.7.6]

メモリー・アロケーターにメモリー割り当てを要求します。

C
```
void *omp_alloc (size_t size,
    omp_allocator_handle_t allocator);

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

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

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

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

omp_free [3.13.7] [3.7.7]

割り当てたメモリーを解放します。

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

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

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

omp_calloc と omp_aligned_calloc [3.13.8]

メモリー・アロケーターにゼロで初期化されたメモリー割り当てを要求します。

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

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

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

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

omp_realloc [3.13.9]

割り当て済みのメモリー割り当てを解放し、メモリー・アロケーターにメモリー割り当てを要求します。

C
```
void *omp_realloc (void *ptr, size_t size,
  omp_allocator_handle_t allocator,
  omp_allocator_handle_t free_allocator);
```

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

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

ツール制御ルーチン

omp_control_tool [3.14] [3.8]

プログラムがアクティブなツールにコマンドを渡すことを可能にします。

C/C++
```
int omp_control_tool (int command,
    int modifier, void *arg);
```

For
```
integer function omp_control_tool ( &
  command, modifier )
integer (kind=omp_control_tool_kind) &
  command
integer modifier
```

command:

omp_control_tool_start: 監視がオフの場合は監視を開始または再開します。監視がオンの場合は保持されます。監視が永続的にオフの場合は効果がありません。

omp_control_tool_pause: 一時的に監視をオフにします。オフの場合は保持されます。

omp_control_tool_flush: バッファーのデータをフラッシュします。監視がオン/オフでも適用されます。

omp_control_tool_end: 監視を永続的にオフにします。ツールは自動的に終了し、すべての出力をフラッシュします。

環境表示ルーチン

omp_display_env [3.15]

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

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

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

節

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

allocate 節 [2.13.4] [2.11.4]

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

allocator 修飾子*:*
allocator (allocator): allocator は次のいずれか。
C/C++ **omp_allocator_handle_t** タイプ
For **omp_allocator_handle_kind** *kind*

align (アライメント): アライメントは 2 の正の整数乗の定数。

データコピー節 [2.21.6] [2.19.6]

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

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

データ map 節 [2.21.7] [2.19.7]

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

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

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

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

節 (続き)

defaultmap (*implicit-behavior [: variable-category]*)
target 構造で参照されるデータマッピング属性を明示的に決定します。それ以外は、暗黙的に決定されます。

implicit-behavior: **alloc**、**to**、**from**、**tofrom**、**firstprivate**、**none**、**default**、**present**

C++ *variable-category:*
scalar, aggregate, pointer
For *variable-category:*
scalar, aggregate, pointer, allocatable

データ共有節 [2.21.4] [2.19.4]

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

default (**private | firstprivate | shared | none**)
parallel、**teams**、およびタスク生成構造内で参照される変数のデフォルトのデータ共有属性を明示的に決定し、構造内のデータ共有属性が暗黙的に決定されるすべての変数参照に適用します [C/C++ の firstprivate と private は 5.1 で利用可能になりました]。

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

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

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

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

lastprivate 修飾子: **conditional**
conditional は、入れ子のループのシーケンシャル実行後に、リスト項目の値がリスト項目に割り当てられることを指定します。

linear (*linear* リスト *[: リニアステップ]*)
プライベートであり、節が指定された構造に関連するループの反復空間とリニアな関係を持つ 1 つ以上のリスト項目を宣言します。

linear リスト*:* リスト もしくは修飾子*(*リスト*)*
修飾子*:* **ref**、**val**、または **uval** (C では **val** のみ)

depend 節 [2.19.11] [2.17.11]

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

depend (依存関係タイプ)
依存関係タイプは、**source** です。

depend (依存関係タイプ : *vec*)
依存関係タイプは、*sink* で *vec* 反復は次の形式です: x_1 [± d_1]、x_2 [± d_2]、. . . 、x_n [± d_n]

depend(*[depend 修飾子,]* 依存関係タイプ **:** *locator* リスト)
depend 修飾子*:* **iterator** (iterators 定義)
依存関係タイプ*:* **in**、**out**、**inout**、**mutexinoutset**、**inoutset**、**depobj**

- **in**: 生成されるタスクは、**out** または **inout** *dependence-type* リストの 1 つ以上のリスト項目を参照する、以前に生成されたすべての兄弟タスクに依存します。
- **out** と **inout**: 生成されるタスクは、**in**、**out**、**mutexinoutset**、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。
- **mutexinoutset:** リスト項目の少なくとも 1 つの格納場所が、兄弟タスクが生成された構造の **in**、**out**、**inout** *dependence-type* を持つ **depend** 節のリスト項目の格納場所と同じである場合、生成されるタスクは兄弟タスクの従属タスクになります。リスト項目の少なくとも 1 つの格納場所が、兄弟タスクが生成された構造の **mutexinoutset** 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。

if 節 [2.18] [2.15]

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

if (*[*ディレクティブ名修飾子*:]* スカラー式) C/C++
if (*[*ディレクティブ名修飾子 *:]* スカラー論理式) For

order と ordered 節 [2.11.3] [2.9.2]

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。

ordered *[(n)]*
構造に関連付けるループまたはループ数を示します。

reduction 節 [2.21.5] [2.19.5]

in_reduction (*reduction* 識別子: リスト)
タスクがリダクションに参加することを指定します。reduction 識別子: **reduction** と同じです。

task_reduction (*reduction* 識別子 **:** リスト)
タスク間のリダクションを指定します。
reduction 識別子: **reduction** と同じです。

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

reduction 修飾子: **inscan**、**task**、**default**

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

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

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

SIMD 節 [2.11.5] [2.9.3]

データ共有節と if 節を参照してください。

aligned(引数リスト *[: アライメント]*)
指定するバイト数にアライメントする 1 つ以上のリスト項目を宣言します。

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

inbranch
関数が常に SIMD ループの条件文から呼び出されることを指定します (**simd** ではなく **declare simd** で使用します) 。

nontemporal (リスト)
リスト項目が参照するストレージの場所へのアクセスが、ストレージにアクセスする反復全体で時間的な局所性が低いことを指定します。

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

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

simdlen(レングス)
正の定数整数式で関数の同時引数の数を指定します。

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

task 節 [2.12] [2.10]

affinity (*[aff* 修飾子*:] locator* リスト)
リスト項目の位置に隣接して実行するヒントを指定します。*aff* 修飾子は **iterator** (iterators-definition) です (**taskloop** では使用できません)。

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

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

default (private | **firstprivate** | **shared** | **none)**
このページのデータ共有節を参照してください (**taskloop** では使用できません)。

depend (*[depend* 修飾子*,]* 依存関係タイプ*: locator* リスト)
このページの **depend** 節を参照してください (**taskloop** では使用できません)。

detach (リスト)
タスクが完了してもシステム内に存在するため、そのタスクの完了を待機するほかのタスクは解放されません (omp_fulfilled_event も参照)。

final (スカラー式) C/C++
final (スカラー論理式) For
final 式が true と評価される場合、生成されるタスクは final タスクとなります。

firstprivate (リスト)
このページのデータ共有節を参照してください。

grainsize (*[**strict** :]* グレインサイズ)
生成される各タスクに割り当てられる論理ループの反復数は、グレインサイズ式の値と論理ループの反復回数の最小値以上、グレインサイズ式の値の 2 倍未満にします。最後の反復を除き、厳密には実際のグレインサイズを強制します (**task** では使用できません)。

if (*[* **task :** *]* スカラー式) C/C++
if (*[* **task :** *]* スカラー論理式) For
このページの if 節を参照してください。

節 (続き)

in_reduction (*reduction* 識別子: リスト)
13 ページの **reduction** 節を参照してください。

lastprivate (リスト)
13 ページのデータ共有節を参照してください。

mergeable
生成されるタスクがマージ可能であることを指定します。

nogroup
暗黙の **taskgroup** 領域を作成しないようにします (**task** では使用できません)。

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

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

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

reduction (*[reduction 修飾子 ,] reduction* 識別子: リスト)
13 ページの reduction 節を参照してください (**task** では使用できません)。

shared (リスト)
13 ページのデータ共有節を参照してください。

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

イテレーター

iterators [2.1.6] [2.1.6]
節の中で複数の値に展開される識別子。

iterator (*iterators* 定義)

iterators 定義:
iterator 指定子 *[, iterators* 定義 *]*

iterators 指定子:
[iterator タイプ *] identifier* = *range* 指定子

identifier: ベース言語の識別子。

range 指定子: *begin* : *end[*: *step]*
begin, *end*: 型を *iterator* タイプに変換できる式。

step: 整数式
iterator タイプ: タイプ名 C/C++
iterator タイプ: タイプ指定子 For

内部制御変数 (ICV) 値

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

ICV 初期値の表 (表 2.2) および ICV 値の変更と取得方法 (表 2.3)

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

内部制御変数 (ICV) 値 (続き)

ICV	環境変数	初期値	値の変更	値の取得	スコープ	参照
display-affinity-var	**OMP_DISPLAY_AFFINITY**	false	(なし)	(なし)	グローバル	[6.13] [6.13]
affinity-format-var	**OMP_AFFINITY_FORMAT**	実装定義	**omp_set_affinity_format()**	**omp_get_affinity_format()**	デバイス	[6.14] [6.14]
default-device-var	**OMP_DEFAULT_DEVICE**	実装定義	**omp_set_default_device()**	**omp_get_default_device()**	データ環境	[6.15] [6.15]
target-offload-var	**OMP_TARGET_OFFLOAD**	デフォルト	(なし)	(なし)	グローバル	[6.17] [6.17]
max-task-priority-var	**OMP_MAX_TASK_PRIORITY**	ゼロ	(なし)	**omp_get_max_task_priority()**	グローバル	[6.16] [6.16]
tool-var	**OMP_TOOL**	有効	(なし)	(なし)	グローバル	[6.18] [6.18]
tool-libraries-var	**OMP_TOOL_LIBRARIES**	空の文字列	(なし)	(なし)	グローバル	[6.19] [6.19]
tool-verbose-init-var	**OMP_TOOL_VERBOSE_INIT**	無効	(なし)	(なし)	グローバル	[6.20]
debug-var	**OMP_DEBUG**	無効	(なし)	(なし)	グローバル	[6.21] [6.20]
num-procs-var	(なし)	実装定義	(なし)	**omp_get_num_procs()**	デバイス	-
thread-num-var	(なし)	ゼロ	(なし)	**omp_get_thread_num()**	実装タスク	-
final-task-var	(なし)	False	(なし)	**omp_in_final()**	データ環境	-
implicit-task-var	(なし)	True			データ環境	-
team-size-var	(なし)	1	(なし)	**omp_get_num_threads()**	チーム	-
def-allocator-var	**OMP_ALLOCATOR**	実装定義	**omp_set_default_allocator()**	**omp_get_default_allocator()**	実装タスク	[6.22] [6.21]
nteams-var	**OMP_NUM_TEAMS**	ゼロ	**omp_set_num_teams()**	**omp_get_max_teams()**	デバイス	[6.23]
teams-thread-limit-var	**OMP_TEAMS_THREAD_LIMIT**	ゼロ	**omp_set_teams_thread_limit()**	**omp_get_teams_thread_limit()**	デバイス	[6.24]

メモ

環境変数

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

OMP_ALLOCATOR [6.22] [5.21]

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 2-8 に示します。
- アロケーターの特性を表 2-9 に示します。
- 事前定義されたアロケーターを表 2-10 に示します。

例:

```
setenv OMP_ALLOCATOR omp_high_bw_mem_alloc

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

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

表 2-8 メモリー空間名

omp_default_mem_space
omp_large_cap_mem_space
omp_const_mem_space
omp_high_bw_mem_space
omp_low_lat_mem_space

表 2–9 アロケーター特性と許容値 (デフォルト)

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

表 2–10 事前定義アロケーター (メモリー空間と特性値)

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

OMP_AFFINITY_FORMAT *format* **[6.14] [5.14]**

OpenMP スレッド・アフィニティー情報を表示する際の形式を定義する内部制御変数 *affinity-format-var* の初期値を設定します。引数は文字列で、他の文字に加えて、サブ文字列として 1 つ以上のフィールド指定子を含むことができます。各フィールド指定子の形式は、%[[[0].*size*] *type* で、フィールドタイプは次にリストされる短縮文字または名前です **[表 6.3] [表 5.14]**。

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

OMP_CANCELLATION [6.11] [5.11]

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

OMP_DEBUG [6.21] [5.20]

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

OMP_DEFAULT_DEVICE *device* **[6.15] [5.15]**

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

OMP_DISPLAY_AFFINITY *var* **[6.13] [5.13]**

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

OMP_DISPLAY_ENV *var* **[6.12] [5.12]**

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

OMP_DYNAMIC *var* **[6.3] [5.3]**

内部制御変数 *dyn-var* を設定します。*var* は **true** または **false** です。**true** の場合、**parallel** 領域を実行するスレッド数を動的に調整することを許可します。

OMP_MAX_ACTIVE_LEVELS *levels* **[6.8] [5.8]**

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

OMP_MAX_TASK_PRIORITY *level* **[6.16] [5.16]**

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

● ● **OMP_NESTED** *nested* **[6.9] [5.9]**

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

OMP_NUM_TEAMS [6.23]

内部制御変数 nteams-var を設定することで、**teams** 構造で生成されるチームの最大数を設定します。

OMP_NUM_THREADS リスト **[6.2] [5.2]**

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

OMP_PLACES *places* **[6.5] [5.5]**

実行環境で利用可能な OpenMP プレースを定義する内部制御変数 *place-partition-var* を設定します。*places* は、抽象的な名称 (**threads**、**cores**、**sockets**、**ll_caches**、**numa_domains**)、または中括弧で区切られた数値の各場所が、デバイス上の順序付けされていないプロセッサーのセットである順序付きリストです。

OMP_PROC_BIND *policy* **[6.4] [5.4]**

対応する入れ子レベルの **parallel** 領域で使用するスレッド・アフィニティー・ポリシーを定義するグローバル内部制御変数 *bind-var* を設定します。*policy* には、**true**、**false**、または引用符で囲ったカンマ区切りの **primary**、**close**、もしくは **spread** リストを指定できます [5.1 では、**master** は **primary** に置き換えられました]。

OMP_SCHEDULE *[modifier:] kind [,chunk]* **[6.1] [5.1]**

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

OMP_STACKSIZE *size[***B** | **K** | **M** | **G***]* **[6.6] [5.6]**

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

OMP_TARGET_OFFLOAD *arg* **[6.17] [5.17]**

target-offload-var の初期値を設定します。引数は、**mandatory**、**disabled**、または **default** のいずれかす。

OMP_TEAMS_THREAD_LIMIT [6.24]

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

OMP_THREAD_LIMIT *limit* **[6.10] [5.10]**

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

OMP_TOOL (**enabled** | **disabled**) **[6.18] [5.18]**

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

OMP_TOOL_LIBRARY *library* リスト **[6.19] [5.19]**

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

OMP_TOOL_VERBOSE_INIT [6.20]

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

OMP_WAIT_POLICY *policy* **[6.7] [5.7]**

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

ツールのアクティベーション

OMPT ツールをアクティブにする [4.2] [4.2]
OpenMP 実装がツールをアクティブにするには 3 つのステップがあります。ここでは、そのためにツールと OpenMP 実装がどのように作用するか説明します。
OMPT インターフェイスには、ターゲットデバイスのアクティビティーをトレースする監視インターフェイスも含まれています (4.2.5 節参照)。

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

ツールが OpenMP 実装に **ompt_start_tool** 定義を提供するには、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_LIBRARY** 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。

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

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

ompt_set_error
ompt_set_never
ompt_set_impossible
ompt_set_sometimes
ompt_set_sometimes_paired
ompt_set_always

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

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

メモ