インテル® コンパイラーを使用した OpenMP* による並列プログラミング

セッション 4: OpenMP* のオフロード機能

IA Software User Society (iSUS)
編集長　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 V19.1 でサポートされる OpenMP* 5.0 の機能と実装を紹介します。さらに新たなアクセラレーター・デバイスへのオフロードについて考えます

セッションの対象者

すでに 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

© 2020 iSUS *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。

セッションリスト

セッション	説明
はじめに (13:35 – 14:20)	異なるバージョンのインテル® コンパイラーや異なるコンパイラー間で OpenMP* を使用する注意点や制限について説明します
OpenMP* のタスク機能 (14:30 – 15:30)	OpenMP* 3.1 で追加されたタスク機能が 4.0 から 4.5 でどのように進化したかを例を使用して説明し、最新の OpenMP* 5.0 で強化された新機能を紹介します
OpenMP* の SIMD 機能 (13:30 – 14:30)	OpenMP* のスレッド化機能を使用してプログラマーがマルチスレッドの動作をプログラミングしたように、OpenMP* 4.0 からは omp simd を使用してプログラマーが明示的にベクトル化もできるようになりました。OpenMP* simd に関連する機能を 4.0 から 5.0 までの進化を追って紹介します
OpenMP* のオフロード機能 (14:30 – 15:30)	OpenMP* 4.0 で追加されたオフロード機能を利用することで、これまで共有メモリー型並列処理に加え分散メモリー型の並列処理を表現できるようになりました。このセッションでは、注目されるヘテロジニアス・プログラミング環境での OpenMP* オフロード機能について説明します
OpenMP* 5.0 の注目する機能	セッション2、3、4でカバーされなかった OpenMP* 5.0 のそのほかの機能について紹介します
インテル® C++/Fortran コンパイラーのバージョン 19.1 を使用して GPU オフロードに備えましょう	oneAPI 向けのデータ並列 C++ (DPC++) へ移行する前に、現行のインテル® C++/Fortran コンパイラー V19.1 やインテル® oneAPI HPC ツールキットに含まれるベータ版インテル® C++/Fortran コンパイラー 2021 を使用して簡単にインテル® グラフィックスへのオフロードを行うソフトウェアを開発および検証方法を紹介します

© 2020 iSUS *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。

内容

- はじめに (OpenMP* が必要とされる背景) と概要 (OpenMP* とは、歴史、各バージョンの機能概要)
- OpenMP* の各バージョンの機能 (4.0、4.5 および 5.0 の注目される新機能)
- 次世代インテル® コンパイラー (nextgen) の機能

OpenMP* 5.0 API シンタックス・クイック・リファレンス・カードの日本語訳を公開しました:

https://www.isus.jp/products/c-compilers/openmp-ref-5-0-0519-released/

© 2020 iSUS *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。

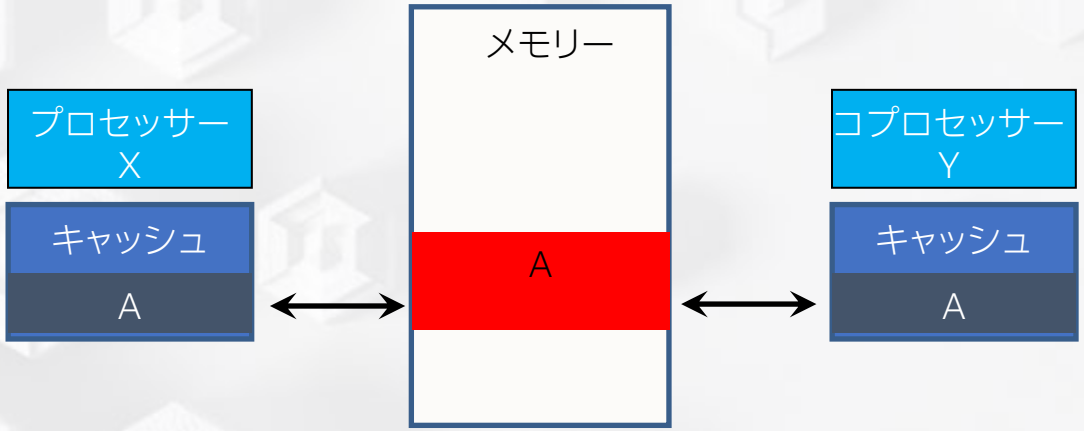
内容

- OpenMP* の各バージョンの機能
- OpenMP* 4.0 と 4.5、および 5.0 の新機能
  - タスク
  - SIMD
  - オフロード
  - OpenMP* 5.0 の注目する新機能

© 2020 iSUS *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。

データ共有/マッピング: 共有もしくは分散メモリー

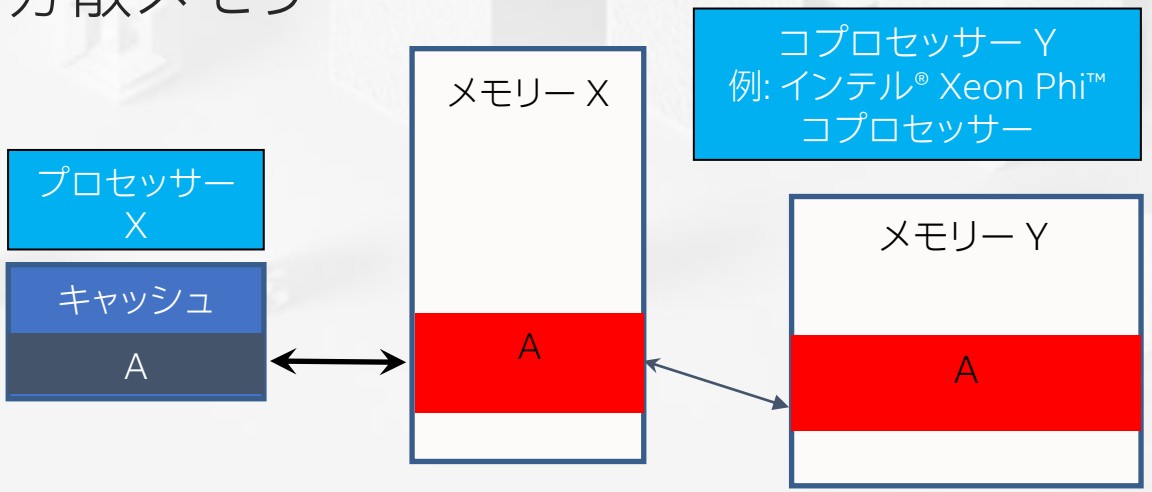
共有メモリー



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 (レジスター、キャッシュなど) の一時的なビューを保持できる

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・スタックを保持できる

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

© 2020 iSUS *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。

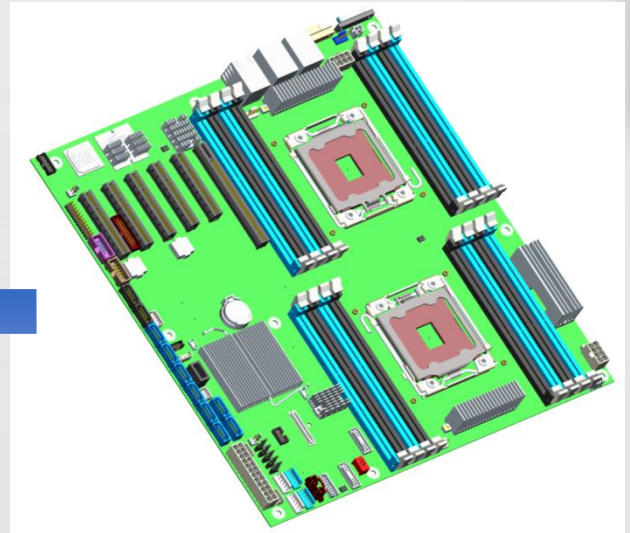
OpenMP* デバイスモデル

OpenMP* は、アクセラレーターとコプロセッサーをサポート

デバイスモデル:

- １つのホスト
- 同種の複数のアクセラレーター/コプロセッサー/GPGPU

プロセッサー/コプロセッサー/GFX/GPGPU



ホスト

© 2020 iSUS *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。

インテル® Xeon Phi™ コプロセッサーへオフロードする際の注意点

要件

インテル® コンパイラーのバージョン 19.0 以降、インテル® Xeon Phi™ コプロセッサーへのオフロードは未サポートとなりました

ls

© 2020 iSUS *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。

赤字が OpenMP* 4.5 での拡張

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

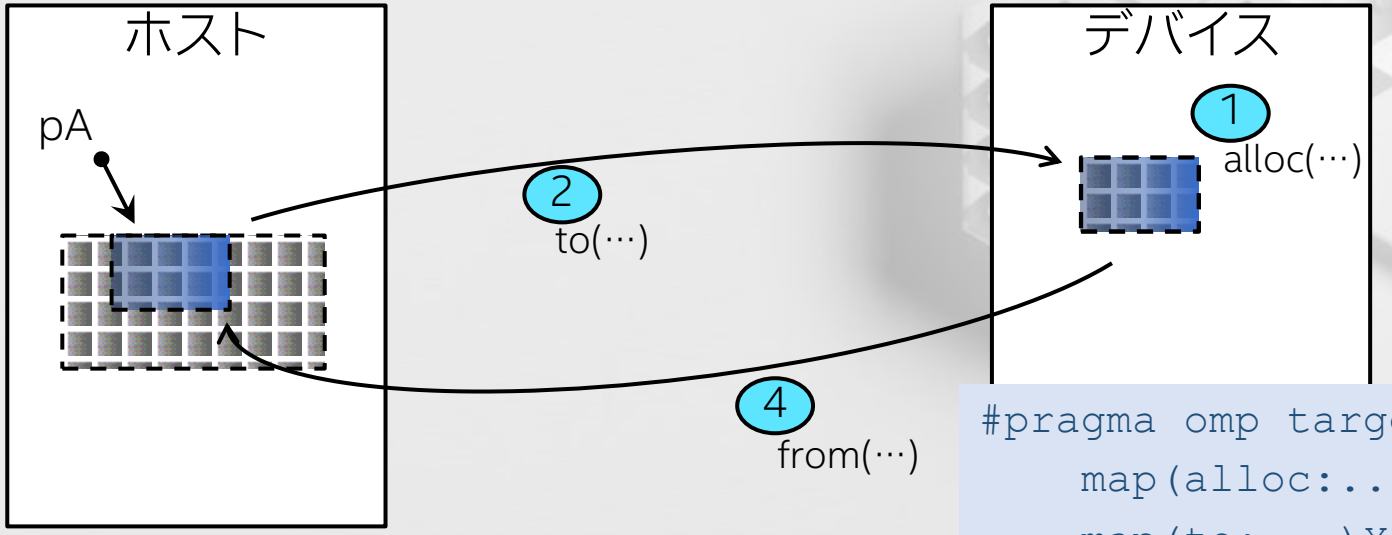
- OMP_DEFAULT_DEVICE を介してデフォルトデバイスを制御
- 負ではない整数値

© 2020 iSUS *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。

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




```
#pragma omp target ¥
    map(alloc:...)¥
    map(to:...)¥
    map(from:...)
{ ...}
```

3

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

© 2020 iSUS *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。

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);
}
```

グローバル変数がプログラム全体でデバイスデータ環境にマップされることを示す

ホストとデバイス間で一貫性を保つため target update を使用する

parallel for simd ループがターゲットへオフロードされることを示す

© 2020 iSUS *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。

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) を活用できます

```
#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();
  }
  #pragma omp taskwait
}
```

© 2020 iSUS *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。

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** 節が追加されました
taskwait でホストはターゲットの完了を待機します

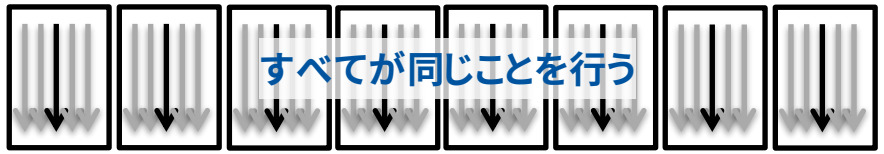
```
#pragma omp parallel sections
{
#pragma omp target map(in:input[:N]) map(out:result[:N]) nowait
  #pragma omp parallel for
      for (i=0; i<N; i++) {
        result[i] = some_computation(input[i], i);
      }
   // 以下をホストで非同期に実行
   do_something_important_on_host();
#pragma omp taskwait
}
```

ホスト
ターゲット
ホスト

© 2020 iSUS *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。

例: teams+parallel for (SAXPY – アクセラレーター向けコード)

```
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)
```




```
   for (int i = 0; i < n; i += num_blocks){
      for (int j = i; j < i + num_blocks; j++) {
           y[j] = a*x[j] + y[j];
  } }
  }
   free(x); free(y); return 0;
}
```

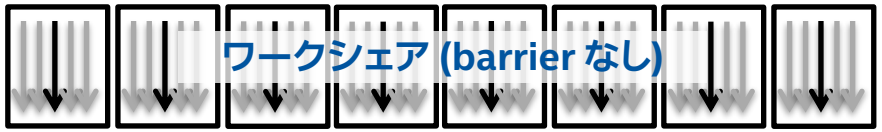
1 つ以上のループの反復を実行するスレッドチームを生成し、マスタースレッドで実行を開始します

© 2020 iSUS *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。

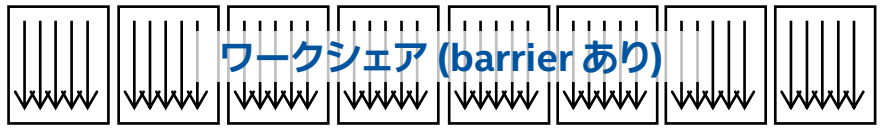
例: teams+parallel for (SAXPY – アクセラレーター向けコード)

```
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

© 2020 iSUS *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。

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(tofrom: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]);
  }
```

© 2020 iSUS *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。

OpenMP* と OpenACC* 対応例

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

© 2020 iSUS *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。

GFX へオフロードする際の注意点 (Windows*)

要件:

1. インテル® C++ コンパイラー 15.0 以降を入手する (17.0 まで …)

インテル® コンパイラーのバージョン 19.0 以降、インテル® Xeon Phi™ コプロセッサーへのオフロードは未サポートとなりました

http://www.isus.jp/products/psxe/getting-started-with-compute-offload-to-gfx/

© 2020 iSUS *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。

GFX へオフロードする際の注意点 (Linux*)

要件:

1. インテル® C++ コンパイラー 15.0 以降を入手する (17.0 まで…)
2. インテル® HD グラフィックス・ドライバー、またはオープンソース・メディア・カーネル・ランタイムを入手する
3. 「video」グループの権限を持つアカウントで、Linux* マシンにログインする
4. /usr/lib/x86_64-linux-gnu を LD_LIBRARY_PATH に追加する:
   export LD_LIBRARY_PATH=/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH
5. リンカーがライブラリーを検索できるようにするため、/etc/ld.so.conf.d/x86_64-linux-gnu.conf ファイルに /usr/lib/x86_64-linux-gnu パスを追加する

Linux* ドライバーのバージョンによる OS サポート:
http://www.isus.jp/products/psxe/getting-started-with-compute-offload-to-gfx/

© 2020 iSUS *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。

GFX へのオフロード: OpenMP* 4.0 offload への追加機能

```
bool Sobel::execute_offload()
{
            int w = COLOR_CHANNEL_NUM * image_width;
            float *outp = this->output;
            float *img = this->image;
            int iw = image_width;
            int ih = image_height;
#pragma omp target map(to: ih, iw, w) ¥
                        map(tofrom: img[0:iw*ih*COLOR_CHANNEL_NUM], ¥
                                                outp[0:iw*ih*COLOR_CHANNEL_NUM])
#pragma omp parallel for collapse(2)
     for (int i = 1; I < ih - 1; i++) {
          for (int k = COLOR_CHANNEL_NUM; k < (iw - 1) * COLOR_CHANNEL_NUM; k++) {
                        float gx = 1 * img[k + (i - 1) * w -1 * 4]
                                    + 2 * img[k + (i - 1) * w +0 * 4]
                                    + 1 * img[k + (i - 1) * w +1 * 4]
                                    - 1 * img[k + (i + 1) * w -1 * 4]
                                    - 2 * img[k + (i + 1) * w +0 * 4]
                                    - 1 * img[k + (i + 1) * w +1 * 4];
                        float gy = 1 * img[k + (i - 1) * w -1 * 4]
                                    - 1 * img[k + (i - 1) * w +1 * 4]
                                    + 2 * img[k + (i + 0) * w -1 * 4]
                                    - 2 * img[k + (i + 0) * w +1 * 4]
                                    + 1 * img[k + (i + 1) * w -1 * 4]
                                    - 1 * img[k + (i + 1) * w +1 * 4];
                        outp[i * w + k] = sqrtf(gx * gx + gy * gy) / 2.0;
               }
     }
     return true;
}
```

利用方法:

- OpenMP* の一部の機能のみをサポート
- 配列データはポインター渡し “tofrom” を “pin” へマップ
- GFX 向けのコンパイルを指示 “-qopenmp-offload=gfx” を指定

© 2020 iSUS *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。

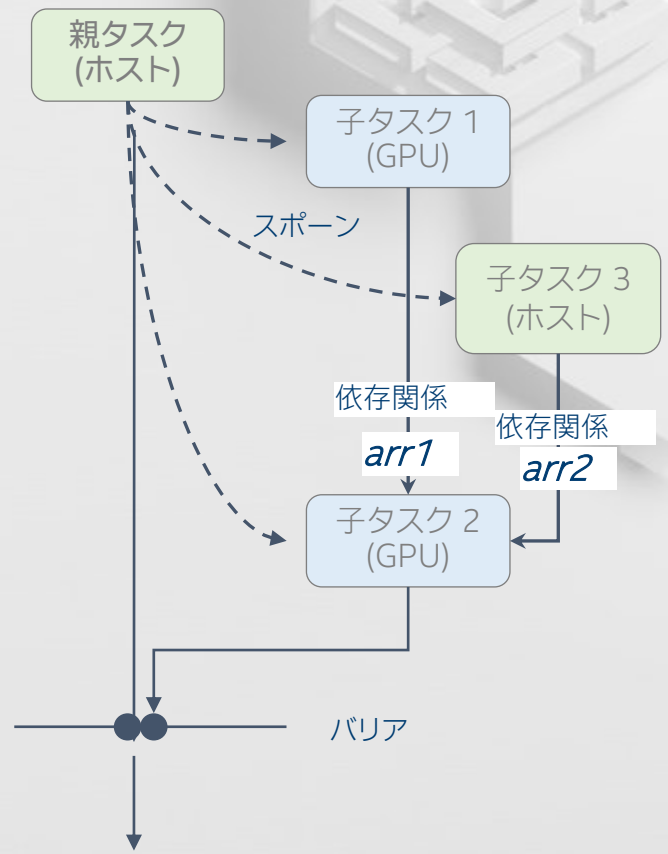
target ディレクティブで depend 節を使用して GPU へ非同期オフロードする例

```
// arr1 を初期化 - ターゲットへオフロード
#pragma omp target map(from: arr1[0:SIZE]) depend(out:arr1) nowait
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { arr1[i] +=  i; }

// arr2 の初期化
#pragma omp task depend(out:arr2)
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { arr2[i] += -i; }

// ターゲット上で中間結果を計算
#pragma omp target ¥
  map(to: arr1[0:SIZE], arr2[0:SIZE]) ¥
  map(from:arr3[0:SIZE]) ¥
  nowait depend(in:arr1, arr2)
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { arr3[i] = arr1[i] + arr2[i]; }

#pragma omp taskwait
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { res[i] += arr3[i]; }
```



© 2020 iSUS *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。

OpenMP* 4.5 Offload 拡張 (omp target)

OpenMP* 4.5 によるオフロード拡張への追加機能

- 主要機能は、インテル® コンパイラー 16.0 で実装済み

新機能:

- target 構文と API への新しいデバイスポインター節の追加

```
#pragma omp target data …  use_device_ptr(list) …
void omp_target_free(void * device_ptr, int device_num);
```

- link 節による遅延マップ

```
#pragma omp declare target …  [to] ( extended-list ) link (list)
```

- target 構文の節の多様性

```
#pragma omp declare target …  private(list) firstprivate(list) if (…)
```


© 2020 iSUS *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。

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 [2.15.5、2.15.7]
- 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 [2.15.7]
- 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 [2.15.7]
- 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 [2.15.8]
- 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 [2.20.6.1]
- 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 [3.2.36] 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 [5.17]

© 2020 iSUS *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。

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
- 以前は、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

© 2020 iSUS *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。

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

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

© 2020 iSUS *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。

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)
```

© 2020 iSUS *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。

続き

- データ属性
  - 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

© 2020 iSUS *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。

omp declare target 内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
      }
}
```

© 2020 iSUS *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。

入れ子になった declare 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

© 2020 iSUS *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。

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する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。ポインターフィールドが有効なデバイスメモリーを指すようにプログラマーが要求した場合、デバイス上にメモリーを確保してデバイスのポインターフィールドを明示的に更新する必要があります。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

© 2020 iSUS *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。

OpenMP* 5.0 のオフロード機能まとめ

- ヘテロジニアス・プログラミングが注目されています
- インテル® コンパイラーでは、バージョン 18.0 以降 OpenMP* オフロード機能のサポートに一貫性があるとは言えませんでした
- バージョン 19.1 以降はインテル® グラフィックスへの OpenMP* オフロードが復活しつつあります

詳細はセッション 6 で

Intel、インテル、Intel ロゴ、Xeon、Intel Xeon Phi 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。
© 2020 iSUS *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。

iSUS

ソフトウェア・セミナー