OpenMP* 5.0:
スレッドとタスクに関する進化

工学博士 (Dr.-Ing) Michael Klemm
インテル社 シニア･アプリケーション･エンジニア、
OpenMP* アーキテクチャー･レビュー委員会 最高経営責任者 (CEO)

https://techdecoded.intel.io/essentials/openmp-5-0-a-story-about-threads-and-tasks/

概要

OpenMP* のワークシェア

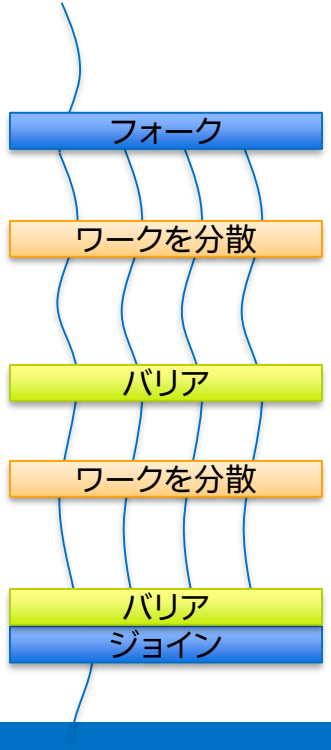
要約: OpenMP* のタスク処理

OpenMP* API バージョン 5.0 機能

OpenMP* のワークシェア

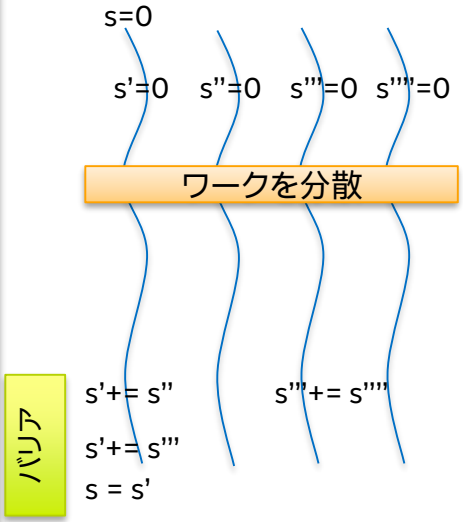
OpenMP* (以前の) ワークシェア処理

```
#pragma omp parallel
{
    #pragma omp for
    for (i = 0; i<N; i++)
    {…}

    #pragma omp for
    for (i = 0; i< N; i++)
    {…}
}
```



OpenMP* (以前の) ワークシェア処理 (2)

```
double a[N];
double l, s = 0;
#pragma omp parallel for reduction(+:s) ¥
        private(l) schedule(static,4)
for (i = 0; i<N; i++)
{
   l = log(a[i]);
   s += l;
}
```



OpenMP* タスクベースの実行モデル

構造化されていない並列処理のサポート

- 無限ループ

```
while ( <expr> ) {
    ...
}
```

- 再帰関数

```
void myfunc( <args> )
{
    ...; myfunc( <newargs> ); ...;
}
```

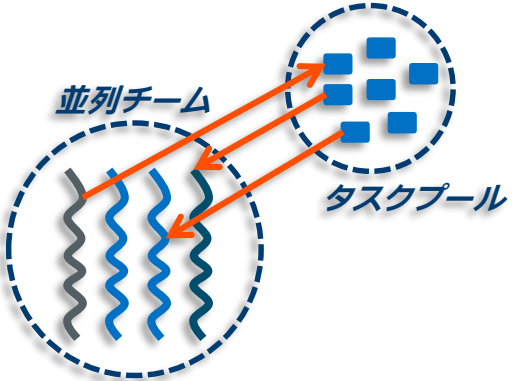
多くの可能性があります:

- 単一のクリエーター
- 複数のクリエーター
- 入れ子になったタスク (タスクとワークシェア)

チーム内のすべてのスレッドはタスクを実行できます

例:

```
#pragma omp parallel
#pragma omp master
while (elem != NULL) {
    #pragma omp task
        compute(elem);
    elem = elem->next;
}
```



OpenMP* バージョン5.0 における並列処理レベル

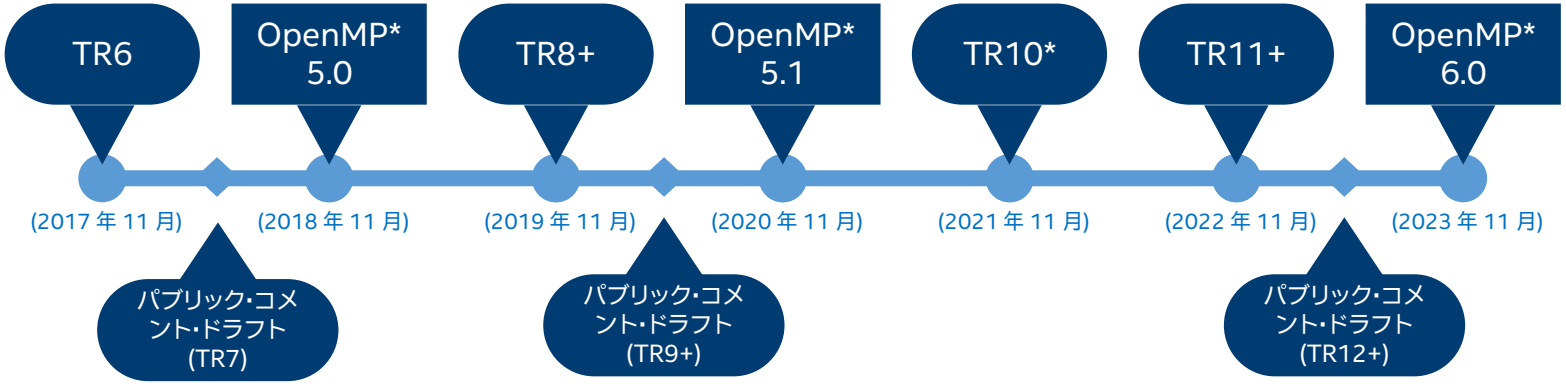
クラスター	コンピューターのグループ 高速相互接続を介して通信
コプロセッサー/アクセラレーター	特殊な計算デバイス 特別なインターコネクトを介してローカルノードに接続
ノード	キャッシュ・コヒーレントなプロセッサーのグループ 共有メモリー/キャッシュを介して通信
コア	ダイ上の機能ユニットグループ レジスターを介して通信
ハイパースレッド	機能ユニットを共有するスレッド・コンテキストのグループ
スーパースカラー	機能ユニットを共有する命令グループ
パイプライン	機能ユニットを共有する命令のシーケンス
ベクトル	複数の機能ユニットを使用する単一命令

OpenMP* ロードマップ

主要リリースに向けた今後 5 年間の計画

主要リリースの間には、1 つのマイナーリリースが発表されます (例えば 5.1)

(少なくとも) 1 つのテクニカル・レポート (TR) と機能レビューが毎年行われます



+ TR に割り当てられる番号は追加の TR がリリースされると変更されます

OpenMP* バージョン 5.0

OpenMP* 5.0 では、プログラミングの容易性を改善するため新たに強力な機能を導入

タスクのリダクション

デタッチ可能なタスク

メモリー・アロケーター

ツール API

初期の C11、C++11、C++14 および C++17 サポート

依存関係オブジェクト

Fortran 2003 を完全にサポート、Fortran 2008 の初期サポート

アフィニティー・サポートの向上

ユニファイド共有メモリー

loop 構造

非矩形ループの折りたたみ

タスクからデータへのアフィニティー

複数レベルの並列処理

オフロードでのデータシリアル化

並列スキャン

メタ・ディレクティブ

関数バリアント

リバースオフロード

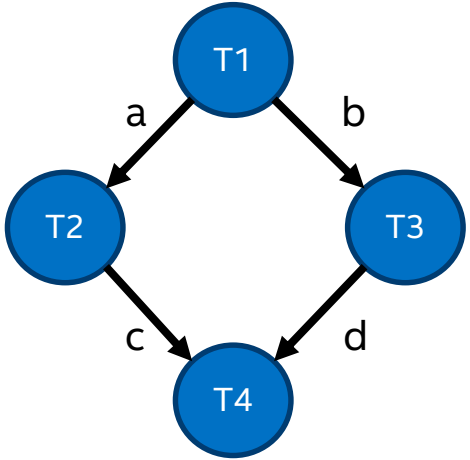
タスク依存関係の改善

相互運用性と使い勝手の向上

要約: OpenMP* のタスク処理

タスクの同期: タスクの依存関係

```
subroutine task_dep() {
    integer :: a, b, c, d
!$omp task depend(out:a, b)                    ! T1
    a = 0
    b = 0
!$omp end task

!$omp omp task depend(in:a) depend(out:c)  ! T2
    c = computation_1(a)
!$omp end task

!$omp task depend(in:b) depend(out:d)       ! T3
    d = computation_2(b)
!$omp end task

!$omp task depend(in:c,d)                   ! T4
    computation_3(c, d)
!$omp end task
end subroutine
```



OpenMP* API におけるタスク依存関係の歴史

OpenMP* API バージョン 4.0

- task 構造に depend 節を追加

OpenMP* API バージョン 4.5

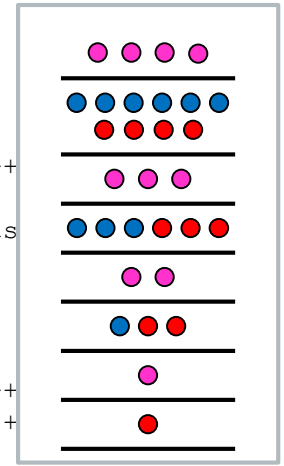
- target 構造に depend 節を追加
- doacross ループをサポート

OpenMP* API バージョン 5.0

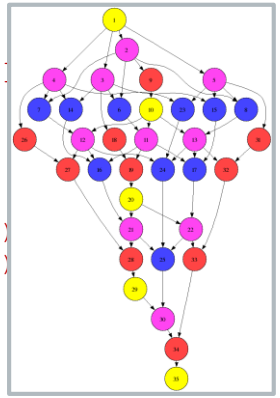
- depend 節の lvalue 式
- 新しい依存関係タイプ: mutexinoutset
- depend 節にイテレーターを追加
- taskwait 構造に depend 節を追加
- 依存関係をストアするオブジェクト

OpenMP* タスク処理の例: コレスキー分解

```
void cholesky(int ts, int nt, double* a[nt][nt]) {
  for (int k = 0; k < nt; k++) {
  // 対角ブロック分解
    potrf(a[k][k], ts, ts);

  // 三角システム
    for (int i = k + 1; i < nt; i++
      #pragma omp task
    trsm(a[k][k], a[k][i], ts, ts
    }
    #pragma omp taskwait

  //  末端の行列を更新
    for (int i = k + 1; i < nt; i++
      for (int j = k + 1; j < i; j+
        #pragma omp task
      dgemm(a[k][i], a[k][j], a[j][i], ts, ts);
      }
      #pragma omp task
    syrk(a[k][i], a[i][i], ts, ts);
    }
    #pragma omp taskwait
  }
}
```



```
void cholesky(int ts, int nt, double* a[nt][nt]) {
  for (int k = 0; k < nt; k++) {
  // 対角ブロック分解
    #pragma omp task depend(inout: a[k][k]
  potrf(a[k][k], ts, ts);

  // 三角システム
    for (int i = k + 1; i < nt; i++) {
      #pragma omp task depend(in: a[k][k])
                       depend(inout: a[k][i])
    trsm(a[k][k], a[k][i], ts, ts);
    }

  //  末端の行列を更新
    for (int i = k + 1; i < nt; i++) {
      for (int j = k + 1; j < i; j++) {
        #pragma omp task depend(inout: a[j][i]) ¥
                         depend(in: a[k][i], a[k][j])
      dgemm(a[k][i], a[k][j], a[j][i], ts, ts);
      }
      #pragma omp task depend(inout: a[i][i]) ¥
                       depend(in: a[k][i])
    syrk(a[k][i], a[i][i], ts, ts);
} } }
```

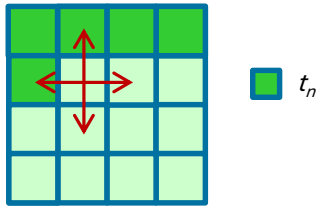


タスク処理の活用例: ガウスザイデル法

```
void serial_gauss_seidel(int tsteps, int size, int (*p)[size]) {
  for (int t = 0; t < tsteps; ++t) {
    for (int i = 1; i < size-1; ++i) {
      for (int j = 1; j < size-1; ++j) {
        p[i][j] = 0.25 * (p[i][j-1] * // left
                          p[i][j+1] * // right
                          p[i-1][j] * // top
                          p[i+1][j]); // bottom
      }
    }
  }
}
```

アクセスパターンの解析

固有の t、i および j



それぞれのセルには次のような依存関係があります:

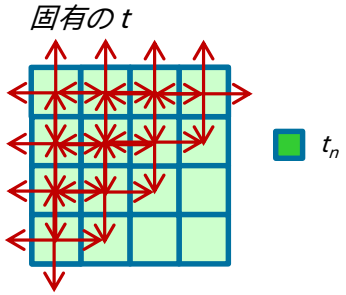
- 現在の時間ステップで計算された 2 つのセル (上と左)、
- および直前の時間ステップで計算された 2 つのセル (下と右)

タスク処理の活用例: ガウスザイデル法

```
void serial_gauss_seidel(int tsteps, int size, int (*p)[size]) {
  for (int t = 0; t < tsteps; ++t) {
    for (int i = 1; i < size-1; ++i) {
      for (int j = 1; j < size-1; ++j) {
        p[i][j] = 0.25 * (p[i][j-1] * // left
                          p[i][j+1] * // right
                          p[i-1][j] * // top
                          p[i+1][j]); // bottom
      }
    }
  }
}
```

変換が必要: ウェーブフロントを生成するループの傾斜とウェーブ間でスレッドの同期を行うバリア。

スレッドとウェーブフロント:



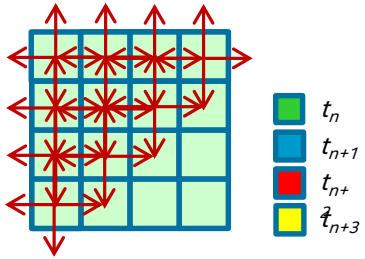
タスク処理の活用例: ガウスザイデル法

```
void gauss_seidel(int tsteps, int size, int TS, int (*p)[size]) {
  int NB = size / TS;
  #pragma omp parallel
  for (int t = 0; t < tsteps; ++t) {
  // 最初の NB 対角線
    for (int diag = 0; diag < NB; ++diag) {
      #pragma omp for
      for (int d = 0; d <= diag; ++d) {
        int ii = d;
        int jj = diag - d;
        for (int i = 1+ii*TS; i < ((ii+1)*TS); ++i)
          for (int j = 1+jj*TS; i < ((jj+1)*TS); ++j)
            p[i][j] = 0.25 * (p[i][j-1] * p[i][j+1] *
                              p[i-1][j] * p[i+1][j]);
      } // バリア
    }
  // 最後の NB 対角線
    for (int diag = NB-1; diag >= 0; --diag) {
   // 前のループと同様のコード
} } }
```

タスク処理の活用例: ガウスザイデル法

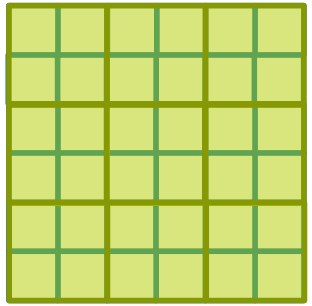
```
void serial_gauss_seidel(int tsteps, int size, int (*p)[size]) {
  for (int t = 0; t < tsteps; ++t) {
    for (int i = 1; i < size-1; ++i) {
      for (int j = 1; j < size-1; ++j) {
        p[i][j] = 0.25 * (p[i][j-1] * // left
                          p[i][j+1] * // right
                          p[i-1][j] * // top
                          p[i+1][j]); // bottom
      }
    }
  }
}
```

複数、並列ウェーブフロント

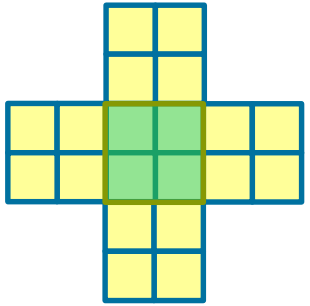


タスク処理の活用例: ガウスザイデル法

```
void gauss_seidel(int tsteps, int size, int TS, int (*p)[size]) {
  int NB = size / TS;

  #pragma omp parallel
  #pragma omp single
  for (int t = 0; t < tsteps; ++t)
    for (int ii=1; ii < size-1; ii+=TS)
      for (int jj=1; jj < size-1; jj+=TS) {
        #pragma omp task depend(inout: p[ii:TS][jj:TS])
            depend(in: p[ii-TS:TS][jj:TS], p[ii+TS:TS][jj:TS],
                       p[ii:TS][jj-TS:TS], p[ii:TS][jj:TS])
        {
          for (int i=ii; i<(1+ii)*TS; ++i)
            for (int j=jj; j<(1+jj)*TS; ++j)
               p[i][j] = 0.25 * (p[i][j-1] * p[i][j+1] *
                                 p[i-1][j] * p[i+1][j]);
}     } }
```



追跡する依存関係の数を減らす依存関係パターン



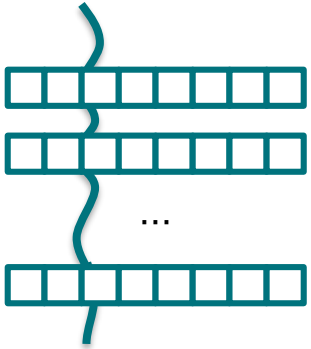
OpenMP* API バージョン 5.0 機能

loop 構造

既存のループ構造は、それらの実行モデルと密接に関係:

```
#pragma omp parallel for
for (i=0; i<N;++i) {…}
```

```
#pragma omp simd
for (i=0; i<N;++i) {…}
```




```
#pragma omp taskloop
for (i=0; i<N;++i) {…}
```

loop 構造は OpenMP* 実装が並列ループに適した並列化スキームを選択することを可能にします

loop 構造

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
 // スカラー n、a、b を定義し、x、y を初期化

#pragma omp parallel
  {
#pragma omp loop
    for (int i = 0; i < n; ++i){
      y[i] = a*x[i] + y[i];
    }
  }
}
```

metadirective ディレクティブ

異なる OpenMP* コンテキスト用の OpenMP* ディレクティブ

OpenMP* ディレクティブと句に対する限定形式のメタ・プログラミング

```
#pragma omp metadirective ¥
            when( device={isa(avx2)} : parallel for simd ) ¥
            default( parallel for )
for (i = lb; i < ub; i++)
   v3[i] = v1[i] * v2[i];
```

例: 実際の名前は異なることがあります

Declare Variant

OpenMP* 4.5 では、複数バージョンの関数を作成できます

```
#pragma omp declare simd simdlen(4)
#pragma omp declare simd simdlen(8)
double important_stuff(double x)
{
  // 関数 'important_stuff'のコード
}
```

declare variant はユーザー定義バージョンの関数を挿入します

```
#pragma omp declare variant(int important_stuff(int x)) ¥
            match( context={simd(simdlen(4))}, device={isa(avx2)} )
__m256 _mm256_important_stuff(__m256 x);
{
    /*
    関数 'important_stuff' の特殊化されたコード
    インテル® AVX2 を実行するプロセッサーで呼び出されます
      */
}
```

例: 実際の名前は異なることがあります

メモリー・アロケーター

データ共有節を含むすべての構造に対する新しい句

- `allocate( [allocator:] list )`

割り当て:

- `omp_alloc(size_t size, omp_allocator_t *allocator)`

解放:

- `omp_free(void *ptr, const omp_allocator_t *allocator)`
- `allocator` 引数はオプションです

allocate ディレクティブ

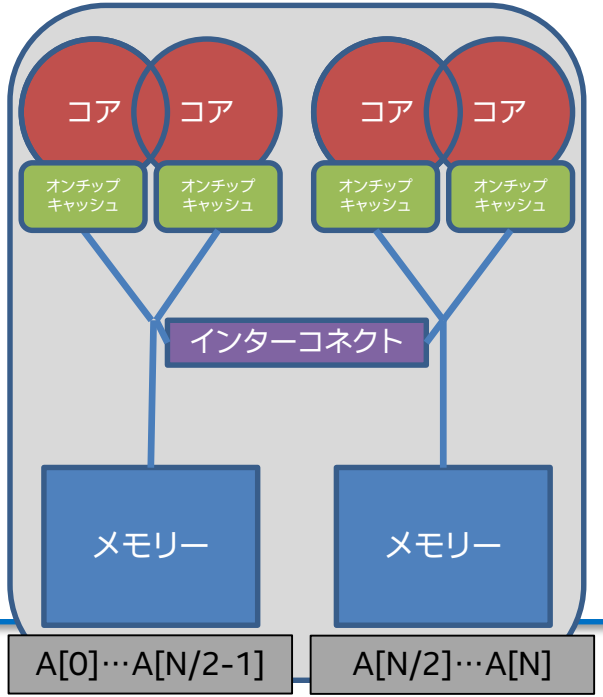
- 割り当て、割り当ての解放向けのスタンドアロン・ディレクティブ

例：メモリー・アロケーターの使用

```
void allocator_example(omp_allocator_t *my_allocator) {
    int a[M], b[N], c;
    #pragma omp allocate(a) allocator(omp_high_bw_mem_alloc)
    #pragma omp allocate(b) // OMP_ALLOCATOR と/または omp_set_default_allocator で制御されます
    double *p = (double *) omp_alloc(N*M*sizeof(*p), my_allocator);

    #pragma omp parallel private(a) allocate(my_allocator:a)
    {
        some_parallel_code();
    }

    #pragma omp parallel private(b) allocate(omp_high_bw_mem_alloc:b)
    {
        some_other_parallel_code();
    }

    omp_free(p);
}
```

メモリーのパーティション化

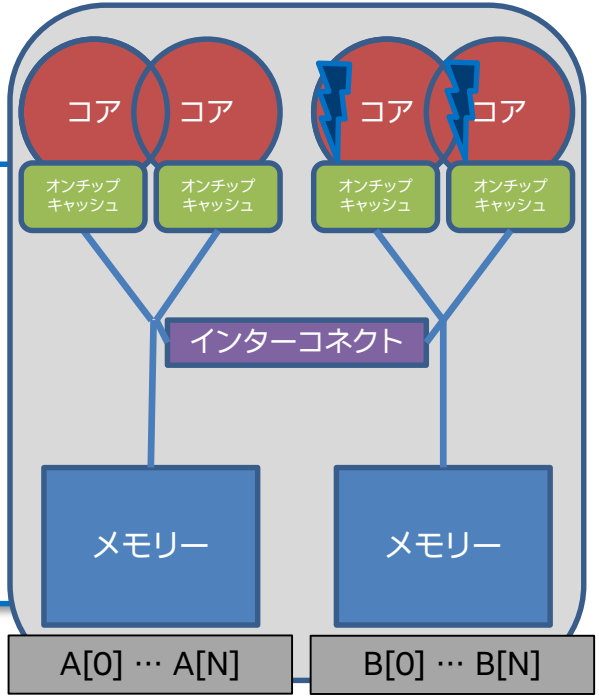
```
void allocator_example() {
    double *array;

    omp_allocator_handle_t allocator;
    omp_alloctrait_t traits[] = {
        {omp_atv_partition, omp_atv_blocked}
    };
    int ntraits = sizeof(traits) / sizeof(*traits);
    allocator = omp_init_allocator(omp_default_mem_space,
                                   ntraits, traits);

    array = omp_alloc(sizeof(*array) * N, allocator);

#pragma omp parallel for proc_bind(spread)
    for (int i = 0; i < N; ++i) {
        important_computation(&array[i]);
    }

    omp_free(array, allocator);
}
```



タスクからデータへのアフィニティー

OpenMP* API バージョン 5.0 は、OpenMP* タスクのアフィニティー・ヒントをサポートします

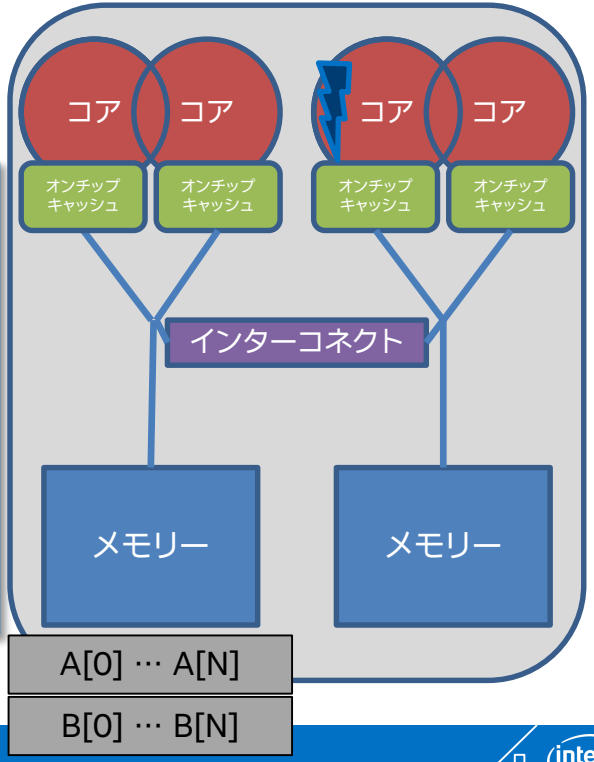
```
void task_affinity(double *A, size_t n) {
    double* B;
#pragma omp task shared(B) depend(out:B)
    {
        B = init_B_and_important_computation(A, n);
    }
#pragma omp task shared(B) depend(in:B)
    {
        important_computation_too(B, n);
    }
#pragma omp taskwait
}
```



タスクからデータへのアフィニティー

OpenMP* API バージョン 5.0 は、OpenMP* タスクのアフィニティー・ヒントをサポートします

```
void task_affinity() {
    double* B;
#pragma omp task shared(B) depend(out:B) affinity(A[0:N])
    {
        B = init_B_and_important_computation(A);
    }
#pragma omp task firstprivate(B) depend(in:B) affinity(A[0:N])
    {
        important_computation_too(B);
    }
#pragma omp taskwait
}
```



タスクのリダクション

タスク・リダクションは、従来のリダクションを任意のタスクグラフに拡張

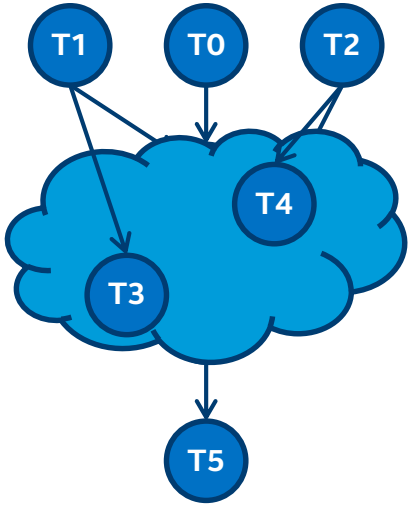
既存の task と taskgroup 構造を拡張

同様に taskloop 構造でも動作

```
int res = 0;
node_t* node = NULL;
...
#pragma omp parallel
{
   #pragma omp single
   {
      #pragma omp taskgroup task_reduction(+: res)
      {
         while (node) {
            #pragma omp task in_reduction(+: res) ¥
                             firstprivate(node)
            {
               res += node->value;
            }
            node = node->next;
         }
      }
   }
}
```

タスクの依存性

```
int x = 0, y = 0, res = 0;
#pragma omp parallel
#pragma omp single
{
  #pragma omp task depend(out: res)  //T0
  res = 0;

  #pragma omp task depend(out: x)  //T1
  long_computation(x);

  #pragma omp task depend(out: y)  //T2
  short_computation(y);

  #pragma omp task depend(in: x) depend(inout: res)  //T3
  res += x;

  #pragma omp task depend(in: y) depend(inout: res)  //T4
  res += y;

  #pragma omp task depend(in: res)  //T5
  std::cout << res << std::endl;
}
```



まとめ

OpenMP* 5.0 は前進への大きな飛躍

- 多くの主要機能を新たに追加し、既存の機能を大幅に強化
- 並列処理を表現する新しい方法
- ツール向けの定義されたインターフェイス

OpenMP* は現代風のディレクティブ・ベースのプログラミング・モデル

- 指示と記述のセマンティクス間の優れたバランスを提供
- 複雑なアルゴリズム向けの強力な言語機能
- ハイレベルな並列処理; 高効率のプログラミングへの道
- スレッド、タスク、そして SIMD の複数レベルの並列処理をサポート
- タスクベースのプログラミング・モデルは、並列処理の現代的な手法

インテル® コンパイラーのサポート状況

Intel Compiler 19.0 Update 4 Intel(R) 64 Visual Studio 2017

```
$ icl
インテル(R) 64 対応インテル(R) C++ コンパイラー (インテル(R) 64 対応アプリケーシ
ョン用) バージョン 19.0.4.228 ビルド 20190417
(C) 1985-2019 Intel Corporation.  無断での引用、転載を禁じます。

icl: コマンドライン・エラー: ファイルが指定されていません。ヘルプを表示するには
"icl /help" と入力してください。

$ icl version.c /Qopenmp /nologo
version.c

$ version
OpenMP version 201611
```

Q&A

Q: What's the timeline for Intel® Compiler to support OpenMP 5.0?
A: We will likely see OpenMP 5.0 support in the September 2019 release of the Intel Compiler.

Q: It would be great if there are discussions on gcc/clang vs Intel compiler support of OpenMP 5.
A: See https://www.openmp.org/resources/openmp-compilers-tools/

すべての Q&A はこちらでご覧いただけます:

https://techdecoded.intel.io/webinar-qas-from-openmp-5-0-a-story-about-threads-and-tasks-webinar/

参考資料

- 現在と将来の OpenMP* API 仕様
  https://www.isus.jp/hpc/pu27-01-present-and-future-of-openmp/
- OpenMP* 誕生から 20 年
  https://www.isus.jp/products/c-compilers/pu28-02-happy-20th-bd-openmp/
- OpenMP* 20 周年
  https://www.isus.jp/products/c-compilers/pu29-02-openmp_is_turning_20/
- 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
  https://www.isus.jp/products/vtune/better_threaded_performance_and_scalability/

TECH. DECODED の他のビデオ

- OpenMP* 5.0: A Story about Threads and Tasks
- OPENMP*: The Once and Future API
- OpenMP and TBB Task Graphs: Unraveling the Spaghetti with Flow Graph Analyzer

著作権と商標についておよび最適化に関する注意事項

性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。 より詳しい情報は、www.intel.com/benchmarks をご覧ください。

本資料に掲載されている情報は現状のまま提供され、いかなる保証もいたしません。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスを許諾するためのものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他、知的財産権の侵害への保証を含む) をするものではありません。

© 2019 Intel Corporation および OpenMP Architecture Review Board. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation またはその子会社の商標です。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

最適化に関する注意事項

インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。これには、インテル® ストリーミング SIMD 拡張命令 2、インテル® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。

注意事項の改訂 #20110804

intel
Software