intel
Software

インテル® コンパイラー 19.0 の新機能

Xiaoping Duan

テクニカル・コンサルティング・エンジニア

インテル コーポレーション

内容

オプションの変更と IDE 統合サポートの拡張

インテル® C++ および Fortran コンパイラー 19.0 の共通の新機能

インテル® C++ コンパイラー 19.0 の新機能

インテル® Fortran コンパイラー 19.0 の新機能

デバッグ情報の拡張

オプションの変更と IDE 統合サポートの拡張

オプションの変更

1. -m<feature>: プロセッサー機能を細かく制御
2. -qopenmp-simd (Linux*、macOS*)、/Qopenmp-simd (Windows*) はデフォルトで有効
3. -fcf-protection[=*keyword*] (Linux*)
   - 既存の -qcf-protection のエイリアス
   - コマンドラインで -mibt または -mshstk を指定する必要がある
4. -[Q]rcd は廃止予定の非推奨のオプション

開発コード名によるターゲット・プロセッサーの指定

ターゲット・プロセッサーの指定オプションが機能セット名に加えてプロセッサー開発コード名に対応

- -mtune、-march、-arch、-x、-ax (および Windows* の等価なオプション) で利用可能
- サポートされるプロセッサー開発コード名はコマンドライン・ヘルプで確認
- 将来のプロセッサー開発コード名 Cannonlake および Icelake をサポート

例:

Windows*:

/Q[a]xcannonlake、/Q[a]xhaswell、/tune=cannonlake、/tune=skylake

Linux*/macOS*:

-xcannonlake、-xknl、-march/-mtune=skylake-avx512

IDE 統合サポートの拡張

- Microsoft* Visual Studio* で状況依存ヘルプが復活。必要な追加ステップの詳細はこちら (英語) を参照
- Visual Studio* 2017 Build Tools をサポート
  - C++: Microsoft* Visual Studio* 2017 IDE をインストールしていない compilervars 環境と MSBuild を使用する Visual Studio* C++ プロジェクト
  - Fortran: Microsoft* Visual Studio* 2017 IDE をインストールしていない compilervars 環境
- Fortran エディターの拡張
  - クラスビューにインターフェイス・ジェネリック名を追加
  - 認識される構造: MODULE SUBROUTINE、MODULE FUNCTION、MODULE PROCEDURE
- Eclipse* でアセンブリー・ファイルを処理可能

intel
Software

インテル® C++ および Fortran コンパイラー 19.0 の共通の新機能

新しい動的アライメント節

構文:

C/C++

```
#pragma vector dynamic_align[(pointer)]
#pragma vector nodynamic_align
```

Fortran

```
!DIR$ vector dynamic_align[(var)]
!DIR$ vector nodynamic_align
```

セマンティクス:

コンパイラーは指定されたポインターに対してピールループを生成する

ポインターが指定されていない場合、コンパイラーはアライメントされたロードとストアを生成するポインターを自動的に決定するか、ピールループを生成しない

nodynamic_align 節を指定すると、コンパイラーはピールループを生成しない

新しいベクトル長節

構文:

C/C++

```
#pragma vector vectorlength(vl1,vl2, ... , vln)
```

Fortran

```
!DIR$ vector vectorlength(vl1,vl2, ... , vln)
```

セマンティクス:

ベクトライザーはコストモデルに応じてリストから最適なベクトル長を選択する

リストのすべてのベクトル長が適していない場合、ループはスカラーのままとなる

#pragma omp simd simdlen とは異なり、このプラグマ/ディレクティブはベクトル化を強制しないため、すべてのループに安全に使用できる

規則:

$vl_i (\geq 2)$ は 2 の累乗

Software

インテル® C++ コンパイラー 19.0 の新機能

インテル® C++ コンパイラー 19.0 の新機能

1. 新しい C++17 標準機能++
   - インライン変数 (英語)+
   - if 文と switch 文の変数と条件を分離 (英語)+
   - 列挙型クラスの構文規則 (英語)+
   - 畳み込み式 (英語)
   - 例外仕様を型システムの一部にする (英語)
   - constexpr ラムダ式 (英語)
   - ***this** のラムダ・キャプチャー (英語)
   - if constexpr 文 (英語)
   - 構造体のバインド (英語)
   - 廃止予定の動的例外仕様を削除 (英語)

+ 18.0 Update 1 以降で利用可能

++最新情報は、「インテル® C++ コンパイラーでサポートされる C++17 の機能」を参照

2. 明示的な包括/排他スキャン構文
3. インダクション節とユーザー定義のインダクション
4. インテル® C++ コンパイラー for Windows* で GNU* 形式のインライン・アセンブリーをサポート

包括スキャンと排他スキャン

構文:

C/C++

```
#pragma omp simd reduction [parallel] (inscan, scan-op: item-list)
#pragma omp scan inclusive(item-list)
#pragma omp scan exclusive(item-list)
```

Fortran (未実装)

包括スキャン:

```
x = 0;
#pragma omp simd reduction (inscan, +: x)
for (i = 0; i < n; ++i) {
    x += A[i];
#pragma omp scan inclusive(x)
    B[i] = x;
}
```

排他スキャン:

```
x = 0;
#pragma omp simd reduction (inscan, +: x)
for (i = 0; i < n; ++i) {
    B[i] = x;
#pragma omp scan exclusive(x)
    x += A[i];
}
```

インダクション節とユーザー定義のインダクション

構文 (インテル® コンパイラーの拡張):

C/C++

```
induction( induction-id : list : step )
#pragma omp declare induction ( induction-id : induction-type : step-type : inductor ) [collector( collector )]
```

Fortran (未実装)

```
class A;   // インダクション変数のクラス
class S;   // ステップ式のクラス

#pragma omp declare induction ( op : A : S :                                ¥
                                        omp_out = omp_out + omp_step )  ¥
                                collector ( omp_step = omp_step * omp_index )
    // A←A+S と S←S*int を定義する必要がある
...
A a; S s;  // コンストラクターによって初期化される
...
#pragma omp simd[parallel] induction( op : a : s )
for(int i=0; i<N; i++) { work(a); a=a+s; }
```

operator+ と operator* のベクトル
バージョンがあったほうがよい

追加の GNU* asm キーワード (Windows*)

```
__asm__("nop" :); // 18.0 でサポート
__asm("nop" :);
_asm("nop" :);
asm("nop" :);
__asm { nop };
_asm { nop };
asm{ nop };
```

インテル® Fortran コンパイラー 19.0 の新機能

インテル® Fortran コンパイラー 19.0 の新機能

1. Fortran 2018 暫定版の機能
   - COSHAPE 組込み関数
   - IMPORT 文の拡張
   - Co-Array イベント
   - C_F_POINTER を除く ISO_C_BINDING のすべてのプロシージャーを PURE に変更
   - PUBLIC と PRIVATE 文で MODULE 名を使用可能
2. 配列の形状チェック
3. OpenMP* 並列領域内の BLOCK 構文
4. OpenMP* ユーザー定義リダクション

Fortran 2018 - COSHAPE 組込み関数

COSHAPE (COARRAY [, KIND])

結果は、Co-Array の corank と同じサイズのランク 1 の整数配列。KIND が指定されている場合、結果配列は指定された KIND となり、そうでない場合はデフォルトの整数 KIND となる。結果配列の各要素には、Co-Array の対応する codimension のサイズが格納されている

```
integer(kind=8) :: a[3,2:4,*] ! num_images=10 を仮定
integer :: c(3)
c = coshape(a,kind=8)        ! COSHAPE
if (this_image()==1) then
        print *,c  ! c は (/3,3,2/) になる
endif
```

Fortran 2018 - IMPORT 文の拡張

IMPORT 文はホストサブプログラムまたは BLOCK 構文で使用可能。IMPORT では ONLY、NONE、および ALL 節を使用できる

```
module m1
integer, parameter :: N = 4
contains
subroutine m_sub1
 import
 integer :: N = 1   ! N は m_sub1 に対してローカル
end subroutine m_sub1
subroutine m_sub2
 import, all
 integer N  ! エラー: N はローカルに再宣言できない
end subroutine
subroutine e_sub(N)
 import, none
 do i=1,N   ! N がローカルに宣言されるように強制
 ...
end subroutine
end module
```

Fortran 2018 - Co-Array イベント

新しいイベント機能は異なるイメージ上のプログラム実行セグメントを順序付けるのに役立つ

```
program main
use iso_fortran_env
implicit none
type(event_type) :: event1[*]
integer :: count
if (THIS_IMAGE() == 1) then
call event_query(event1, count)
if (count > 0) then ! 通知があったため待機の必要なし
print *, “Event 1 succeeded, no wait”
else
print *, “Waiting for Event 1”
event wait(event1, until_count=1)  ! event1 の count が >= 1 になるまで待機
print *, “Event 1 succeeded”
else if (THIS_IMAGE() == 2) then
event post(event1[1])
end if
end program main
```

Fortran 2018 – C_F_POINTER を除く ISO_C_BINDING のすべてのプロシージャーを PURE に変更

- C_F_POINTER を除く ISO_C_BINDING のすべてのプロシージャーを PURE に変更
- すべての PURE でない動作を抑制することが目的
- すべての標準組込み関数は PURE

Fortran 2018 - PUBLIC と PRIVATE 文で MODULE 名を使用可能

- 使用モジュールで宣言されているエンティティーのデフォルトのアクセシビリティーとは別に、USE に関連するエンティティーのデフォルトのアクセシビリティーを変更可能
- USE 文で使用されている MODULE 名を PUBLIC または PRIVATE 文で使用できる。モジュール名はスコープ単位内のすべてのアクセシビリティー文で最大 1 回使用可能

```
module m1
integer x1, x2
end module
module m2
 use m1! この文がないと private m1 はエラーになる
 private m1
 public x2 ! x1 はプライベートだが x2 を再エクスポート
    ...
end module
```

配列の形状チェック

- /check:shape と /warn:shape (Windows*)、-check shape と –warn shape (Linux*、macOS*) オプションは、代入時に配列の形状が一致しない場合にエラーまたは警告を出力する
- 代入 (source= 構文による割り当ての暗黙の代入) のみチェックされる

```
real(8) :: a(20)=1,b(20)=2
integer :: n=10,m=20
a(1:10)=3
b(1:m)=b(1:m)+a(1:n) ! /check:shape でエラーとして検出される
```

OpenMP* 領域内の BLOCK 構文

BLOCK/ENDBLOCK 構文は Fortran 2008 標準で追加され、インテル® Fortran コンパイラー 19.0 以降では OpenMP* 並列領域内での使用がサポートされたことでスレッドセーフなコードの記述が容易に

```
!$omp parallel shared (x)       ! この 'x' は共有
    block
        real :: x = 2.          ! この 'x' はプライベート (ブロックスコープ)
        real :: in_block        ! この 'in_block' はプライベート
        in_block = x            ! この 'x' はプライベート
    end block
!$omp end parallel
```

OpenMP* ユーザー定義リダクション

OpenMP* 4.5 構文:

C/C++ (インテル® C++ コンパイラー 16.0 以降でサポート)

```
#pragma omp declare reduction(reduction-identifier:typename-list:combiner)
[initializer-clause] new-line
```

Fortran

```
!$omp declare reduction(reduction-identifier: type-list : combiner)
[initializer-clause]
```

セマンティクス:

ユーザー定義のリダクションを指定

規則:

OpenMP* 4.5 仕様を参照

OpenMP* 4.5 を完全サポート!

OpenMP* ユーザー定義リダクションの例

```
type my_type
 integer :: component  = 4
end type

!$omp declare reduction (+ : my_type : omp_out = omp_out + omp_in) initializer (omp_priv=my_type (0))

type(my_type) :: my_var    !my_type に対して定義されている演算子を仮定

!$omp parallel reduction (+ : my_var) num_threads(4)
my_var%component = omp_get_thread_num() + 1
!$omp end parallel

print *, "sum of thread numbers is ", my_var%component ! 14
```

デバッグ情報の拡張

デバッグ情報の拡張

追加のデバッグ情報

Fortran の名前付き BLOCK 構文

-g は -O0 であってもデフォルトで -debug inline-debug-info を設定

- -O0 の手動インライン展開に関するデバッグ情報を生成
- 以前のバージョンでは -g と -O2 以降でのみ設定

関連情報

- リリースノート: 最新情報は最新のリリースノートを確認のこと
  - インテル® C++ コンパイラー 19.0 リリースノート (英語)
  - インテル® Fortran コンパイラー 19.0 リリースノート (英語)
- オンライン・サービス・センター (英語)

法務上の注意書きと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。 詳細については、www.intel.com/benchmarks (英語) を参照してください。

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も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。

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