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 Fortran のみ

2 つの新しい宣言子: 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）の新しい宣言子:

!DIR$ OFFLOAD_TRANSFER <節> [signal<タグ>]

非同期アクティビティーの完了を待機する新しい宣言子:

!DIR$ OFFLOAD_WAIT <節> [wait<タグ>]

既存の offload 宣言子はオプションの signal/wait 節を指定可能

!DIR$ OFFLOAD <節> [signal<タグ>] [wait<タグ>]

<ステートメント>

新しい宣言子は単独で使用する

！DIR$ OFFLOAD など後にステートメントを記述しない

!DIR$ OFFLOAD_TRANSFER

・この宣言子は単独で使用し、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 節を持つ宣言子は、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
!DIR$ 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
!DIR$ OFFLOAD_TRANSFER target(mic:0) in(a,b,c) signal(s)
```

!DIR$ OFFLOAD_WAIT

- この宣言子は単独で使用し、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
!DIR$ OFFLOAD_TRANSFER target(mic:0) wait(s)
```

MIC メモリーの割り当てと解放

- !DIR$ 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(.TRUE.) free_if(.FALSE.)
#define FREE     alloc_if(.FALSE.) free_if(.TRUE.)
#define REUSE    alloc_if(.FALSE.) free_if(.FALSE.)

!MIC 上のメモリーを割り当て(データ転送は行わない)
!DIR$ OFFLOAD_TRANSFER target(mic:0) nocopy(p,q : ALLOC)
  ...
          do ...
          !offload で MIC 上に割り当てられたメモリーを使用
          !DIR$ OFFLOAD target(mic:0) in(p: REUSE) out(q: REUSE)
          !p と q  を使用して演算  ...
          enddo
  ...
!MIC 上のメモリーを解放(データ転送は行わない)
!DIR$ OFFLOAD_TRANSFER target(mic:0) nocopy(p,q :  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
!DIR$ OFFLOAD_TRANSFER target(mic:0) in(p,q,r) signal(s)
...
...
!データ転送が完了した後 offload を開始
!DIR$ OFFLOAD target(mic:0) wait(s)
!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番目の宣言子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 宣言子は計算を終えたらデータの転送をスキップする
- 非同期転送宣言子でデータの受信を開始
- 受信したデータが必要であれば、!DIR$ 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
!DIR$ OFFLOAD target(mic:0) nocopy(p)
! オフロード計算
   ...

!MIC -> CPU の非同期データ転送を開始
!DIR$ OFFLOAD_TRANSFER target(mic:0) out(p) signal(s)
...
...
!データが転送されるのを待機
!DIR$ OFFLOAD_WAIT target(mic:0) wait(s)
```

MIC での非同期計算

```
!例 6
character :: signal_var
integer, allocatable, dimension(:) :: p
  do ...
        // 非同期計算を開始
        !DIR$ OFFLOAD ... in(p) signal(signal_var)
        call mic_compute()  ! MICで処理

        call concurrent_cpu_activity()  ! CPU で処理
        !DIR$ OFFLOAD_WAIT (signal_var)
  enddo
```

- 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、!DIR$ OFFLOAD_WAIT で待機

シグナルをテストする API

```
program prog
   use mic_lib
   implicit none
...
!例 7:
!非同期計算を開始
integer :: c
!DIR$ OFFLOAD target(mic:0) signal(c) ...
          !オフロード計算
          ...
// 計算が終わったか確認
if (Offload_signaled(0, c) /= 0 ) then
          ....
endif
```

- “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

```
! 例 8
subroutine  do_sync()
integer :: i
        do i=1, iter
                !DIR$ OFFLOAD target(mic) &
                        in(in1 :  REUSE ) &
                        out(out1 :  REUSE )
                call compute(in1, out1)
        enddo
end subroutine
```

例 8: ダブルバッファー入力

```
subroutine do_async_in()
Integer ::  i
!DIR$ OFFLOAD_TRANSFER target(mic:0) in(in1 : REUSE) signal(sig1)
    do i=0, iter
          if ( MOD(I, 2) == 1) then
          !DIR$ OFFLOAD_TRANSFER target(mic:0) if(i/=iter) &
                    in(in2 : REUSE) signal(sig2)
          !DIR$ OFFLOAD target(mic:0) nocopy(in1) wait(sig1) out(out1 : REUSE)
          call compute(in1, out1)
          else
           !DIR$ OFFLOAD_TRANSFER target(mic:0) if(i/=iter) &
                    in(in1 : REUSE ) signal(sig1)
           !DIR$ OFFLOAD target(mic:0) nocopy(in2) wait(sig2) out(out2 : REUSE)
          call compute(in2, out2)
          endif
    enddo
end subroutine
```

例 8: ダブルバッファー出力

```
subroutine do_async_out()
integer :: i
    do i=0, iter
            if ( MOD(I, 2) == 1) then !奇数
                if (i<iter) then
                !DIR$ OFFLOAD target(mic:0) in(in1 : REUSE) nocopy(out1)
                        call compute(in1, out1)
                !DIR$ OFFLOAD_TRANSFER target(mic:0) out(out1 : REUSE) signal(sig1)
                endif
                if (i>1) then
                !DIR$ OFFLOAD_WAIT target(mic:0) wait(sig2)
                        call use_result(out2)
                endif
            else
                if (i<iter) then
                !DIR$ OFFLOAD target(mic:0) in(in2 : REUSE) nocopy(sig2)
                        call compute(in2, out2)
                !DIR$ OFFLOAD_TRANSFER target(mic:0) out(out2 : REUSE)) signal(sig2)
                endif
                if (i>0) then
                !DIR$ OFFLOAD_WAIT target(mic:0) wait(sig1)
                        call use_result(out1)
                endif
            endif
    enddo
end subroutine
```

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。

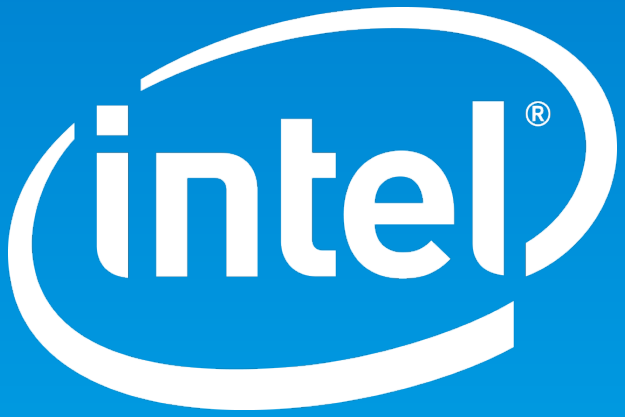
性能に関するテストや評価は、特定のコンピューター・システム、コンポーネント、またはそれらを組み合わせて行ったものであり、このテストによるインテル製品の性能の概算の値を表しているものです。システム・ハードウェアの設計、ソフトウェア、構成などの違いにより、実際の性能は掲載された性能テストや評価とは異なる場合があります。システムやコンポーネントの購入を検討される場合は、ほかの情報も参考にして、パフォーマンスを総合的に評価することをお勧めします。インテル製品の性能評価についてさらに詳しい情報をお知りになりたい場合は、http://www.intel.co.jp/jp/performance/resources/benchmark_limitations.htm 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ー向けには、インテル製マイクロプロセッサー向けと同等レベルの最適化が行わ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ーに対して、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ー固有の最適化は、インテル製マイクロプロセッサー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ー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


intel