



新しい MIC Offload 機能



内容

- 非同期データ転送
- 非同期演算
- データ転送を伴わないメモリー管理
- C/C++ と Fortran で利用可能
 - スライド中の例は C/C++ のみ

2 つの新しいプラグマ: データ転送と完了の待機

データ転送のみ(オプションで非同期)の新しいプラグマ:

```
#pragma offload_transfer <節> [ signal(<タグ>) ]
```

非同期アクティビティの完了を待機する新しいプラグマ:

```
#pragma offload_wait <節> wait(<タグ>)
```

既存の offload プラグマはオプションの signal/wait 節を指定可能

```
#pragma offload <節> [ signal(<タグ>) ] [ wait(<タグ>) ]
```

<ステートメント>

新しいプラグマは単独で使用する

#pragma offload の後などには記述しない

#pragma offload_transfer

- ・このプラグマは単独で使用し、後にステートメントなどは持たない
- ・**target** 節および **in/out** 節を指定する
- ・**signal** 節が指定されない場合は同期データ転送を行う
- ・また **wait** 節を指定できる
- ・**signal** 節はデータ転送のみを開始する
 - 後述の **wait** 節を持つプラグマは、データ転送の完了を待機する

signal/wait 節の式は、非同期操作を行うタグとして提供されるアドレスサイズを持った値

```
// 例 1:  
// CPU -> MIC の同期データ転送  
// 以降のステートメントは、データ転送が完了してから実行される  
#pragma offload_transfer target(mic:0) in(a,b,c)  
// 例 2:  
// CPU -> MIC の非同期データ転送を開始  
#pragma offload_transfer target(mic:0) in(a,b,c) signal(&a)
```

#pragma offload_wait

- ・このプラグマは単独で使用し、後にステートメントなどは持たない
- ・**target** 節を指定する
- ・タグに関連した非同期のアクティビティーが完了した後に実行を開始する **wait** 節を含む

```
// 例 3:  
// シグナルされた &p のアクティビティーの完了を待機  
#pragma offload_wait target(mic:0) wait(&p)
```

MIC メモリーの割り当てと解放

- #pragma offload_transfer は、データ転送を伴わないメモリー領域の割り当てのみを行うこともできる
- 通常、メモリー割り当てのコストを償却するためループ外で使用

// 例 4:

```
#define ALLOC    alloc_if(1) free_if(0)
#define FREE     alloc_if(0) free_if(1)
#define REUSE    alloc_if(0) free_if(0)
```

// MIC 上のメモリーを割り当て(データ転送は行わない)

```
#pragma offload_transfer target(mic:0) nocopy(p,q : length(l) ALLOC)
```

...

```
    for (...) {
```

```
        // offload で MIC 上に割り当てられたメモリーを使用
```

```
#pragma offload target(mic:0) in(p:length(l) REUSE) out(q:length(l) REUSE)
```

```
    { // p と q を使用して演算 }
```

```
    }
```

...

// MIC 上のメモリーを解放(データ転送は行わない)

```
#pragma offload_transfer target(mic:0) nocopy(p,q : length(l) FREE)
```

入力データを非同期に送信

一般的にデータ転送と CPU の処理をオーバーラップし、後述の offload でそのデータを使用するために利用する

```
// 例 5:  
// CPU -> MIC の非同期データ転送を開始  
#pragma offload_transfer target(mic:0) in(p,q,r) signal(&p)  
...  
...  
// データ転送が完了した後 offload を開始  
#pragma offload target(mic:0) wait(&p)  
{  
// オフロード演算  
    ... = p;  
}
```

データは転送開始時に指定された変数と同じ場所に格納されている
それらの変数は、2番目のプラグマでアクセス可能である必要がある

出力データを非同期に受信

- ・offload で計算した結果を、後で転送する際に利用される
- ・offload プラグマは計算を終えたらデータの転送をスキップする
- ・非同期転送プラグマでデータの受信を開始
- ・受信したデータが必要であれば、`#pragma offload_wait` で待機

```
// 例 6:  
// offload 処理を終えても結果をすぐにはコピーしない  
#pragma offload target(mic:0) nocopy(p)  
{ p = ...; }  
  
// MIC -> CPU の非同期データ転送を開始  
#pragma offload_transfer target(mic:0) out(p) signal(&p)  
...  
...  
// データが転送されるのを待機  
#pragma offload_wait target(mic:0) wait(&p)
```


MIC での非同期計算

```
// 例 6
char signal_var;
int *p;
do {
    // 非同期計算を開始
    #pragma offload ... in( p:length(1000) ) signal(&signal_var)
    {
        mic_compute(); // MICで処理
    }
    concurrent_cpu_activity(); // CPU で処理
    #pragma offload_wait (&signal_var);
} while (1);
```

- CPU は、非同期でオフロードを行うことを指示
- CPU は、オフロード開始後次のステートメントを処理できる
- オフロードで計算したデータが必要であれば、#pragma offload_wait で待機

シグナルをテストする API

```
// 例 7:  
// 非同期計算を開始  
int c;  
#pragma offload target(mic:0) signal(&c) ...  
  { S3; }  
...  
// 計算が終わったか確認  
if _Offload_signaled(0, &c) ....
```

- ・“c” でシグナルされている計算が終わったかを確認
- ・オフロードが完了したかを確認する非ブロッキングのメカニズム

例 8: オフロードの繰り返し

```
void do_sync()
{
int i;
    for (i=0; i<iter; i++)
    {
        #pragma offload target(mic) ¥
            in(in1 : length(count) REUSE ) ¥
            out(out1 : length(count) REUSE )
        compute(in1, out1);
    }
}
```

例 8: ダブルバッファ入力

```
void do_async_in() {
int i;
#pragma offload_transfer target(mic:0) in(in1 : length(count) REUSE) signal(in1)
  for (i=0; i<iter; i++)
  {
    if (i%2 == 0) {
      #pragma offload_transfer target(mic:0) if(i!=iter-1) ¥
        in(in2 : length(count) REUSE) signal(in2)
      #pragma offload target(mic:0) nocopy(in1) wait(in1) ¥
        out(out1 : length(count) REUSE)
      compute(in1, out1);
    } else {
      #pragma offload_transfer target(mic:0) if(i!=iter-1)
        in(in1 : length(count) REUSE ) signal(in1)
      #pragma offload target(mic:0) nocopy(in2) wait(in2) ¥
        out(out2 : length(count) REUSE)
      compute(in2, out2);
    }
  }
}
```

例 8: ダブルバッファー出力

```
void do_async_out() {
int i;

for (i=0; i<iter+1; i++) {
    if (i%2 == 0) {
        if (i<iter) {
            #pragma offload target(mic:0) in(in1 : length(count) REUSE) nocopy(out1)
                compute(in1, out1);
            #pragma offload_transfer target(mic:0) out(out1:length(count) REUSE) signal(out1)
        }
        if (i>0) {
            #pragma offload_wait target(mic:0) wait(out2)
                use_result(out2);
        }
    } else {
        if (i<iter) {
            #pragma offload target(mic:0) in(in2 : length(count) REUSE) nocopy(out2)
                compute(in2, out2);
            #pragma offload_transfer target(mic:0) out(out2:length(count) REUSE) signal(out2)
        }
        if (i>0) {
            #pragma offload_wait target(mic:0) wait(out1)
                use_result(out1);
        }
    }
}
}
```

まとめ

- 非同期オフロードはデータ転送と計算のオーバーラップを可能にする
- 追加の CPU スレッドを使用しない
- パイプライン処理に有効

法務上の注意書きと最適化に関する注意事項

本資料の情報は、現状のまま提供され、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。

性能に関するテストや評価は、特定のコンピューター・システム、コンポーネント、またはそれらを組み合わせて行ったものであり、このテストによるインテル製品の性能の概算の値を表しているものです。システム・ハードウェアの設計、ソフトウェア、構成などの違いにより、実際の性能は掲載された性能テストや評価とは異なる場合があります。システムやコンポーネントの購入を検討される場合は、ほかの情報も参考にして、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。インテル製品の性能評価についてさらに詳しい情報をお知りになりたい場合は、http://www.intel.co.jp/jp/performance/resources/benchmark_limitations.htm を参照してください。

Intel、インテル、Intel ロゴ、Intel Core、Xeon、Cilk、VTune は、アメリカ合衆国およびその他の国における Intel Corporation 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® コンパイラーは、互換マイクロプロセッサ向けには、インテル製マイクロプロセッサ向けと同等レベルの最適化が行われられない可能性があります。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® SSE2)、インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3)、ストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。インテルでは、インテル製ではないマイクロプロセッサに対して、最適化の提供、機能、効果を保証していません。本製品のマイクロプロセッサ固有の最適化は、インテル製マイクロプロセッサでの使用を目的としています。インテル® マイクロアーキテクチャーに非固有の特定の最適化は、インテル製マイクロプロセッサ向けに予約されています。この注意事項の適用対象である特定の命令セットの詳細は、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。

改訂 #20110804

