インテル® コンパイラー V19.0 による 並列プログラミング: OpenMP* 5.0 プレビュー Part 2

2018 年 10 月

iSUS 編集部
すがわら　きよふみ

このセッションの目的

明示的な並列プログラミング手法として注目されてきた OpenMP* による並列プログラミングに加え、インテル® コンパイラーがサポートする OpenMP* 4.0 と 4.5 の機能を使用したベクトル・プログラミングとオフロード・プログラミングの概要をリフレッシュし、インテル® コンパイラー V19.0 でサポートされる OpenMP* 5.0 の機能と実装を紹介します

セッションの対象者

- 既に OpenMP* でマルチスレッド・プログラミングを開発し、4.0 以降でサポートされる新たなベクトル化とオフロードを導入し、アプリケーションのパフォーマンス向上を計画する開発者

本日の内容

概要 (OpenMP* とは、歴史、各バージョンの機能概要)
OpenMP* 4.0、4.5 の機能と5.0 の新機能
- タスク
- omp simd
- オフロード
- メモリー操作
- アフィニティー

インテル® コンパイラー V19.0 での重要な変更点

- インテル® C/C++ および Fortran コンパイラー 19.0 では、デフォルトで /Qopenmp-simd (-qopenmp-simd) オプションが有効になります

```
#pragma omp simd (!$ OMP SIMD)
#pragma omp declare simd (!$ OMP DECLARE SIMD)

#pragma omp for simd
```

  無効にするには、/Qopenmp-simd- (-qopenmp-simd-) を指定します

- インテル® C/C++ および Fortran コンパイラー 19.0 では、/Qopenmp-offload:gfx (-qopenmp-offload=gfx) が未サポートとなりました

OpenMP* 5.0 のメモリーと同期管理

メモリー管理ディレクティブ

OpenMP メモリー空間は、変数を格納および取得できるストレージリソースを表します。特定のメモリー空間を選択するため、事前定義メモリー空間とアロケーター特性が定義されています

特定のメモリー空間の選択は、割り当てのため特定の特性を有するストレージを使用することを意味します。各メモリー空間の実際のストレージリソースは実装により定義されます

事前定義メモリー空間

メモリー空間名	ストレージ選択の意味
omp_default_mem_space	システムのデフォルトストレージを表します。
omp_large_cap_mem_space	大容量のストレージを表します。
omp_const_mem_space	定数値の変数向けに最適化されたストレージを表します。このストレージへの書き込み結果は不定です。
omp_high_bw_mem_space	高い帯域幅のストレージを表します。
omp_low_lat_mem_space	低レイテンシーのストレージを表します。

メモリー・アロケーター

OpenMP* のメモリー・アロケーターは、プログラムが割り当て要求に使用できます。メモリー・アロケーターは、特定**サイズ**のストレージ割り当て要求を受け取ると、少なくとも要求されたメモリー空間のリソース内に論理的に連続したサイズの*メモリー*を割り当てようとします

事前定義アロケーター

アロケーター名	関連するメモリー空間	デフォルト以外の特性値
omp_default_mem_alloc	omp_default_mem_space	(なし)
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

新しい allocate ディレクティブと節

V19.0 では未サポート

新しい allocate ディレクティブは、API 呼び出しで割り当てられない変数の割り当てを制御するために提案されました (自動変数または静的変数など)。OpenMP* ディレクティブによる割り当て操作に使用できます (変数のプライベート・コピーなど)

```
#pragma omp allocate(a,b) memtraits(bandwidth=highest, pagesize=2*1024*1024)
int a[N], b[M];

void example() {
#pragma omp parallel private(b) allocate(memtraits(latency=lowest):b)
        {
        // ...
        }
}
```

2MB ページを使用する最も高い帯域幅のメモリーに変数 a と b の割り当てを変更

OpenMP* 4.5 のロックのヒントを変更

現代のプロセッサー・アーキテクチャーの中には、トランザクショナル・メモリーをサポートするものがある。例えば、インテル® TSX (インテル® Transactional Synchronization Extensions)

この機能が同期 (ロック) を最適に実行するかどうかは、競合条件に依存

開発者は、コンパイラーの実装へこの情報を渡す意味を知っている必要がある

OpenMP* 4.5 では、これに相当する拡張を提供:

- 追加された OpenMP* API/ランタイムルーチン:

```
omp_init_lock_with_hint(omp_lock_t *lock,
                        omp_lock_hint_t hint)
omp_init_nest_lock_with_hint(omp_nest_lock_t *lock,
                        omp_lock_hint_t hint)
```

サポートされるヒント:

```
omp_lock_hint_none
omp_lock_hint_uncontended
omp_lock_hint_contended
omp_lock_hint_nonspeculative
omp_lock_hint_speculative
```

- critical 構文の新たな節 hint(type): “type” は、新しいロック API と同じ値を指定可能

OpenMP* 5.0 で同期のヒントを定義

ロックヒント (Lock hint) の名前が同期ヒント (synchronization hint) に変更されました。古い名前は廃止予定です

OpenMP* ロック機能の一部:

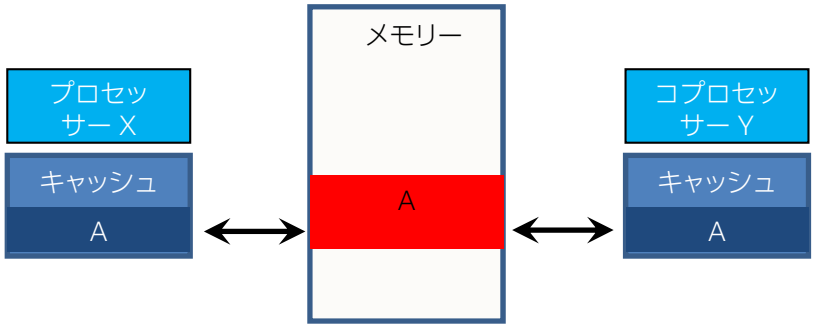
- C/C++ の omp_lock_hint_t と Fortran の omp_lock_hint_kind タイプを廃止予定
- および、ロックのヒント定数 omp_lock_hint_none、omp_lock_hint_uncontended、omp_lock_hint_contended、omp_lock_hint_nonspeculative、および omp_lock_hint_speculative を廃止

lock → sync

ヘテロジニアス・コンピューティングをサポートする target (または Offload) 構造の拡張

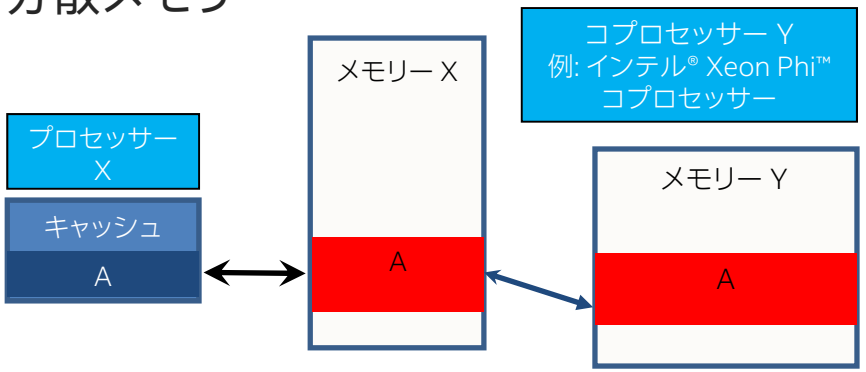
データ共有/マッピング：共有もしくは分散メモリー

共有メモリー



スレッドは共有メモリーへアクセスできる
共有データ向けに各スレッドは、同期バリア間の共有メモリーの一時的なビューを保持
スレッドはプライベート・メモリーを持つ
プライベート・データ向けに各スレッドは、実行される各タスクのローカル・データ・スタックを保持

分散メモリー



デバイスデータ環境に対応する変数は、元の変数とストレージを共有

対応する変数への書き込みは、元の変数の値を更新

OpenMP* 4.0/4.5 Target 拡張

ターゲットデバイス上で実行するためコードをオフロード

omp target [節[[,] 節],…] [nowait]

　　構造化ブロック

omp declare target

　　[関数定義または宣言]

ターゲットデバイスへ変数をマップ

map ([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 | tofrom | to | from | release | delete

　　マップタイプ修飾子: always

omp target [enter | exit] data [節[[,] 節],…]

　　構造化ブロック

omp target update [節[[,] 節],…]

omp declare target

　　[関数定義または宣言]

アクセラレーション向けのワークシェア

omp teams [節[[,] 節],…]

　　構造化ブロック

omp distribute [節[[,] 節],…]

　　for ループ

ランタイム･サポート･ルーチン

void omp_set_default_device(int dev_num)
int omp_get_default_device(void)
int omp_get_num_devices(void);
int omp_get_num_teams(void)
int omp_get_team_num(void);
Int omp_is_initial_device(void);

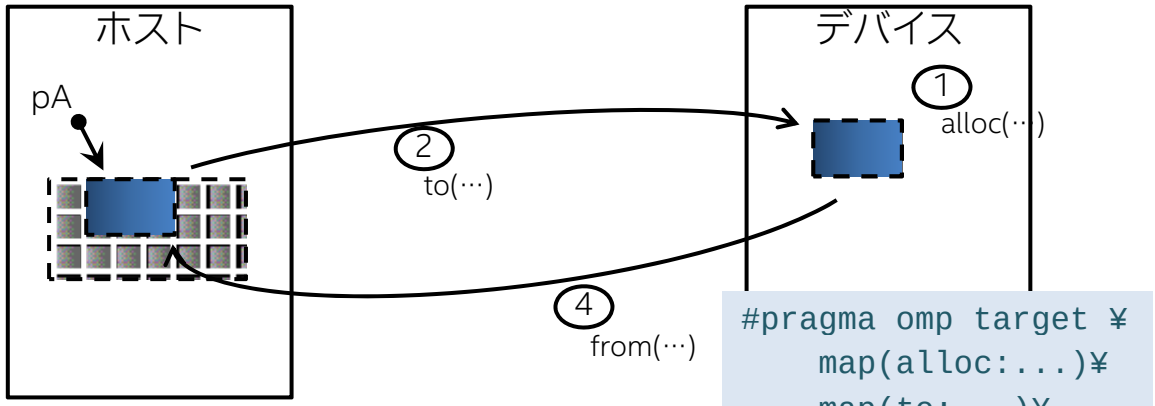
環境変数

OMP_DEFAULT_DEVICE を介してデフォルトデバイスを制御
負ではない整数値

赤字が OpenMP* 4.5 での拡張

オフロードとデバイスデータのマッピング

target 構文を使用して
ホストからターゲット
デバイスへ制御を転送
ホストとターゲットデバイス
のデータ環境間で変数を
マップ

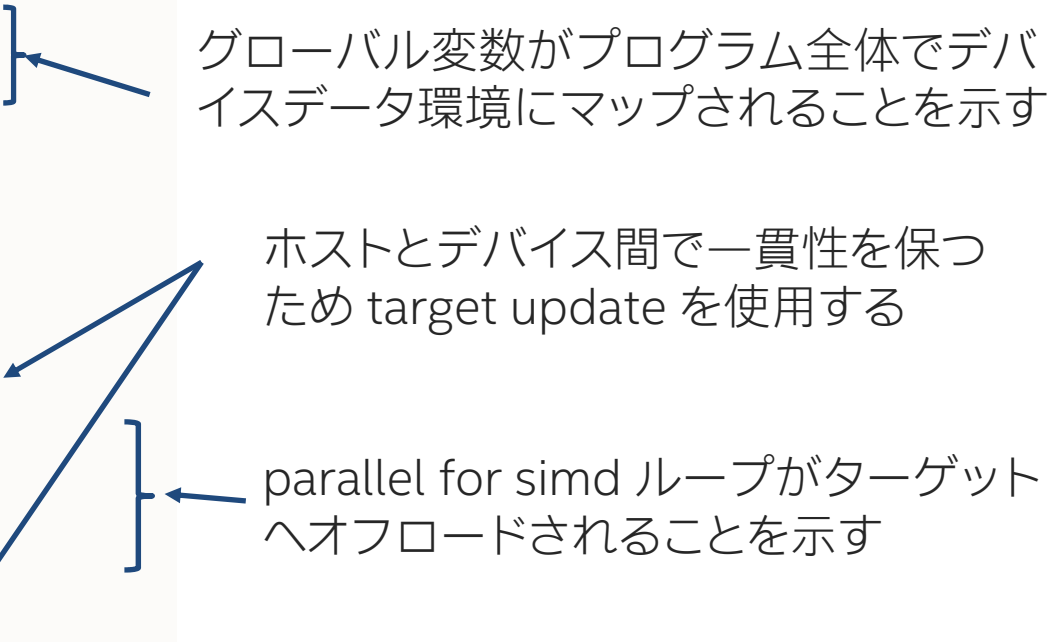


ホストスレッドはターゲット (オフロードされた) タスクをスポーン
同期オフロード (スレッドはターゲットタスクを待機)
非同期オフロード (スレッドはターゲットタスクを待機することなく継続)
map 節は、データ環境の元の変数をデバイスデータ環境の対応する変数にどのように
マップするかを決定する

例: target + map

```
#define N 1000
#pragma omp declare target
float p[N], v1[N], v2[N];
#pragma omp end declare target
extern void init(float *, float *, int);
extern void output(float *, int);
void vec_mult()
{
   int i;
   init(v1, v2, N);
   #pragma omp target update to(v1, v2)
   #pragma omp target
   #pragma omp parallel for simd
   for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];

   #pragma omp target update from(p)
   output(p, N);
}
```



例: OpenMP* 4.0 での非同期オフロード実装

OpenMP* 4.0 の target 構文は、非同期オフロードをサポートするため既存の OpenMP* の機能 (task) を活用できる

```
#pragma omp parallel sections
{
  #pragma omp task
  {
    #pragma omp target map(in:input[:N]) map(out:result[:N])
      #pragma omp parallel for
      for (i=0; i<N; i++) {
        result[i] = some_computation(input[i], i);
      }
  }
  #pragma omp task
  {
    do_something_important_on_host();
  }
}
```

例: OpenMP* 4.5 での非同期オフロード実装

非同期オフロードをサポートするため target 構文に nowait 節が追加された。
taskwait で待機

```
#pragma omp parallel sections
{
#pragma omp target map(in:input[:N]) map(out:result[:N]) nowait
  #pragma omp parallel for
      for (i=0; i<N; i++) {
        result[i] = some_computation(input[i], i);
      }
     // 以下をホストで非同期に実行
   do_something_important_on_host();
#pragma omp taskwait
}
```

ホスト
ターゲット
ホスト

teams+distriobute+parallel for を使用したオフロードの例

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
   // Define scalars n, a, b & initialize x, y
#pragma omp target data map(to:x[0:n])
{
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) thread_limit(nthreads)
```


```
#pragma omp distribute
  for (int i = 0; i < n; i += num_blocks){
```


```
#pragma omp parallel for
      for (int j = i; j < i + num_blocks; j++) {
```


```
        y[j] = a*x[j] + y[j];
      }
  }
} free(x); free(y); return 0; }
```

1 つ以上のループの反復を実行するスレッドチームを生成。マスタースレッドで実行を開始

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

distribute されたスレッドのチームで、ループ反復をワークシェア

OpenMP* と OpenACC* の比較例

OpenMP* 4.0 / 4.5 – チームとスレッド間で parallel for ループをアクセラレート

```
#pragma omp target teams map(X[0:N]) num_teams(numblocks)
#pragma omp distribute parallel for
  for (i=0; i<N; ++1) {
    X[i] += sin(X[i]);
  }
```

OpenACC* 2.0 / 2.5 – ギャングとワーカー間で for ループをアクセラレート

```
#pragma acc parallel copy(X[0:N]) num_gangs(numblocks)
#pragma acc loop gang worker
  for (i=0; i<N; ++i) {
    X[i] += sin(X[i]);
  }
```

OpenMP* と OpenACC* 対応

機能	OpenACC*	OpenMP*
オフロード	parallel または kernels	target
データ環境	data	target data
データ転送、変数定義	copy(変数)/ create(変数)	map(tofrom:変数) map(alloc:変数)
並列処理(チーム)	loop gang num_gangs(n)	teams distribute num_teams(n) thread_limit(n)
並列処理	loop vector vector_length(n)	parallel for

OpenMP* 5.0 の offload 拡張

- プログラマーの負担を軽減するため、いくつかの関数 (C、C++、Fortran) とサブルーチン (Fortran) で暗黙の declare target ディレクティブが追加されました
- 入れ子になった declare target ディレクティブのサポートが追加されました
- デバイス固有の関数実装をサポートするため、declare target ディレクティブに implements 節が追加されました
- 複雑なデータタイプのマッピングをサポートするため、declare mapper ディレクティブが追加されました
- 配列セクションへのマップで、ポインター変数へのマップ (C/C++) とデバイスメモリーのアドレス割り当てが追加されました
- スレッドがどのデバイスで実行されているかを特定するため、omp_get_device_num ランタイムルーチンが追加されました
- オフロード動作の制御をサポートするため、OMP_TARGET_OFFLOAD 環境変数が追加されました

暗黙の declare target ディレクティブ

オフロード領域で呼び出されている関数を自動的に検出して、それらの関数が declare target ディレクティブで指定されているかのように扱います。OpenMP* 4.5 では、オフロード領域で呼び出されるすべての関数は、declare target ディレクティブによって明示的にタグ付けされている必要がありました

```
#pragma omp declare target
void foo() {
        // ...
}
#pragma omp end declare target
void bar() {
#pragma omp target
        {
        foo();
        }
}
```

OpenMP* バージョン 4.5 のスタイル

```
void foo() {
        // ...
}
void bar() {
#pragma omp target
        {
        foo();
        }
}
```

OpenMP* バージョン 5.0 のスタイル

静的記憶域を含む変数の自動検出

OpenMP* 5.0 では、静的記憶域を含む変数も自動的に検出できます。これにより、次の 2 つの例は等価となります

```
int x;
#pragma omp declare target to (x)
void bar() {
#pragma omp target
        {
        x = 5;
        }
}
```

OpenMP* バージョン 4.5 のスタイル

```
int x;
void bar() {
#pragma omp target
        {
        x = 5;
        }
}
```

OpenMP* バージョン 5.0 のスタイル

配列セクションへのマップで、ポインター変数へのマップ (C/C++) とデバイスメモリーのアドレス割り当て

- OpenMP* バージョン 4.5 では、use_device_ptr 節が追加されましたが、use_device_ptr の変数は、使用する前にマップする必要があります。変数は 1 つのデータ節にのみ記述できるため、プログラマーは個別の#pragma target data 節を記述する必要がありました:
  - #pragma omp target data map(buf)
  - #pragma omp target data use_device_ptr(buf)

- OpenMP* 5.0 では、単一構文で変数を map 節と use_device_ptr 節の両方に記述できる例外が追加されました:
  - #pragma omp target data map(buf) use_device_ptr(buf)

つづき

- OpenMP* 4.5 では、最初の構造がターゲットである結合構造の reduction 節または lastprivate 節で使用されるスカラー変数は、ターゲット構造の firstprivate として扱われます。ホストの変数が更新されることはありません。ホストの変数を更新するには、プログラマーは結合構造から omp target ディレクティブを分離してスカラー変数を明示的にマップする必要があります

- OpenMP* 5.0 では、これらの変数は自動的にmap(tofrom:variable) が適用されているかのように扱われます

omp declare target の静的データメンバー

- OpenMP* 5.0 では、スタティック・データ・メンバーが omp declare target 構文内のクラスで使用できるようになりました
- また、スタティック・メンバーを含むクラス・オブジェクトは map 節でも使用できます

```
#pragma omp declare target
class C {
        static int x;
        int y;
}
class C myclass;
#pragma omp end declare target
void bar() {
#pragma omp target map(myclass)
        {
        myclass.x = 10
        }
}
```

入れ子になった target のサポート

- 外側の omp target data 構文内で構造体変数のフィールドをマップして、内側の入れ子の omp target 構文内で構造体変数のアドレスを使用すると、構造体の一部がすでにマップされている場合、構造体変数全体をマップしようとします

```
struct {int x,y,z} st;
int A[100];
#pragma omp target data map(s.x A[10:50])
{
#pragma omp target
        {
        A[20] = ; // OpenMP*4.5ではエラー、5.0ではOK
        foo(&st); // OpenMP*4.5ではエラー、5.0ではOK
        }
#pragma omp target map(s.x, A[10:50])
        {
        A[20] = ; // OpenMP*4.5と5.0の両方でOK
        foo(&st); // OpenMP*4.5と5.0の両方でOK
        }
}
```

OpenMP 5.0 では、プログラマーが想定した動作になるように、これらのケースが修正されました

ヘテロジニアス・プログラミングの向上

OpenMP* のデバイスサポートを向上のために次のような機能が検討されています:

- 現在、map 節の構造は、構造のポインターフィールドを含めて、ビット単位でコピーされます。委員会は、構造のポインターフィールドのサポートを拡張することにより、プログラマーが map 節を使用して構造のポインターフィールドの自動割り当て/ 割り当て解除を指定できるようにする拡張について議論しています
- 関数ポインターを target 領域で使用できるようにすること、および関数ポインターを declare target に記述できるようにすることを検討しています
- 非同期に実行できる新しいデバイス memcpy ルーチンのサポート
- target 構文の「デバイスで実行または失敗」セマンティクスのサポート。現在、デバイスが利用できない場合、ターゲット領域はホストで実行されます
- デバイスのみに存在し、ホストベースのコピーでない変数や関数のサポート
- 単一アプリケーションでの複数のデバイスタイプのサポート

その他の機能

OpenMP* 5.0 のツール・インターフェイス

OpenMP API を実装して開発された OpenMP プログラムの監視、パフォーマンス、または正当性の解析とデバッグをサポートする高品質で移植性のあるツールの開発を可能にするため、OpenMP API に OMPT と OMPD の 2 つのインターフェイスが追加されています

OMPT と OMPD

OMPT インターフェイスは、ファースト・パーティー・ツール向けに以下を提供:

- ファースト・パーティー・ツールを初期化するメカニズム
- ツールが OpenMP 実装機能の判断を可能にするルーチン
- ツールがスレッドに関連する OpenMP 状態情報の調査を可能にするルーチン
- ツールが実装レベルの呼び出しコンテキストをソースレベルに対応付けるためのメカニズム
- ツールが OpenMP イベントの通知を受信することを可能にするコールバック・インターフェイス
- ツールが OpenMP ターゲットデバイス上のアクティビティーをトレースすること可能にするトレース・インターフェイス
- アプリケーションがツールのコントロールに使用できるランタイム・ライブラリー・ルーチン

OMPD は以下を定義します:

- ツールが実行を開始したプログラムの OpenMP 状態にアクセスするのに使用する OMPD ライブラリーがエクスポートするインターフェイス
- ライブラリーが OMPD ライブラリーを使用して実行を開始したプログラムの OpenMP 状態にアクセスできるよう、ツールが OMPD ライブラリーに提供するコールバック・インターフェイス
- ツールが OpenMP 実装に使用する適切な OMPD ライブラリーを検出してイベントの通知を容易にするため、OpenMP 実装によって定義される最小限のシンボル情報

廃止予定の機能

ICV nest-var の廃止に伴う廃止予定の機能:

- OMP_NESTED 環境変数
- omp_set_nested および omp_get_nested API

OpenMP* ロック機能の一部

- C/C++ の omp_lock_hint_t と Fortran の omp_hint_hint_kind タイプを廃止予定
- および、ロックのヒント定数 omp_lock_hint_none、omp_lock_hint_uncontended、omp_lock_hint_contended、omp_lock_hint_nonspeculative、および omp_lock_hint_speculative

インテル® コンパイラーの OpenMP* サポート状況

- インテル® コンパイラー 16.0 は、ほとんどすべての OpenMP* 4.0 といくつかの OpenMP* 4.5 仕様をサポート
- インテル® コンパイラー 17.0 は大部分の OpenMP* 4.5 仕様をサポート
  - ロックとクリティカル領域のトランザクショナル・メモリー・サポートを除く
  - 2016 年 9 月リリース
- インテル® コンパイラー 18.0 は OpenMP* 5.0 TR4 をサポート
  - 2017 年 9 月リリース
- インテル® コンパイラー 19.0 は OpenMP* 5.0 TR6 の一部をサポート
  - 2018 年 9 月リリース

参考資料

- OpenMP* 5.0 TR7 の仕様抜粋訳
- インテル® コンパイラー 18.0 の OpenMP* 5.0 サポート
- OpenMP* 誕生から 20 年 Part1
- OpenMP* 誕生から 20 年 Part2
- 現在と将来の OpenMP* API 仕様
- OpenMP* バージョン 4.5: 標準化の進化
- OpenMP サポートのまとめ

オンライン・トレーニング・コース:

- インテル® コンパイラーによる OpenMP* 入門 (8回)
- OpenMP* 4.x による新しいレベルの並列化 (2回)
- インテル® VTune™ Amplifier + OpenMP* によりスレッドのパフォーマンスとスケーラビリティーを向上する
- インテル® ソフトウェア開発製品 ライブ・ウェビナー・シリーズ