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 5: OpenMP* 5.0 の注目する機能

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 (13:30 – 14:30)	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 (13:40 – 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 (13:30 – 14:15)	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 (14:15 – 15:30)	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

iSUS © 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

iSUS © 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
(メモリー管理、アフィニティーなど)

© 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。

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

iSUS © 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。

新しい allocate ディレクティブの例

- 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たものです (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)

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

iSUS © 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: 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);
}
```

iSUS © 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。

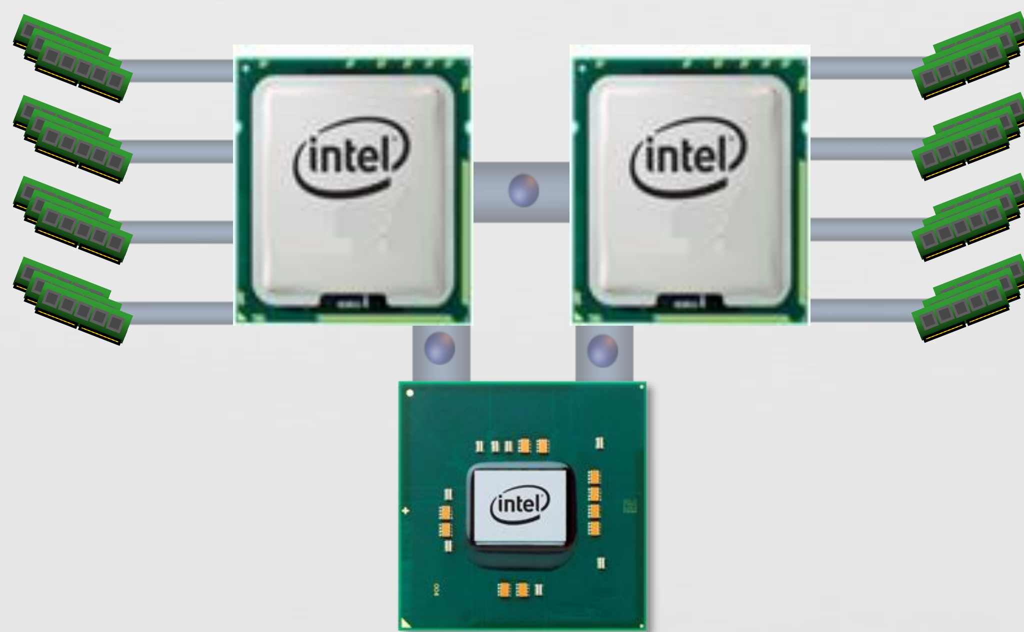
アフィニティー拡張

© 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。

NUMA はここに属する ...

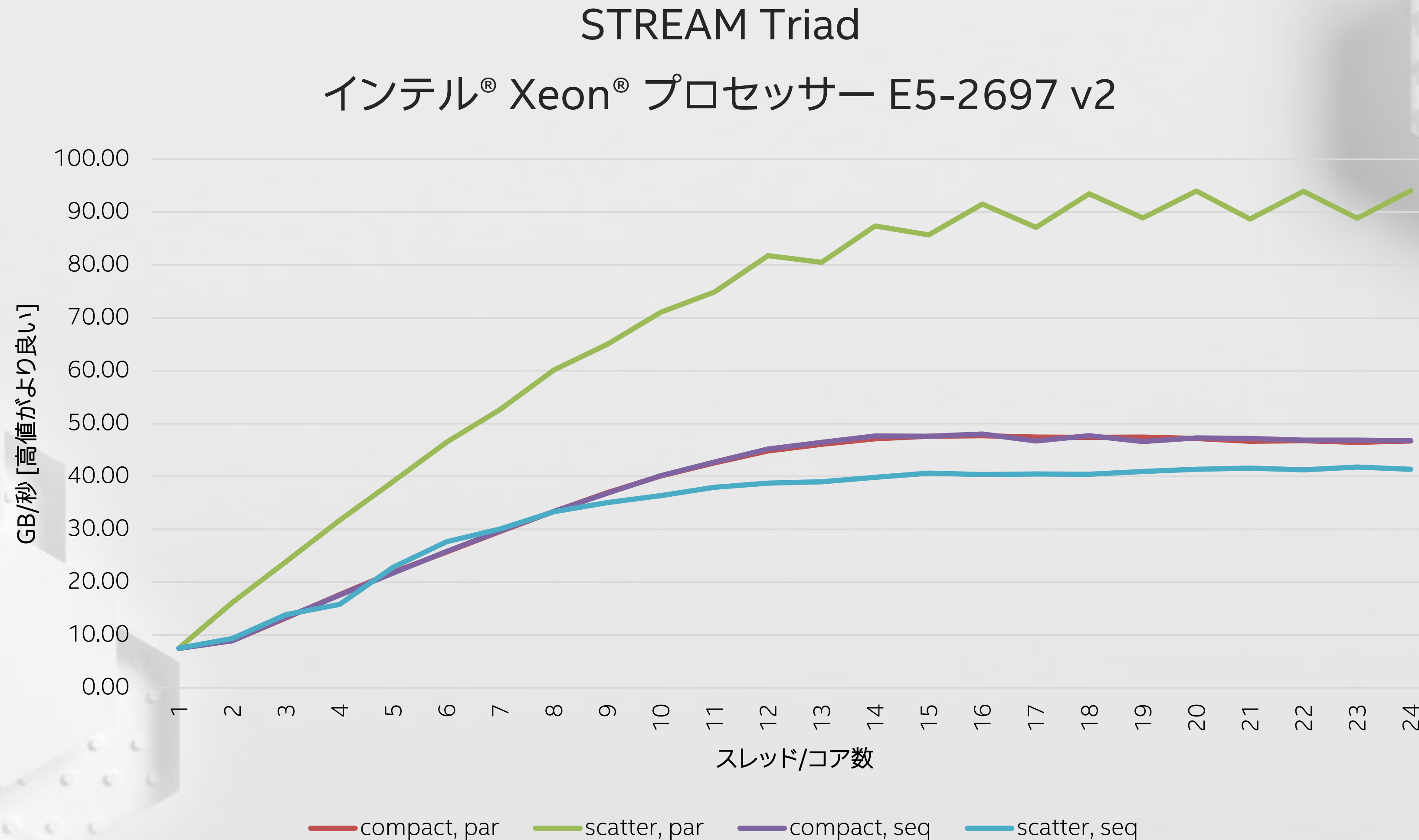
すべてのマルチソケット計算サーバーは、NUMA システム

- 異なるメモリー位置へのアクセス･レイテンシーは一定ではない
- 異なるメモリー位置の帯域幅は異なる可能性がある



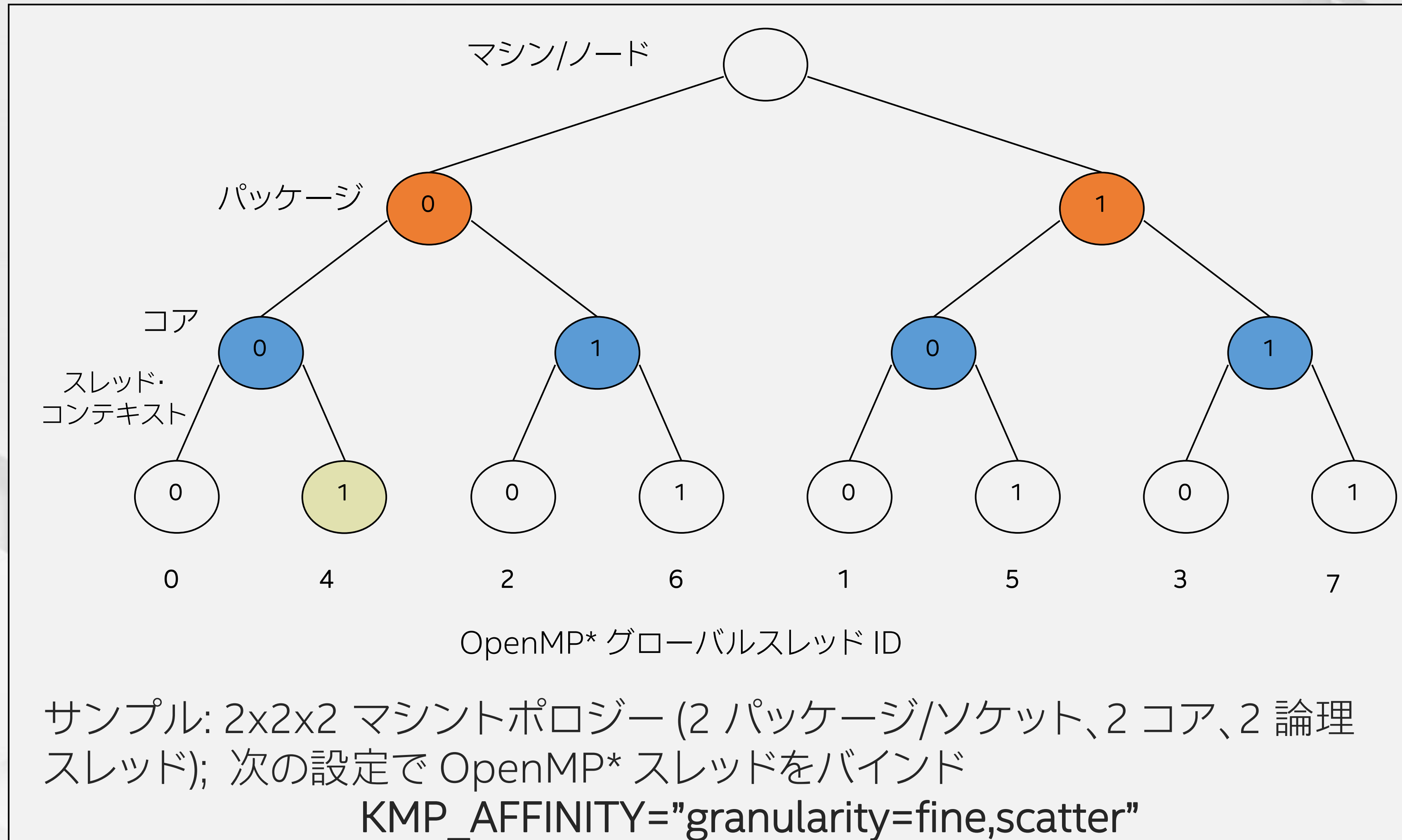
© 2020 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

スレッド・アフィニティー - なぜ重要なのか?



© 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。

KMP_AFFINITY: 柔軟性のある OMP スレッドの割り当て (インテル固有の実装)



© 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。

KMP_PLACE_THREADS (インテル固有の実装)

- スレッドの配置を制御
  - ソケット数
  - ソケットあたりのコア数
  - コアあたりのスレッド数
- KMP_PLACE_THREADS 変数は、プログラムで使用されるハードウェア・リソースを制御します。この環境変数は、使用するソケット数、各ソケットで使用するコア数、および各コアに割り当てるスレッド数を指定します
- 例えば、インテル® Xeon Phi™ コプロセッサーでは、各コプロセッサーの最大スレッド数は 4 であるが、コアあたり 3 スレッド以下にしてもパフォーマンスが向上することがある
- サンプル: KMP_PLACE_THREADS=2s,4c,2t

iSUS © 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 で標準化されたアフィニティー制御

parallel 領域向けの追加された節: proc_bind (アフィニティー･タイプ)

```
#pragma omp parallel for proc_bind(spread)
for (i=0; i<MAX; i++)
      A[i] += B[i] * C[i];
```

環境変数によるアフィニティー設定制御:

- OMP_PROC_BIND

  例えば: export OMP_PROC_BIND=“spread,spread,close,close”

- OMP_PLACES

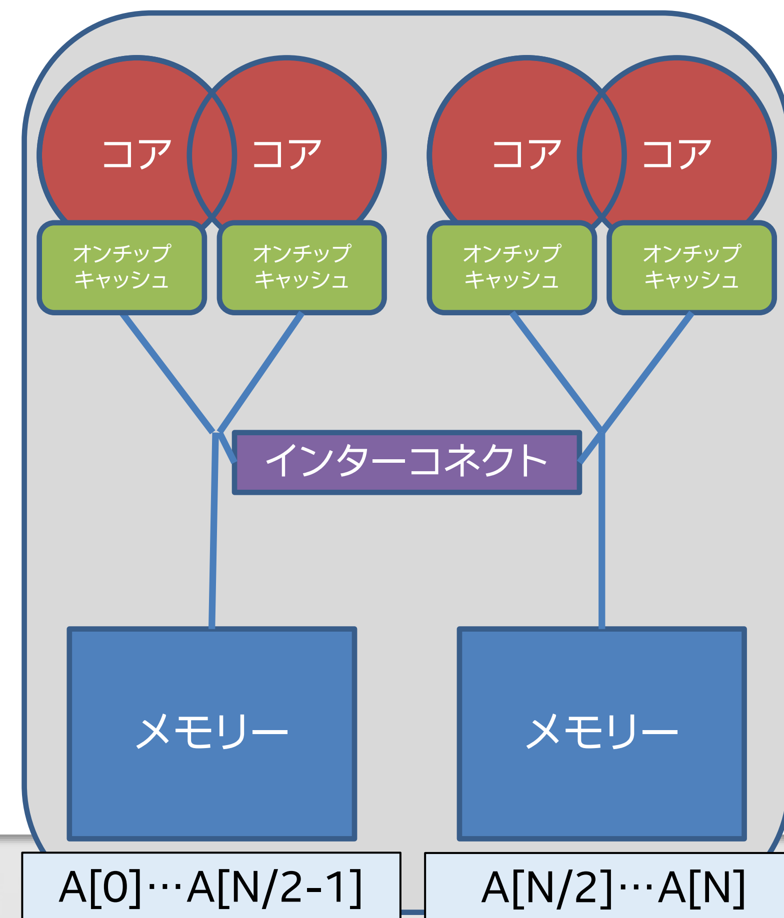
  例えば: export OMP_PLACES=“{0,1,2,3},{4,5,6,7},{8:4},{12:4}”

OMP_PLACES は実装依存で、OpenMP* では定義されない

iSUS © 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。

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);
}
```

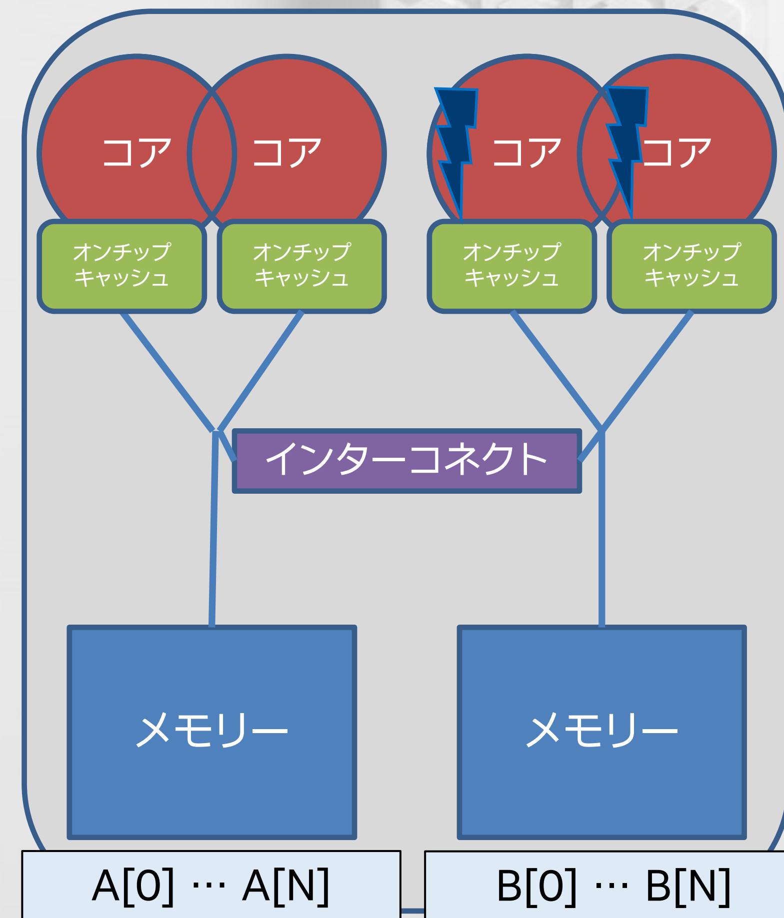


© 2020 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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
}
```

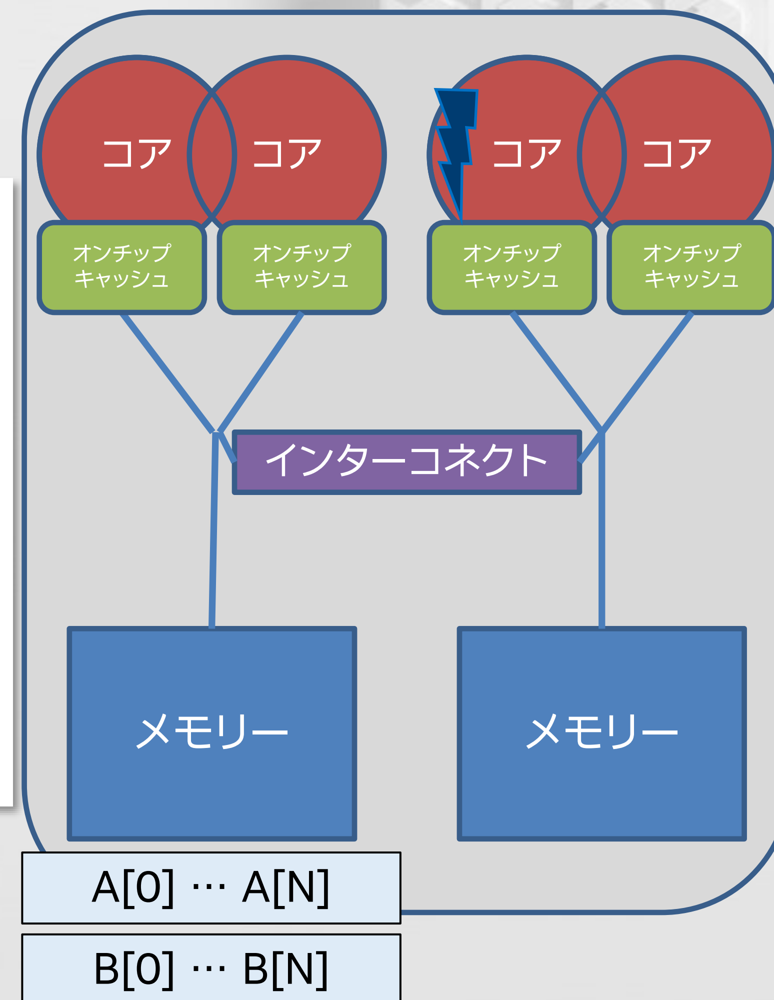


© 2020 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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
}
```



© 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。

タスクのリダクション

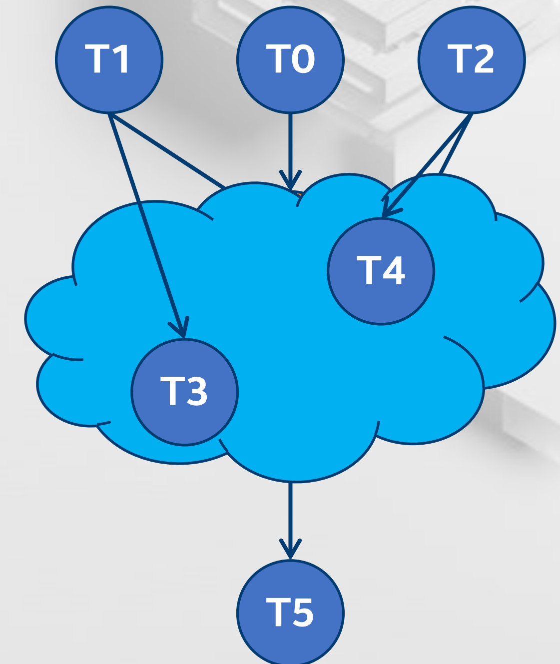
- タスク･リダクションは、従来のリダクションを任意のタスクグラフに拡張
- 既存の task と taskgroup 構造を拡張
- 同様に taskloop 構造でも動作

```
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;
         }
      }
   }
}
```


iSUS © 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。

タスクの依存性 (mutexinoutset)

```
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(mutexinoutset: res)  //T3
  res += x;

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

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



iSUS © 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* バージョン 5.0 その他の機能

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
メモリー・アロケーター
デタッチ可能なタスク
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

iSUS © 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。

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

iSUS
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。

ソフトウェア·セミナー