OpenMP 4.0 API C/C++ シンタックス・クイック・リファレンス・カード

C/C++ 日本語翻訳版

OpenMP アプリケーション・プログラム・インターフェイス (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 は複数の共有メモリー型プラットフォームをサポートします。Unix プラットフォームと Windows プラットフォームを含む、すべてのアーキテクチャー上での C/C++ と Fortran での並列プログラミング。仕様に関しては www.openmp.org をご覧ください。

4.0 はバージョン 4.0 の新機能を指します。

[n.n.n] は、OpenMP API 仕様書のバージョン 4.0 のセクションを示し、 **[n.n.n]** は 3.1 を指します。

宣言子

OpenMP の実行指示句は、後続の構造化ブロックや OpenMP 構文に適用されます。各宣言子は **#pragma omp** で始まります。宣言子の残りの部分は、C と C++ コンパイラー宣言子の標準規約に準拠します。構造化ブロックは、単一のステートメントか出入り口が各 1 つの複合文です。

parallel [2.5] [2.4]

スレッドのチームを形成し、並列実行を開始します。

#pragma omp parallel *[節[[,]節] ...]*
　構造化ブロック

節:
- **if**(スカラー式)
- **num_threads**(整数式)
- **default**(**shared** | **none**)
- **private**(リスト)
- **firstprivate**(リスト)
- **shared**(リスト)
- **copyin**(リスト)
- **reduction**(リダクション演算子**:**リスト)
- **4.0** **proc_bind**(**master** | **close** | **spread**)

loop [2.7.1] [2.5.1]

指定するループ反復は、暗黙的なタスクのコンテキストでスレッドのチームによって並列に実行されます。

#pragma omp *for* *[節[[,]節] ...]*
　for ループ

節:
- **private**(リスト)
- **firstprivate**(リスト)
- **lastprivate**(リスト)
- **reduction**(リダクション演算子**:**リスト)
- **schedule**(タイプ*[*,チャンクサイズ*]*)
- **collapse**(*n*)
- **ordered**
- **nowait**

タイプ:
- **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。
- **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、**環境変数**から取得されます。

sections [2.7.2] [2.5.2]

一連の構造化ブロックを含む非反復型のワークシェアリング構文は、スレッドのチームに分散され実行されます。

#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*]*
　}

節:
- **private**(リスト)
- **firstprivate**(リスト)
- **astprivate**(リスト)
- **reduction**(リダクション演算子**:**リスト)
- **nowait**

single [2.7.3] [2.5.3]

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

#pragma omp single *[節[[,]節] ...]*
　構造化ブロック

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

4.0 simd [2.8.1]

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

#pragma omp simd *[節[[,]節] ...]*
　for ループ

節:
- **safelen**(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*]*)
- **private**(リスト)
- **lastprivate**(リスト)
- **reduction**(リダクション演算子**:**リスト)
- **collapse**(*n*)

4.0 declare simd [2.8.2]

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

#pragma omp declare simd *[節[[,]節] ...]*
　*[***#pragma omp declare simd** *[節[[,]節] ...]]*
　[...]
　　関数定義または宣言

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

4.0 loop simd [2.8.3]

ループが SIMD 命令を使用して命令レベルの並列性を持つと同時に、それらの反復がスレッドのチームで並列に実行できることを指示。

#pragma omp for simd *[節[[,]節] ...]*
　for ループ

節:
SIMD もしくは **for** 宣言子と同一の節と制限が適用されます。

4.0 target [data] [2.9.1, 2.9.2]

これらの構文は、領域範囲のデバイスデータ環境を作成します。ターゲットはデバイス上で実行を開始します。

#pragma omp target data *[節[[,]節] ...]*
　構造化ブロック

#pragma omp target *[節[[,]節] ...]*
　構造化ブロック

節:
- **device**(整数式)
- **map**(*[*マップタイプ**:***]* リスト)
- **if**(スカラー式)

4.0 target update [2.9.3]

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。

#pragma omp target update *[節[[,]節] ...]*

節は、モーション節もしくは次のうち 1 つ:
- **device**(整数式)
- **if**(スカラー式)

モーション節:
- **to**(リスト)
- **from**(リスト)

4.0 declare target [2.9.4]

デバイスにマップされる変数と関数を指定する宣言子。

#pragma omp declare target
宣言定義シーケンス
#pragma omp end declare target

4.0 teams [2.9.5]

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。

#pragma omp teams *[節[[,]節] ...]*
　構造化ブロック

節:
- **num_teams**(整数式)
- **thread_limit**(整数式)
- **default**(**shared** | **none**)
- **private**(リスト)
- **firstprivate**(リスト)
- **shared**(リスト)
- **reduction**(リダクション演算子**:**リスト)

4.0 distribute [simd] [2.9.6, 2.9.7]

1 つ以上のループの反復をすべてのスレッドチームのマスタースレッド間で共有するかどうかを指定します。SIMD 命令を使用して同時に実行される、スレッドチームのマスタースレッド間で分散されるループを指定します。

#pragma omp distribute *[節[[,]節] ...]*
for ループ
#pragma omp for distribute *[節[[,]節] ...]*
for ループ

節:
- **private**(リスト)
- **firstprivate**(リスト)
- **collapse**(n)
- **dist_schedule**(タイプ[, チャンクサイズ])

4.0 distribute parallel for [simd] [2.9.8, 2.9.9]

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 [simd 命令で同時に実行できる] ループを指定します。

#pragma omp distribute parallel for *[節[[,]節] ...]*
　for ループ
#pragma omp distribute parallel for simd *[節[[,]節] ...]*
　for ループ

節: **distribute** の節を参照してください。

宣言子 (続き)

parallel loop [2.10.1] [2.6.1]
1 つ以上のループ構造だけを含み、その他の文を含まない **parallel** 構文を簡潔に指定します。

#pragma omp parallel for *[節[[,]節] ...]*
for ループ
節: **nowait** 節を除き **parallel** もしくは **for** 宣言子と同一の節と制限が適用されます。

parallel sections [2.10.2] [2.6.2]
1 つ以上の **sections** 構文を含み、その他の文を含まない **parallel** 構文を簡潔に指定します。

#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*]*
...
}
節: **nowait** 節を除き **parallel** もしくは **sections** 宣言子と同一の節と制限が適用されます。

4.0 parallel loop simd [2.10.4]
1 つ以上のループ simd 構造だけを含み、その他の文を含まない **parallel** 構文を簡潔に指定します。

#pragma omp parallel for simd *[節[[,]節] ...]*
for ループ
節: **nowait** 節を除き **parallel** 、**for** もしくは **simd** 宣言子と同一の節と制限が適用されます。

4.0 target teams [2.10.5]
teams 構文を含む、**target** 構文を簡潔に指定します。

#pragma omp target teams *[節[[,]節] ...]*
構造化ブロック
節: **target** と **teams** の節を参照してください。

4.0 teams distribute [simd] [2.10.6, 2.10.7]
distribute [simd] 構文を含む、**teams** 構文を簡潔に指定します。

#pragma omp teams distribute *[節[[,]節] ...]*
for ループ
#pragma omp teams distribute simd *[節[[,]節] ...]*
for ループ
節: **teams** と **distribute [simd]** の節を参照してください。

4.0 target teams distribute [simd] [2.10.8, 2.10.9]
teams distribute [simd] 構文を含む、**target** 構文を簡潔に指定します。

#pragma omp target teams distribute *[節[[,]節] ...]*
for ループ
#pragma omp target teams distribute simd *[節[[,]節] ...]*
for ループ

節: **teams** と **teams distribute [simd]** の節を参照してください。

4.0 teams distribute parallel for [simd] [2.10.10, 12]
distribute parallel for [simd] 構文を含む、**teams** 構文を簡潔に指定します。

#pragma omp teams distribute parallel for *[節[[,]節] ...]*
for ループ
#pragma omp teams distribute parallel for simd *[節[[,]節] ...] for* ループ

節: **teams** と **distribute parallel for [simd]** の節を参照してください。

4.0 target teams distribute parallel for [simd] [2.10.11, 13]
teams distribute parallel for [simd] 構文を含む、**target** 構文を簡潔に指定します。

#pragma omp target teams distribute parallel for *[節[[,]節] ...] for* ループ
#pragma omp target teams distribute parallel for simd *[節[[,]節]...] for* ループ

節: **target** と **teams distribute parallel for [simd]** の節を参照してください。

task [2.11.1] [2.7.1]
明示的にタスクを定義します。タスクのデータ環境は、task 構文のデータ共有属性節と適用されるデフォルトに従って作成されます。

#pragma omp task *[節[[,]節] ...]*
構造化ブロック
節:
if(スカラー式)
final(スカラー式)
untied
default(**shared** | **none**)
mergeable
private(リスト)
firstprivate(リスト)
shared(リスト)
4.0 depend(依存性タイプ**:**リスト)

depend 節のリストは部分配列を含むことができます。

依存性タイプ: 生成されるタスクは、少なくとも 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。
- **in: out** または **inout** 節。
- **out** と **inout: in**、**out** または **inout** 節。

taskyield [2.11.2] [2.7.2]
現在のタスクを中断し、別のタスクの実行を優先することを許可します。

#pragma omp taskyield

master [2.12.1] [2.8.1]
チームのマスタースレッドで実行される構造化ブロックを指定します。

#pragma omp master
構造化ブロック

critical [2.12.2] [2.8.2]
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。

#pragma omp critical *[(名称)]*
構造化ブロック

barrier [2.12.3] [2.8.3]
構文が記述された位置に明示的なバリアを指定します。

#pragma omp barrier

taskwait [2.12.4] [2.8.4],
4.0 taskgroup [2.12.5]
これらの構文は、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。**taskgroup** は孫タスクを待ちます。

#pragma omp taskwait

#pragma omp taskgroup
構造化ブロック

atomic [2.12.6] [2.8.5]
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。
[seq_cst] は、**4.0** で追加されました。

#pragma omp atomic *[***read** | **write** | **update** | **capture***] [***seq_cst***]*
式文

#pragma omp atomic capture *[***seq_cst***]*
構造化ブロック

式文は、次の形式のいずれかです:

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

atomic (続き)
そして、構造化ブロックは次のいずれかの形式であってもかまいません:
{v = x; x binop= expr;}{x binop= expr; v = x;}
{v = x; x = x binop expr;}{v = x; x = expr binop x;}
{x = x binop expr; v = x;}{x = expr binop x; v = x;}
{v = x; x = expr;} {v = x; x++;}
{v = x; ++x;} {++x; v = x;}
{x++; v = x;} {v = x; x--;}
{v = x; --x;} {--x; v = x;}
{x--; v = x;}

flush [2.12.7] [2.8.6]
OpenMP の flush 操作を実行します。スレッドのテンポラリー・メモリーのビューとメモリーの一貫性、および変数に対するメモリー操作の順番を強制します。

#pragma omp flush *[(リスト)]*

ordered [2.12.8] [2.8.7]
ループの反復順に実行されるワークシェア・ループ内の構造化ブロックを指定します。

#pragma omp ordered
構造化ブロック

4.0 cancel [2.13.1]
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。**cancel** 宣言子は、if、while、do、switch もしくはラベルの次の文に使用することはできません。

#pragma omp cancel 構文タイプ節*[[,] if 節]*

構文タイプ節:
parallel
sections
for
taskgroup

if 節:
if(スカラー式)

4.0 cancellation point [2.13.2]
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場合に、タスク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。

#pragma omp cancellation point 構文タイプ節

構文タイプ節:
parallel
sections
for
taskgroup

threadprivate [2.14.2] [2.9.2]
変数が各スレッドで独自のコピーを持つ、複製であることを指定します。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。

#pragma omp threadprivate(リスト)
リスト:
カンマで区切られた、不完全な型を持たないファイルスコープ、名前空間、もしくは静的ブロック変数のリスト。

4.0 declare reduction [2.15]
reduction 節で使用できるリダクション演算子を宣言します。

#pragma omp declare reduction(リダクション演算子**:**タイプリスト**:**コンバイナー) *[initializer 節]*

リダクション演算子: ベース言語の演算子または次のいずれかの演算子: +, -, *, &, |, ^, && および ||。C++ では、オペレーター関数 ID を指定できます。
タイプリスト: タイプ名のリスト
コンバイナー: 式
initializer 節: **initializer** (**omp_priv** = 初期化子 | 関数名 (引数リスト))

ランタイム・ライブラリー

戻り型は緑色で示します。

実行環境ルーチンは、スレッド、プロセッサー、そして並列環境を制御し監視します。ライブラリー・ルーチンは、"C" リンクによる外部関数です。

実行環境ルーチン

omp_set_num_threads [3.2.1] [3.2.1]
現在のタスクの内部変数 *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。
void omp_set_num_threads(int *num_threads*);

omp_get_num_threads [3.2.2] [3.2.2]
現在のタスク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、最も内側の並列領域です。
int omp_get_num_threads(void);

omp_get_max_threads [3.2.3] [3.2.3]
num_threads 節を持たない **parallel** 構文でこのルーチンからリターンした場合、新しいチームを形成できる最大スレッドの上限を返します。
int omp_get_max_threads(void);

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。
int omp_get_thread_num(void);

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

omp_in_parallel [3.2.6] [3.2.6]
内部変数 *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。
int omp_in_parallel(void);

omp_set_dynamic [3.2.7] [3.2.7]
スレッド数の動的調整が有効か無効かを示す内部変数 *dyn-var* の値を設定します。
void omp_set_dynamic(int *dynamic_threads*);

omp_get_dynamic [3.2.8] [3.2.8]
このルーチンは内部変数 *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。
int omp_get_dynamic(void);

4.0 omp_get_cancellation [3.2.9]
cancel 構文と cancellation point の動作を制御する内部変数 *cancel-var* の値を返します。
int omp_get_cancellation(void);

omp_set_nested [3.2.10] [3.2.9]
入れ子構造の並列処理を有効または無効にする、*内部変数 nest-var* を設定します。
void omp_set_nested(int *nested*);

omp_get_nested [3.2.11] [3.2.10]
入れ子構造の並列処理が有効か無効かを示す内部変数 *nest-var* の値を返します。
int omp_get_nested(void);

omp_set_schedule [3.2.12] [3.2.11]
runtime がスケジュール・タイプとして使用されている場合に適用されるワークシェア・ループのスケジュールを決定します。
void omp_set_schedule(omp_sched_t *kind*, int *modifier*);
タイプ: 次のいずれか実装の定義でスケジュールします:
omp_sched_static = 1
omp_sched_dynamic = 2
omp_sched_guided = 3
omp_sched_auto = 4

omp_get_schedule [3.2.13] [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変数 *run-sched-var* の値を返します。
void omp_get_schedule(
omp_sched_t *kind, int *modifier);
前述の タイプ を参照してください。

omp_get_thread_limit [3.2.14] [3.2.13]
利用可能な OpenMP スレッドの最大数を示す、内部変数 *thread-limit-var* の値を返します。
int omp_get_thread_limit(void);

omp_set_max_active_levels [3.2.15] [3.2.14]
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変数 *max-active-levels-var* を設定します。
void omp_set_max_active_levels(int *max_levels*);

omp_get_max_active_levels [3.2.16] [3.2.15]
入れ子構造のアクティブな並列領域の最大数を決定する内部変数 *max-active-levels-var* の値を返します。
int omp_get_max_active_levels(void);

omp_get_level [3.2.17] [3.2.16]
呼び出しが含まれたタスクを囲む、入れ子構造の **parallel** 領域の数を示す内部変数 *levels-vars* の値を返します。
int omp_get_level(void);

omp_get_ancestor_thread_num [3.2.18] [3.2.17]
現在のスレッドの入れ子レベル、先祖のスレッド数を返します。
int omp_get_ancestor_thread_num(int *level*);

omp_get_team_size [3.2.19] [3.2.18]
現在のスレッドの入れ子レベル、先祖もしくは現在のスレッドに属するスレッドチームのサイズを返します。
int omp_get_team_size(int *level*);

omp_get_active_level [3.2.20] [3.2.19]
呼び出しが含まれるタスクのアクティブな入れ子の並列領域の数を決定する内部変数 *active-level-vars* の値を返します。
int omp_get_active_level(void);

omp_in_final [3.2.21] [3.2.20]
final タスク領域内で呼ばれた場合、*true* を返します。そうでない場合は、*false* を返します。
int omp_in_final(void);

4.0 omp_get_proc_bind [3.2.22]
proc_bind 節 を指定しない後続の入れ子になった **parallel** 領域に適用するスレッドのアフィニティー・ポリシーを返します。
omp_proc_bind_t omp_get_proc_bind(void);
次のいずれかを返します:
omp_proc_bind_false = 0
omp_proc_bind_true = 1
omp_proc_bind_master = 2
omp_proc_bind_close = 3
omp_proc_bind_spread = 4

4.0 omp_set_default_device [3.2.23]
デフォルトデバイスの内部変数 *default-device-var* を設定することで、デフォルトのターゲットデバイスを制御します。
void omp_set_default_device(int *device_num*);

4.0 omp_get_default_device [3.2.24]
デフォルトのターゲットデバイスを返します。
int omp_get_default_device(void);

4.0 omp_get_num_devices [3.2.25]
ターゲットデバイス数を返します。
int omp_get_num_devices(void);

4.0 omp_get_num_teams [3.2.26]
現在のチーム領域のチーム数を返します。または、チーム領域外から呼び出された場合 1 を返します。
int omp_get_num_teams(void);

4.0 omp_get_team_num [3.2.27]
呼び出されたスレッドのチーム数を返します。チーム数は、0 から **omp_get_num_teams** で返された値より 1 つ少ない整数値です。
int omp_get_team_num(void);

4.0 omp_is_initial_device [3.2.28]
現在のタスクがホストデバイス上で実行されていれば *true* を返します。それ以外は、*false* を返します。
int omp_is_initial_device(void);

ロックルーチン

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

ロックの初期化 [3.3.1] [3.3.1]
OpenMP ロックを初期化します。
void omp_init_lock(omp_lock_t *lock);
void omp_init_nest_lock(omp_nest_lock_t *lock);

ロックの破棄 [3.3.2] [3.3.2]
OpenMP ロックが初期化されていないことを確認します。
void omp_destroy_lock(omp_lock_t *lock);
void omp_destroy_nest_lock(omp_nest_lock_t *lock);

ロックをセット [3.3.3] [3.3.3]
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。
void omp_set_lock(omp_lock_t *lock);
void omp_set_nest_lock(omp_nest_lock_t *lock);

ロックをアンセット [3.3.4] [3.3.4]
OpenMP ロックを解除します。
void omp_unset_lock(omp_lock_t *lock);
void omp_unset_nest_lock(omp_nest_lock_t *lock);

ロックをテスト [3.3.5] [3.3.5]
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。
int omp_test_lock(omp_lock_t *lock);
int omp_test_nest_lock(omp_nest_lock_t *lock);

タイミングルーチン

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

omp_get_wtime [3.4.1] [3.4.1]
経過時間のウォールクロックを秒単位で返します。
double omp_get_wtime(void);

omp_get_wtick [3.4.2] [3.4.2]
omp_get_wtime で使用されるタイマーの精度 (秒のティック数) を返します。
double omp_get_wtick(void);

環境変数 [4]

環境変数は大文字で表され、それらに割り当てられる値は大文字と小文字が区別され、先頭と末尾にスペースを使用することができます。

4.0 OMP_CANCELLATION *policy* **[4.11]**
内部変数 *cancel-var* を設定します。*policy* は、**true** または **false** です。**true** の場合、cancel 構文と cancellation point が有効になり、取り消しが有効になります。

4.0 OMP_DEFAULT_DEVICE *device* **[4.13]**
device 構文で使用するデフォルトデバイス数を制御する内部変数 *default-device-var* を設定します。

4.0 OMP_DISPLAY_ENV *var* **[4.12]**
var が **TRUE** の場合、OpenMP バージョン番号と環境変数に関連する内部変数の値を *name=value* として実行時に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が表示されます。*var* が **FALSE** の場合、情報は表示されません。

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

OMP_MAX_ACTIVE_LEVELS *levels* **[4.9] [4.8]**
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変数 *max-active-levels-var* を設定します。

OMP_NESTED *nested* **[4.6] [4.5]**
入れ子になった並列処理を有効もしくは無効にする内部変数 *nest-var* を設定します。*nested* の有効な値は **true** または **false** です。

OMP_NUM_THREADS *list* **[4.2] [4.2]**
parallel 領域で使用するスレッド数向けの内部変数 *nthreads-var* を設定します。

4.0 OMP_PLACES *places* **[4.5]**
実行環境で利用可能な OpenMP スレッドのマップを定義する内部変数 *place-partition-var* を設定します。*places* is は、抽象的な名称 (**threads**, **cores**, **sockets**, または実装による定義)、か負ではない数のリストです。

OMP_PROC_BIND *policy* **[4.4] [4.4]**
対応する入れ子レベルの parallel 領域で使用するスレッド・アフィニティー・ポリシーを定義するグローバル内部変数 *bind-var* を設定します。 *policy* には、**true**、**false**、またはカンマで区切られた **master**、**close**、もしくは **spread** リストを指定できます。

OMP_SCHEDULE *type[,chunk]* **[4.1] [4.1]**
ランタイム・スケジュールのタイプとチャンクサイズを制御する内部変数 *run-sched-var* を設定します。有効な OpenMP スケジュールタイプは、**static**、**dynamic**、**guided**、および **auto** です。

OMP_STACKSIZE *size[***B | K | M | G***]* **[4.7] [4.6]**
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_THREAD_LIMIT *limit* **[4.10] [4.9]**
OpenMP プログラムに関連するスレッド数を制御する内部変数 *thread-limit-var* を設定します。

OMP_WAIT_POLICY *policy* **[4.8] [4.7]**
待機中のスレッドの動作に関する OpenMP 実装のヒントを提供する内部変数 *wait-policy-var* を設定します。 有効な *policy* の値は **ACTIVE** (待機するスレッドはプロセッサー時間を消費) もしくは **PASSIVE** です。

節

特定の宣言子で有効な一連の節を宣言子とともに説明します。ほとんどの節では、カンマで区切られたリストを指定できます。節で指定されるすべてのリスト項目は、ベースとなる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ないすべての節も宣言子で有効です。特定の宣言子で有効な一連の節を宣言子とともに説明します。

データ共有属性節 [2.14.3] [2.9.3]
データ共有属性は、構文中の節に名前がリストされている変数に対してのみ有効です。

default(**shared** | **none**)
構文内のすべての変数の参照は、暗黙的なデータ共有属性により共有され、parallel、task、および teams 構文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ます。

shared(リスト)
parallel、**task**、または **teams** 構文で生成されるタスクが共有する 1つ以上のリスト項目を宣言します。プログラマーは明示的なタスク領域の実行を完了する前に、その領域で共有されるストレージを確認する必要があります。

private(リスト)
task または 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ます。**private** 節でリストされる変数を構文中の任意の文で参照すると、各タスクは新しいリスト項目を受け取ります。

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、項文中ではオリジナルの変数が持つ値でそれらは初期化されます。

lastprivate(リスト)
暗黙のタスクもしくは SIMD レーンでプライベートにする 1 つ以上のリスト項目を宣言し、領域終了後にリストされた項目がオリジナルの値を更新します。

4.0 linear(リスト*[:*リニアステップ*]*)
SIMD レーンにプライベートにする 1 つ以上のリスト項目を宣言し、リスト項目はループ反復空間に対してリニアな関係を持ちます。

reduction(リダクション演算子:リスト)
リダクション演算子と 1 つ以上のリスト項目を指定します。構文中の実際のリダクション演算子は、リスト項目で宣言されたリダクション演算子と一致している必要があります。

リダクション操作（初期値）

+	(0)	|	**(0)**
*****	(1)	^	**(0)**
-	(0)	&&	**(1)**
&	(~0)	||	**(0)**

max (reduction リスト項目で表現可能な最小値)
min (reduction リスト項目で表現可能な最大値)

データコピー節 [2.14.4] [2.9.4]

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 領域に属している他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。

4.0 Map 節 [2.14.5]

map(*[マップタイプ:]リスト***)**
構文に関連するデバイスのデータ環境へタスクのデータ環境から変数をコピーします。マップタイプ:
alloc: リスト項目に対応するそれぞれの新しい変数が保持する値は未定義です。
to: 領域の入口で、新しいリスト項目は対応するオリジナルのリスト項目の値に初期化されます。
from: 領域の出口で、リスト項目の値は対応するオリジナルのリスト項目に割り当てられます。
tofrom: (デフォルト) 領域の入口で、新しいリスト項目は対応するオリジナルのリスト項目の値に初期化され、領域の出口で、リスト項目の値は対応するオリジナルのリスト項目に割り当てられます。

（続く）

4.0 SIMD 節 [2.8.1]

safelen(レングス)
SIMD 命令によって同時に 2 つの反復が実行できない場合、この値でより大きな論理的反復空間を指定します。

collapse(*n*)
正の整数式で入れ子になったループ反復をより大きな 1 つの反復空間のループに畳み込むことを指定します。

simdlen(レングス)
正の整数式で関数の同時引数の数を指定します。

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。*alignment* は正の整数式でなければなりません。オプションのアライメント・パラメーターが指定されない場合、ターゲット・プラットフォーム上の SIMD 命令の実装で定義されるデフォルトのアライメントが適用されます。

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定値の有するように 1 つ以上の引数を宣言します。

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

notinbranch
関数は常に 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。