

# OpenMP 3.0 C/C++ 構文の 概要



OpenMP API 仕様については、[www.openmp.org](http://www.openmp.org) でダウンロードしてください。

## 宣言子

OpenMP 実行宣言子は、後続の構造化ブロックや OpenMP 構文に適用されます。「構造化ブロック (structured-block)」とは、単文または先頭に入口が 1 つ、末尾に出口が 1 つの複合文です。

`parallel` 構文は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 [clause[ [, ]clause] ...] new-line  
    structured-block
```

指定節: `if (scalar-expression)`  
`num_threads (integer-expression)`  
`default (shared | none)`  
`private (list)`  
`firstprivate (list)`  
`shared (list)`  
`copyin (list)`  
`reduction (operator: list)`

ループ構文は、スレッドチーム内のループ反復の分散、実行を指定します。

```
#pragma omp for [clause[[, ] clause] ... ] new-line  
    for-loops
```

指定節: `private (list)`  
`firstprivate (list)`  
`lastprivate (list)`  
`reduction (operator: list)`  
`schedule (kind[,  
chunk_size]) collapse (n)`  
`ordered`  
`nowait`

最も一般的なループ形式を  
以下に示します。

```
for(var = lb;  
var relational-op b;  
var += incr)
```

`sections` 構文には、スレッドチーム内で分散、実行される構造化ブロックのセットが含まれています。

```
#pragma omp sections [clause[[, ] clause] ...] new-line  
{  
    [#pragma omp section new-line]  
        structured-block  
    [#pragma omp section new-  
        line structured-block ]  
...    (次のページで該当する指定節を参照してください。)  
}
```

## 宣言子 (続き)

指定節: `private (list)`  
`firstprivate (list)`  
`lastprivate (list)`  
`reduction (operator: list)`  
`nowait`

**single** 構文は、チーム内の 1 つのスレッドでのみ (マスターである必要はありません)、関連付けられた構造化ブロックが、暗黙的なタスクのコンテキストで実行されるよう指定します。

```
#pragma omp single [clause[[,] clause] ...] new-line  
structured-block
```

指定節: `private (list)`  
`firstprivate (list)`  
`copyprivate (list)`  
`nowait`

複合並列ワークシェアリング構文は、1 つのワークシェアリング構造のみからなる並列構造を指定するためのショートカットです。許容される指示節は、**parallel** 構文とワークシェアリング構造の両方で使用できる指示節です。

```
#pragma omp parallel for [clause[[,] clause] ...] new-line  
for-loop
```

```
#pragma omp parallel sections [clause[ [, ]clause] ...]  
new-line  
{  
[#pragma omp section new-line]  
structured-block  
[#pragma omp section new-line]  
structured-block ]  
...  
}
```

**task** 構文は明示的な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指定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 [clause[ [, ]clause] ...] new-line  
structured-block
```

指定節: `if (scalar-expression)`  
`untied`  
`default (shared | none)`  
`private (list)`  
`firstprivate (list)`  
`shared (list)`

**master** 構文は、チームのマスタースレッドにより実行される構造化ブロックを指定します。**master** 構文の入口にも出口にも暗黙的なバリアはありません。

```
#pragma omp master new-line  
structured-block
```

## 宣言子 (続き)

**critical** 構文は、関連付けられた構造化ブロックの実行を一度に 1 スレッドだけに制限します。

```
#pragma omp critical [(name)] new-line
    structured-block
```

**barrier** 構文は、この構文が置かれる位置に明示的なバリアを指定します。

```
#pragma omp barrier new-line
```

**taskwait** 構文は、現在のタスクが開始してから生成された子タスクの完了時点で待機するように指定します。

```
#pragma omp taskwait new-line
```

**atomic** 構文は、特定のストレージ・ロケーションをアトミックに更新し、複数のスレッドによる同時書き込みの危険性を回避するようにします。

```
#pragma omp atomic new-line
    expression-stmt
```

*expression-stmt*: 次のいずれかの形式を使用します。

```
x binop =
expr x++
++x
x--
--x
```

**flush** 構文は、OpenMP のフラッシュ操作を実行します。スレッドのメモリー一時ビューとメモリーとの一貫性が保たれ、変数によるメモリー操作の実行順序が制限されます。

```
#pragma omp flush [(list)] new-line
```

**ordered** 構文は、ループ領域中の構造化ブロックをループの反復順に実行するように指定します。これにより、**ordered** 領域内のコードはシーケンシャルに実行され、領域外のコードは並列に実行されます。

```
#pragma omp ordered new-line
    structured-block
```

**threadprivate** 宣言子は、変数を複製して、各スレッドが個別のコピーを持てるようにします。

```
#pragma omp threadprivate(list) new-line
```

## 指定節

すべての指定節が全宣言子で使用できるわけではありません。各宣言子で使用可能な指定節については、それぞれの宣言子の説明に記載されています。ほとんどの指定節をカンマ区切りのリストで指定できます。リストの項目はすべて識別できなければなりません。

### データ共有属性指定節

データ共有属性指定節は、指定節が指定されている構文で識別できる変数にのみ適用されます。

`default (shared | none) ;`

`parallel` または `task` 構文で参照される変数のデフォルトのデータ共有属性を制御します。

`shared (list) ;`

`parallel` または `task` 構文によって生成されるタスクで 1 つ以上のリスト項目を共有することを宣言します。

`private (list) ;`

タスクに対して 1 つ以上のリスト項目をプライベートにすることを宣言します。

`firstprivate (list) ;`

タスクに対して 1 つ以上のリスト項目をプライベートにすることを宣言して、構文に遭遇すると、対応するオリジナルの項目の値で初期化します。

`lastprivate (list) ;`

暗黙的なタスクに対して 1 つ以上のリスト項目をプライベートにすることを宣言して、領域が終わった後に対応するオリジナルの項目を更新します。

`reduction (operator : list) ;`

指定された演算子を使用して、リスト項目に累積します。各リスト項目のプライベート・コピーが作成され、その値でオリジナルの項目は更新されます。

### データコピー指定節

1 つの暗黙的なタスクやスレッドのプライベート変数またはスレッド・プライベート変数のデータ値を、チーム内の別の暗黙的なタスクやスレッドの対応する変数にコピーします。

`copyin (list) ;`

マスタースレッドの `threadprivate` 変数の値を、並列領域で実行するチームの各メンバーの `threadprivate` 変数にコピーします。

`copyprivate (list) ;`

1 つの暗黙的なタスクのデータ環境から並列領域に属している別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。

## ランタイム・ライブラリー・ルーチン

実行環境ルーチンは、スレッド、プロセッサ、および並列環境の監視と操作を行います。ロックルーチンは、OpenMP ロックとの同期をサポートします。タイミングルーチンは、移植性のあるウォール・クロック・タイマーをサポートします。ランタイム・ライブラリー・ルーチンのプロトタイプは `omp.h` ファイルで定義されています。

### 実行環境ルーチン

```
void omp_set_num_threads(int num_threads);
    num_threads 指定節を指定しない次の並列領域で使用されるスレッド数に影響します。

int omp_get_num_threads(void);
    現在のチームのスレッド数を返します。

int omp_get_max_threads(void);
    num_threads 指定節を指定しない parallel 構文を使用して作成可能な新しいチームのスレッド数の最大値を返します。

int omp_get_thread_num(void);
    遭遇したスレッドの ID を返します。ID は 0 から チームのサイズ - 1 の範囲です。

int omp_get_num_procs(void);
    プログラムで利用できるプロセッサ数を返します。

int omp_in_parallel(void);
    ルーチンの呼び出しが、アクティブな並列領域に囲まれている場合は、true を返します。それ以外の場合は、false を返します。

void omp_set_dynamic(int dynamic_threads);
    利用可能なスレッド数の動的な調整を有効または無効にします。

int omp_get_dynamic(void);
    スレッド数の動的な調整が有効か、無効かを示す dyn-var 内部制御変数 (ICV) を返します。

void omp_set_nested(int nested);
    nest-var ICV を設定して、入れ子された並列処理を有効または無効にします。

int omp_get_nested(void);
    入れ子された並列処理が有効か、無効かを示す nest-var ICV 値を返します。

void omp_set_schedule(omp_sched_t kind, int modifier);
    run-sched-var ICV 値を設定して、スケジュール種別が runtime の場合に適用されるスケジュールに影響します。

void omp_get_schedule(omp_sched_t *kind,
                      int *modifier);
    runtime スケジュールが使用されている場合に適用されるスケジュールを返します。
```

## ランタイム・ライブラリー・ルーチン (続き)

```
int omp_get_thread_limit(void)
```

プログラムで利用できる OpenMP スレッド数の最大値を返します。

```
void omp_set_max_active_levels(int max_levels);
```

*max-active-levels-var* ICV を設定して、入れ子されたアクティブな並列領域の数を制限します。

```
int omp_get_max_active_levels(void);
```

入れ子されたアクティブな並列領域の数を示す *max-activelevels-var* ICV 値を返します。

```
int omp_get_level(void);
```

呼び出しが含まれたタスクを囲む、入れ子された並列領域の数を返します。

```
int omp_get_ancestor_thread_num(int level);
```

指定された現在のスレッドの入れ子レベルに対する、先祖または現在のスレッド番号を返します。

```
int omp_get_team_size(int level);
```

指定された現在のスレッドの入れ子レベルに対する、先祖また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。

```
int omp_get_active_level(void);
```

呼び出しが含まれたタスクを囲む、入れ子されたアクティブな並列領域の数を返します。

## ロックルーチン

```
void omp_init_lock(omp_lock_t *lock);
```

```
void omp_init_nest_lock(omp_nest_lock_t *lock);
```

OpenMP ロックを初期化します。

```
void omp_destroy_lock(omp_lock_t *lock);
```

```
void omp_destroy_nest_lock(omp_nest_lock_t *lock);
```

OpenMP ロックを破棄します。

```
void omp_set_lock(omp_lock_t *lock);
```

```
void omp_set_nest_lock(omp_nest_lock_t *lock);
```

OpenMP ロックを設定するための手段を提供します。

```
void omp_unset_lock(omp_lock_t *lock);
```

```
void omp_unset_nest_lock(omp_nest_lock_t *lock);
```

OpenMP ロックを設定するための手段を提供します。

```
int omp_test_lock(omp_lock_t *lock);
```

```
int omp_test_nest_lock(omp_nest_lock_t *lock);
```

OpenMP ロックを設定しようと試みます。ルーチンを実行しているタスクは中断されません。

# ランタイム・ライブラリー・ルーチン (続き)

## タイミングルーチン

```
double omp_get_wtime(void);
```

経過したウォールクロック時間 (秒) を返します。

```
double omp_get_wtick(void);
```

`omp_get_wtime` で使用されるタイマーの精度を返します。

## 環境変数

環境変数名は大文字です。環境変数の値は大文字と小文字を区別せず、前後にスペースを含めることができます。

```
OMP_SCHEDULE type [, chunk]
```

ランタイム・スケジュールの型とチャンクサイズの `run-sched-var` ICV を設定します。OpenMP スケジュールの有効な型は `static`、`dynamic`、`guided`、または `auto` です。`Chunk` には正の整数を指定します。

```
OMP_NUM_THREADS num
```

並列領域で使用するスレッド数の `nthreads-var` ICV を設定します。

```
OMP_DYNAMIC dynamic
```

並列領域で使用するスレッド数の `dyn-var` ICV を設定します。`dynamic` の有効な値は `true` または `false` です。

```
OMP_NESTED nested
```

入れ子された並列処理を有効または無効にする `nest-var` ICV を設定します。`nested` の有効な値は、`true` または `false` です。

```
OMP_STACKSIZE size
```

OpenMP により作成されるスレッドのスタックサイズを指定する `stacksize-var` ICV を設定します。`size` (正の整数) の有効な値は、`size`、`sizeB`、`sizeK`、`sizeM`、`sizeG` です。`B`、`K`、`M` または `G` が指定されない場合、サイズは `K` (キロバイト) になります。

```
OMP_WAIT_POLICY policy
```

待機スレッドの動作を制御する `wait-policy-var` ICV を設定します。`policy` の有効な値は、`active` (待機中のスレッドはプロセッサ・サイクルを消費) および `passive` です。

```
OMP_MAX_ACTIVE_LEVELS levels
```

入れ子されたアクティブな並列領域の最大数を制御する `max-active-levels-var` ICV を設定します。

```
OMP_THREAD_LIMIT limit
```

OpenMP プログラムで使用されるスレッド数の最大値を制御する `thread-limit-var` ICV を設定します。

## 詳細

### リダクションで使用可能な演算子

演算子	初期値
+	0
*	1
-	0
&	~0
	0
^	0
&&	1
	0

### ループ構造のスケジュール型

- static** 反復は *chunk\_size* サイズのチャンクに分割され、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** 各スレッドは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。振り分けるチャンクがなくなるまで、これを繰り返します。
- guided** 各スレッドは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。大きなチャンクサイズから開始して、チャンクがスケジュールされるにしたがって、指定された *chunk\_size* になるまで、徐々に減少します。
- auto** スケジューリングに関する決定は、コンパイラやランタイムシステムが行います。
- runtime** `run-sched-var` ICV からスケジュールとチャンクサイズを取得します。

© 1997-2008 OpenMP Architecture Review Board.

本ドキュメントは、OpenMP Architecture Review Board の著作権と本ドキュメントのタイトルが記載され、OpenMP Architecture Review Board の許諾に基づく転載である旨が明記されている場合に限り、無料で一部または全体を転載できます。いかなる OpenMP の仕様に基づく製品または文献においても、次の著作権表示を明記しなければなりません。

“OpenMP is a trademark of the OpenMP Architecture Review Board. Portions of this product/publication may have been derived from the OpenMP Language Application Program Interface Specification.”