インテル® コンパイラー 15.0

インテル® C++/Fortran コンパイラー for Linux*、OS X*、Windows*

はじめに

大きな可能性: コンピューター・アーキテクチャーに対するさまざまなパフォーマンス拡張により、ソフトウェア開発者の生産性を高めるチャンスが到来

- コア数、ベクトル幅、コプロセッサーの増加は高いパフォーマンスを提供
- 現在および将来もこれらの利点を活用するために C/C++ および Fortran 開発者ができることは?

解決方法: インテル® C++/Fortran コンパイラーを使用する

- インテル® コンパイラーはハイパフォーマンス・アプリケーションの開発者を支援する強力なベクトル化と並列モデルを提供
- これらのモデルは現在のインテル® プロセッサー・ベースのシステムをサポート
- そして、より多いコア数、より広いベクトル幅、コプロセッサーなどの進化に伴い将来もスケーリング

インテル® C++/Fortran コンパイラー 15.0

アプリケーション・パフォーマンスを向上する効率良い言語レベルの並列モデルを提供

- 共通機能
  - 新しい OpenMP* 4.0 のベクトル化により、SIMD 命令を利用してインテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ コプロセッサーで優れたパフォーマンスを実現
  - 強化されたコンパイラーの最適化レポートにより最適化の可能性を素早く特定 (Windows* では Visual Studio* 2010、2012、2013 に統合)
  - Linux*、OS X*、Windows*、Android* に対応
  - 開発ニーズに合わせてさまざまな設定が可能: C++ の詳細、Fortran の詳細
- インテル® C++ コンパイラー
  - インテル® Cilk™ Plus の並列化キーワードにより、タスクとデータ並列処理を簡単に実装
  - C++11 をフルサポート
- インテル® Fortran コンパイラー
  - 最新の Fortran 標準をサポート
  - ローグウェーブ IMSL* Fortran 数値ライブラリー: Windows* Fortran スイート向けのパフォーマンス・アドオン

ベクトル化の向上
インテル® コンパイラー

- ベクトル化の妨げとなるコードを切り離し、SIMD ループと SIMD 対応関数全体をベクトル化
- 部分的なベクトル化の適用
  - 妨げとなるコード領域 (多くの場合は小さな領域) をシリアル実行
  - 文レベルでも、表現レベルでも適用可能
  - 現在、SIMD ループと SIMD 対応 (ベクトル) 関数に対応
- 「文はベクトル化できません」、「操作はベクトル化できません」、「サポートされていないデータ型です」などのあいまいな理由によって SIMD ベクトル化されないケースが大幅に減少
- これによりユーザーの満足度が向上

簡単に利用できる最適化レポート
インテル® コンパイラー

分かりやすいメッセージ

- 関数名、データ変数、制御構造への参照

次のステップをアドバイス

- 例: 動作を変更するためのオプション、プラグマ、節など

```
int size();
 void foo(double *restrict
a, double *b){
  int i;
  for (i=0;i<size();i++){
    a[i] += b[i];
  }
}
```

```
icpc -c -O3 -restrict -opt-report x.cpp
```

インテル® コンパイラー 14.0:

```
x.cpp(6)  (列 15) リマーク: ループはベクトル化されませんでした: サポートされていないループ構造です。
```

インテル® コンパイラー 15.0:

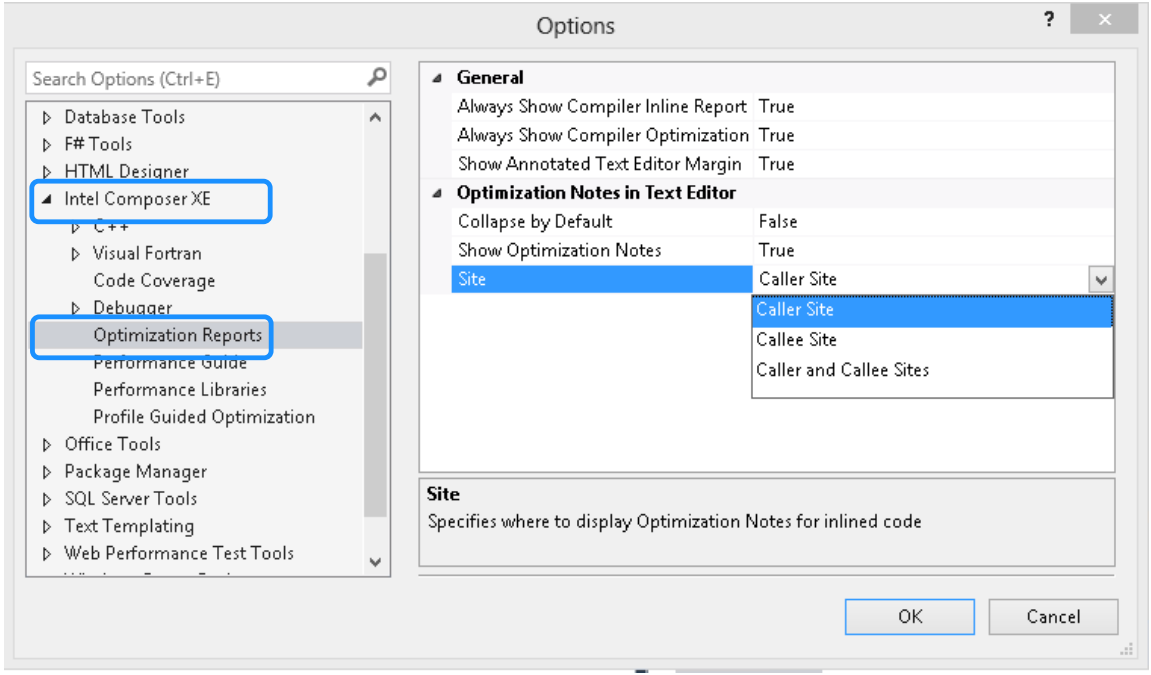
```
LOOP BEGIN at x.cpp(6,15)
   リマーク #15523: ループはベクトル化されませんでした: ループを実行する前にループの反復数を計算できません。
LOOP END
```

簡単に利用できる最適化レポート
インテル® コンパイラー

- より分かりやすく、アクションがとりやすいレポート情報
- すべてのレポートを 1 つのレポートに統合
  例: /Qopt-report、/Qvec-report、/Qopenmp-report などが /Qopt-report に統合
  - 個々のレポートも利用可能
    例: /Qopt-report-phase:vec はベクトル化レポートのみ生成
  - 任意の名前のファイルに出力し、後でビューアーやエディターで解析することも可能
- ほかのレポートオプションも引き続き利用できるが、レポートの内容および形式は新しいモデルと同じ
- デフォルトでは、生成されるオブジェクト・モデルごとに 1 つのレポートファイルを生成
  -opt-report-file=<filename|stderr|stdout> で変更可能。

Visual Studio* で設定できる新しい最適化レポート

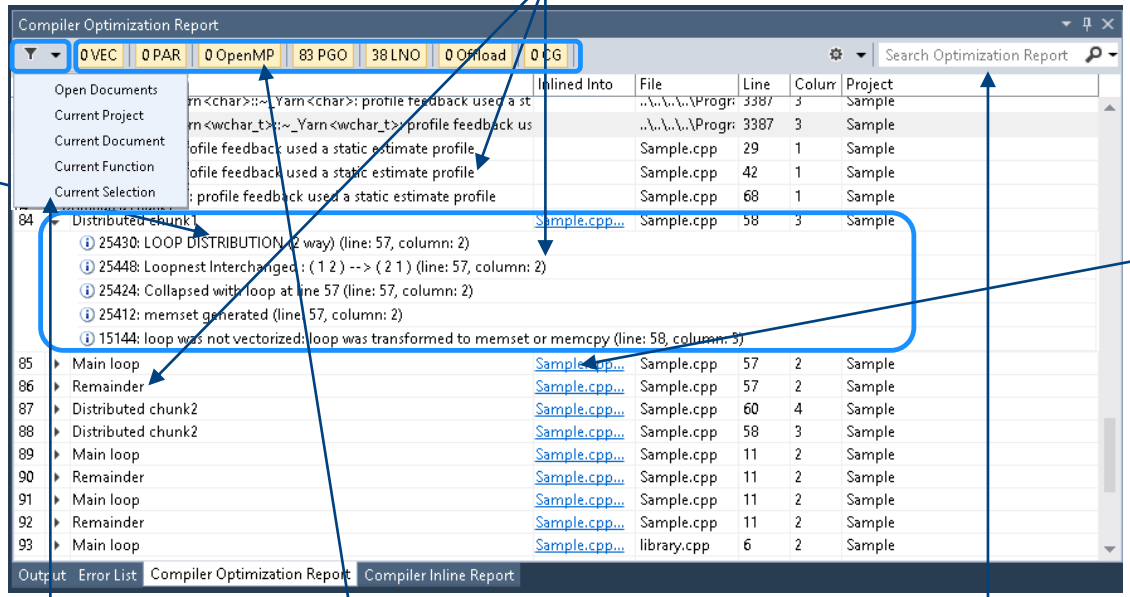
インテル® コンパイラー



新しい最適化レポートのツールウィンドウ

インテル® コンパイラー

ダブルクリックでソース位置にジャンプ



レポートを
階層表示

クリックで
呼び出しに
ジャンプ

コンテキストのスコープで
メッセージをフィルター

最適化フェーズでも
フィルター可能

任意のキーワードで
検索も可能

最適化に関する注意事項

テキストエディターで利用可能な新しい最適化レポート機能

インテル® コンパイラー

呼び出し先の最適化メモ

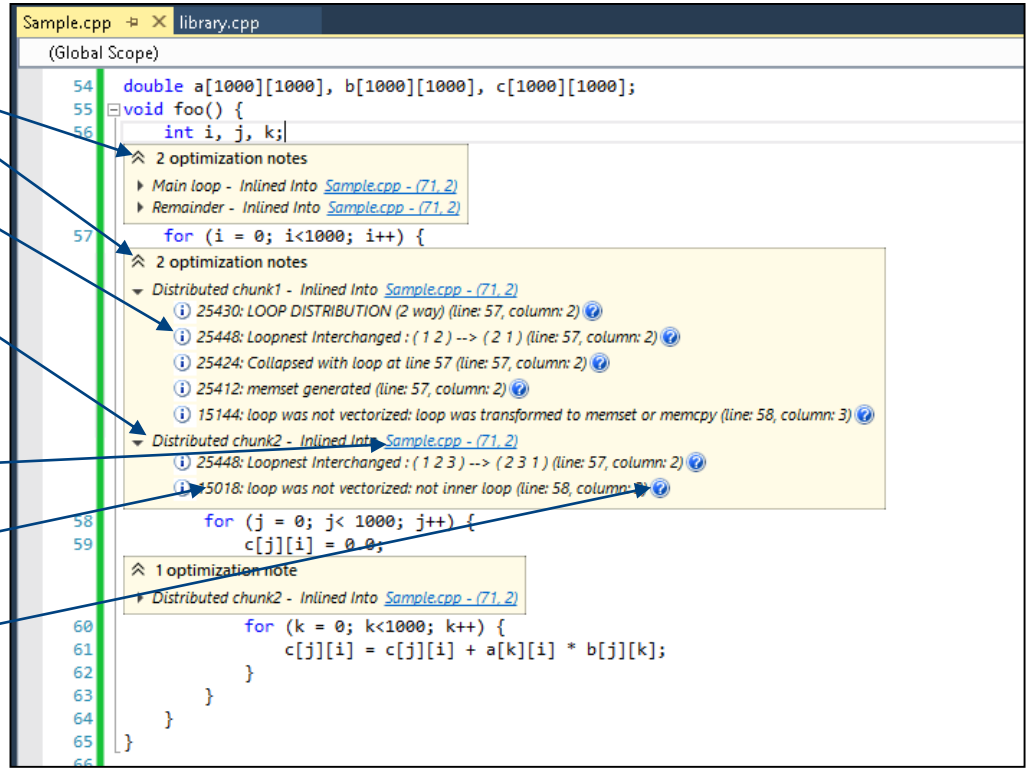
レポートの内容

呼び出し元の最適化メモ

クリックで呼び出し元にジャンプ

クリックで呼び出し先にジャンプ

"?" をクリックしてメッセージの
ヘルプを表示



最適化に関する注意事項

OpenMP* 4.0 サポート
インテル® コンパイラー

- ユーザー定義のリダクションを除くすべての機能をサポート
- CANCEL 宣言子: 最内領域の取り消しを要求
- CANCELLATION POINT 宣言子: 取り消し要求があったかどうかを、暗黙的または明示的なタスクがチェックするポイントを定義
- TASK 宣言子の DEPEND 節: タスク領域の兄弟タスクの依存性を定義することでタスクのスケジュールを制御
- 複合構造 (TEAMS DISTRIBUTE など)

新機能
インテル® C++ コンパイラー

- OpenMP* またはインテル® Cilk™ Plus による優れた外部ループ最適化機能
- インテル® Cilk™ Plus: 明示的なベクトル化。SIMD プラグマのキーワードバージョンを追加: _Simd、_Safelen、_Reduction
- タスク・スケジュールを制御してパフォーマンスを向上するタスク依存性を含め、OpenMP* 4.0 への対応をほぼ完了
- -qopt-report オプションで、より詳細な、ベクトル化を含む包括的な最適化レポートを簡単に生成
- C++11 言語をフルサポート
- 複数の関数バージョン (GCC 互換)

- コンパイラーの詳細
  - Linux* C++ で -O2 以上を指定すると、ベクトル化を含む優れたパフォーマンスが得られるように -ansi-alias がデフォルトで有効になる (gcc* で -fstrict-aliasing がデフォルトで有効になるのと同じ)
  - データ・アライメントに応じた複数のコードパスの生成を無効にする -no-opt-dynamic-align コンパイラー・オプションを追加
  - aligned_new ヘッダーを追加
  - SIMD データ型で算術演算および論理演算が使用可能に (__m128 など)
  - -fast/-Ofast で -fp-model fast=2 を適用
- ラムダ関数のデバッグを向上
- インテル® AVX512 対応のアップデートを近日公開予定

Linux* C++ で -ansi-alias をデフォルトに変更

インテル® C++ コンパイラー

- Linux* C++ で -O2 または -O3 を指定すると -ansi-alias がデフォルトで有効になる (gcc* で -fstrict-aliasing がデフォルトで有効になるのと同じ)
- -ansi-alias はコードが ANSI 準拠の別名規則に従うようにし、コンパイラーがより強力な最適化を有効にできるようにする
- -no-ansi-alias (gcc* では -fno-strict-aliasing) で無効にできる

C++11 標準言語機能をフルサポート
インテル® C++ コンパイラー

- 仮想関数オーバーライド
- 継承コンストラクター: struct Derived { using Base::Base; }
- 例外宣言の廃止
- ユーザー定義リテラル
- Thread_local (C++11 セマンティクス) (Linux* のみ)。C++11 ライブラリー機能は、プラットフォームの標準 C++ ライブラリーでサポートされている場合のみ利用可能。
- Windows*: msvcrt/libcmt、Linux*: libstdc++、OS X*: libc++/libstdc++

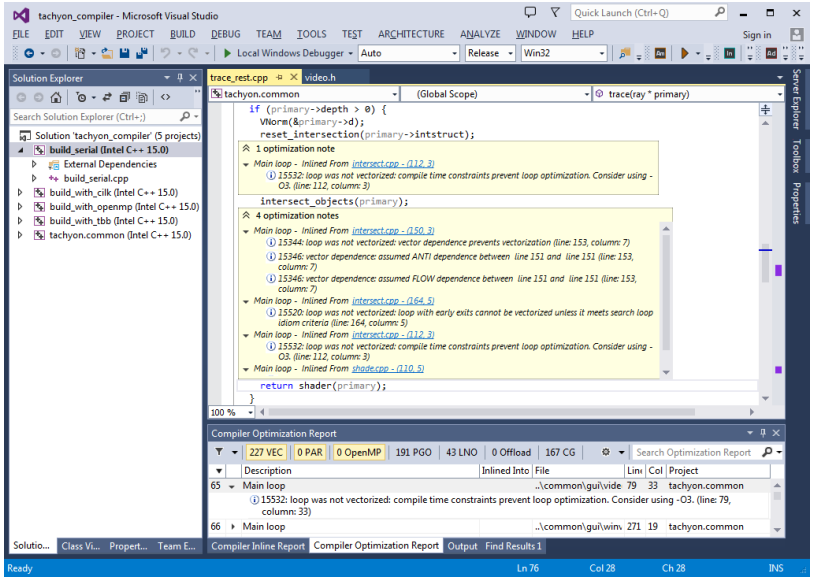
GNU* コンパイラーとの互換性
インテル® C++ コンパイラー

- C++11 サポートを有効にするには -std=c++11 (または -std=c++0x) オプションを使用する
- このオプションは、有効な GNU* 4.8 のヘッダーで使用されているすべての C++11 機能をサポート
- システムの GNU* (PATH の g++) に応じて、異なる機能が有効になることがある
- C++11 機能のサポートには GNU* C/C++ に含まれる C++ ヘッダーファイルからのサポートが必要 - 機能はバージョンに応じて異なる
- GNU* 4.8 以降の使用を推奨

Visual Studio*、Visual C++* との互換性

インテル® C++ コンパイラー

- 最適化レポートの統合



- Microsoft* ツールセット (2013 の作業や 2010 のターゲット ツールなど) を利用可能。まだ XP をターゲットにしている開発者には便利。
- インテル® Parallel Studio XE Composer Edition for C++ は、C++11 機能に関して、デフォルトで Microsoft* Visual C++* 2013 と完全に互換性がある
  - Microsoft* コンパイラーによりサポートされる C++11 機能はインテル® C++ コンパイラー (Windows*) でもサポート
  - C++11 機能を利用するために特別なコマンドライン・オプションや標準ライブラリー・ヘッダーで機能マクロの指定は不要
- VS2012 以前の C++11 サポートは Microsoft* の C++ ヘッダーファイルに含まれる機能に依存。追加の C++11 機能には、インテル固有の /Qstd=c++0x または /Qstd=c++11 オプションを使用。

最適化に関する注意事項

インテル® Cilk™ Plus によりタスク/データ並列処理が簡単に
インテル® C++ コンパイラー

- 3 つの簡単なキーワードだけでタスク/データ並列処理を実装:
  - Cilk_for、Cilk_spawn、Cilk_sync

シリアルコード (左) とインテル® Cilk™ Plus キーワードにより並列化された並列コード。オリジナル・コードは変更なし。

```
int fib  (int n)
{
  if (n <= 2)
    return n;
  else {
    int  x,y;
    x = fib(n-1);
    y = fib(n-2);
    return x+y;
  }
}
```

```
int fib (int n)
{
  if (n <= 2)
    return n;
  else {
    int  x,y;
    x = _Cilk_spawn fib(n-1);
    y = fib(n-2);
    _Cilk_sync;
    return x+y
  }
}
```

- ベクトル化の実装時間を短縮
  インテル® Cilk™ Plus の配列表記と #pragma SIMD

単純なベクトル乗算の配列表記

```
A[0:N] = b[0:N] * c[0:N];
```

より高度な配列表記

```
X[0:10:10] = sin(y[20:10:2]);
```

#pragma SIMD 追加前のサンプルコード

```
for(int  i = 2; i < n ;i++)
    y[i] = y[i-2] + 1;
```

#pragma SIMD 追加後のサンプルコード

```
#pragma simd vectorlength(2)
for(int i = 2; i < n ;i++)
    y[i] = y[i-2] + 1;
```

インテル® Cilk™ Plus と OpenMP* 4.0 の違い
インテル® C++ コンパイラー

インテル® Cilk™ Plus	OpenMP* 4.0
配列表記をサポート	配列表記はサポートしていない
_declspec(vector_variant)implement)...))) によりユーザー実装のベクトル関数をサポート。 コンパイラーにより生成されるベクトルコードの代わりに、ユーザーによる SIMD 対応関数のベクトルバージョンを実装可能。	ユーザー実装のベクトル関数はサポートしていない
#pragma simd には、"omp simd" でサポートされない firstprivate()、vecremainder、assert 節がある。 __assume_aligned() や __builtin_assume_aligned() により OMP4.0 の aligned() 節の動作を模倣できる。	"omp simd" には、#pragma simd でサポートされない collapse() 節と aligned() 節がある
#pragma simd に加えて、明示的なベクトル化で _Simd キーワードをサポート。 次の OpenMP* 4.0 構文と同様に、1 つの "for" ループのスレッド化とベクトル化を有効にできる。 #pragma omp parallel for simd	明示的なベクトル化で _Simd キーワードはサポートしていない
現在処理を実行中の simd レーンを特定する __intel_simd_lane() をサポート	個々の simd レーンの特定はサポートしていない
次のようなビルトインのリダクション操作をサポート: __sec_reduce_add()、__sec_reduce_max_ind()、__sec_reduce_mul()、__sec_reduce_min_ind()、__sec_reduce_all_zero()、__sec_reduce_all_nonzero()、__sec_reduce_min()、__sec_reduce_max()、__sec_reduce_and()、__sec_reduce_or()、__sec_reduce_xor()	ビルトインのリダクション操作はサポートしていない
次を使用したカスタム・リダクション関数の記述をサポート: __sec_reduce() と __sec_reduce_mutating()	カスタム・リダクション関数はサポートしていない
__sec_implicit_index() による配列の暗黙のインデックスをサポート	配列の暗黙のインデックスはサポートしていない

OS X* との互換性
インテル® C++ コンパイラー

- インテル® Parallel Studio 2015 のリリースに伴い、OS X* 用のコンパイラーを再パッケージ
  - インテル® Parallel Studio Composer Edition for OS X* には既存のコンパイラー・テクノロジーが含まれる
  - インテル® INDE 2015 Build Edition for OS X* には Clang ベースのコンパイラーが含まれる
- インテル® Parallel Studio XE 製品: HPC ソフトウェア開発者向け
- インテル® INDE 製品: クライアント/モバイル・ソフトウェア開発者向け

新機能
インテル® Fortran コンパイラー

- 自動ベクトル化と (インテルまたは OpenMP* 宣言子による) 明示的なベクトル化の向上 (特に外側のループのベクトル化) により、優れた安定性とパフォーマンスを提供
- Fortran 2003 をフルサポート (パラメーター化された派生型を含む)
- BLOCK を含む Fortran 2008 の多くの機能をサポート
- タスク依存性を含め、OpenMP* 4.0 への対応をほぼ完了 (残りはユーザー定義のリダクションへの対応)
- 実行ごとの再現性を保証し、(-fp-model precise と比べると) パフォーマンスへの影響がわずかな -no-opt-dynamic-align オプション
- REAL 型と COMPLEX 型のまだ初期化されていない保存済みスカラー/配列変数をシグナル型 NaN に初期化する -init=snan オプション
- 互換性保持機能により Fortran から呼び出し可能な SIMD ベクトル関数の simd レーン番号を表す __intel_simd_lane() 組込み関数
- ポインター型配列と連続していない配列スライスの (インテル® Xeon Phi™ コプロセッサーへの) オフロードをサポート
- GDB デバッガーに対応 (インテル® デバッガーは削除)
- -fast/-Ofast で -fp-model fast=2 を適用
- インテル® AVX512 対応のアップデートを近日公開予定

Fortran OpenMP* サポート: WORKSHARE
インテル® Fortran コンパイラー

- 多くの場合は並列化可能
- A = B + C のような単純な配列代入は並列化される
- A = A + B + C のようなオーバーラップを含む単純な配列代入は並列化される
- A = A + F (B) のようなユーザー定義関数を含む配列代入は並列化される
  (F は ELEMENTAL でなければならない)
- A = A + B(1:4) + C(1:4) のような代入文の右辺に配列スライスを含む配列代入は並列化される。左辺の下限、配列スライスの下限、または右辺の配列スライスのストライドが 1 でない場合は並列化されない。

Fortran 2003 のパラメーター化された派生型

インテル® Fortran コンパイラー

- KIND および長さ無指定の引数型のテンプレートを作成可能
- KIND 型引数はコンパイル時定数、長さ引数は実行時定数
  例:
  - TYPE humongous_matrix(k, d)
    INTEGER, KIND :: k = kind(0.0)
    INTEGER(selected_int_kind(12)), LEN :: d
    REAL(k) :: element(d,d)
  - END TYPE
  - TYPE(humongous_matrix(8,10000000)) :: giant

Fortran 2008: BLOCK 構文

インテル® Fortran コンパイラー

- 実行可能な構文に宣言を含めることができる
- 構文内で宣言された変数はその範囲のローカル変数となる
- COMMON、EQUIVALENCE、NAMELIST、IMPLICIT は許可されていない
- SAVE は構文のローカルで許可されている
- 範囲外の SAVE は BLOCK に影響しない
- ラベルと書式はローカルではない
- スレッドローカルな DO CONCURRENT で便利

インテル® Fortran コンパイラーの BLOCK 構文例

BLOCK 構文例

```
IF (swaxpy) THEN
  BLOCK
    REAL(KIND(x)) tmp
    tmp = x
    x = y
    y = tmp
  END BLOCK
END IF
```

F08: BLOCK を含む DO CONCURRENT の例

```
DO CONCURRENT (I = 1:N)
  BLOCK
    REAL T
    T = A(I) + B(I)
    C(I) = T + SQRT(T)
  END BLOCK
END DO
```

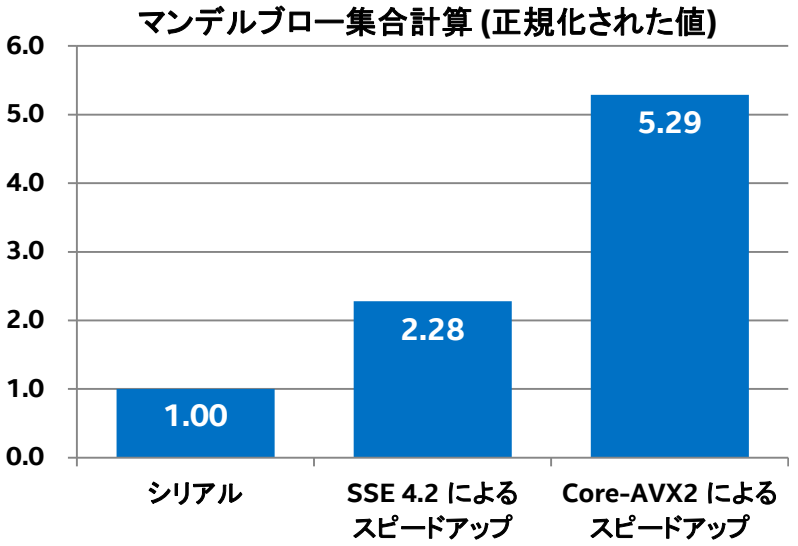
BLOCK を使用せずに各反復でローカルな (threadprivate) 一時変数を作成することはできない

パフォーマンス指標

インテル® コンパイラー

OpenMP* 4.0 SIMD による C++ の明示的なベクトル化

```
typedef float complex fcomplex;
 const uint32_t max_iter = 3000;
#pragma omp declare simd uniform(max_iter), simdlen(16)
uint32_t mandel(fcomplex c, uint32_t max_iter)
{
    uint32_t count = 1; fcomplex z = c;
    while ((cabsf(z) < 2.0f) && (count < max_iter)) {
       z = z * z + c; count++;
    }
    return count;
}
uint32_t count[ImageWidth][ImageHeight];
 …… …….……
      for (int32_t y = 0; y < ImageHeight; ++y) {
        float c_im = max_imag - y * imag_factor;
#pragma omp simd safelen(16)
        for (int32_t x = 0; x < ImageWidth; ++x) {
          fcomplex in_vals_tmp = (min_real + x *
real_factor) + (c_im * 1.0iF);
          count[y][x] = mandel(in_vals_tmp, max_iter);
        }
      }
… … ….….
```



システム構成: インテル® Xeon® プロセッサー E3-1270 v3 @ 3.50GHz (4 コア、ハイパースレッディング有効)、32GB RAM、L1 キャッシュ 256KB、L2 キャッシュ 1MB、L3 キャッシュ 8MB、Windows Server* 2012 R2 Datacenter (64 ビット版)。コンパイラー・オプション: -O3 -Qipo -QxSSE4.2 (SSE4.2 の場合)、または -O3 -Qipo -QxCORE-AVX2 (AVX2 の場合)。詳細は、http://www.intel.com/performance (英語) を参照してください。

高速なコードの迅速な開発

インテル® コンパイラー

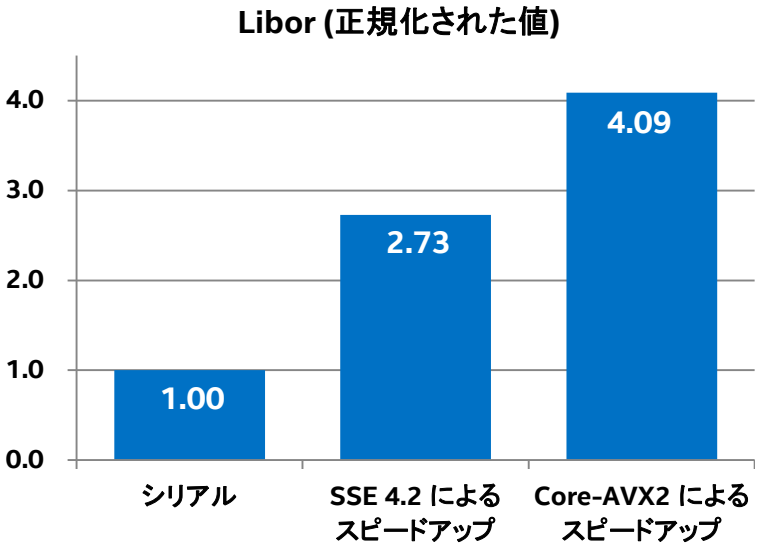
OpenMP* 4.0 SIMD による C++ の明示的なベクトル化

```
#pragma omp declare simd linear(z:40) uniform(L, N, Nmat) linear(k)
float path_calc(float *z, float L[][VLEN], int k, int N, int Nmat)

#pragma omp declare simd uniform(L, N, Nopt, Nmat) linear(k)
float portfolio(float L[][VLEN], int k, int N, int Nopt, int Nmat)
… … …
for (path=0; path<NPATH; path+=VLEN) {
     /* フォワード・レートを初期化 */
     z = z0 + path * Nmat;
#pragma omp simd linear(z:Nmat)
    for(int k=0; k < VLEN; k++) {
       for(i=0;i<N;i++) {
           L[i][k] = L0[i];
       }

       /* LIBOR パスの計算 */
       float temp = path_calc(z, L, k, N, Nmat);
       v[k+path]  = portfolio(L, k, N, Nopt, Nmat);

       /* 次のブロックの先頭にポインターを移動 */
       z += Nmat;
    }
 }
… … … ….
```



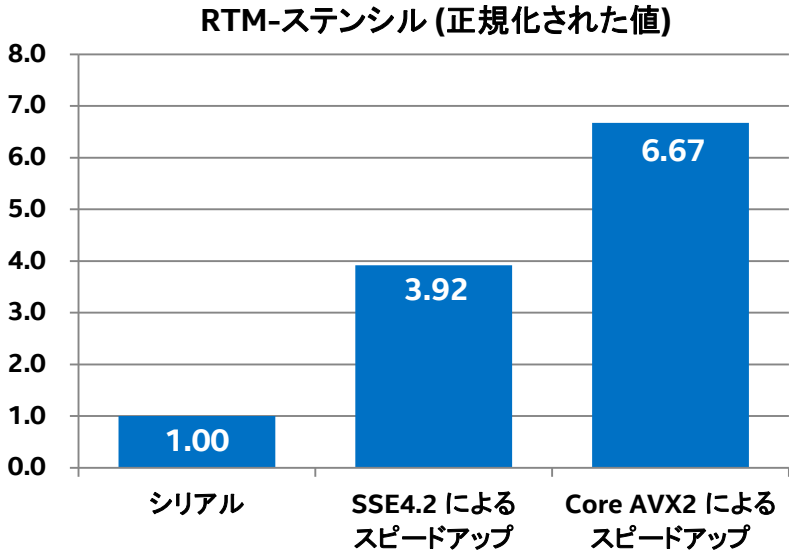
システム構成: インテル® Xeon® プロセッサー E3-1270 v3 @ 3.50GHz (4 コア、ハイパースレッディング有効)、32GB RAM、L1 キャッシュ 256KB、L2 キャッシュ 1MB、L3 キャッシュ 8MB、Windows Server* 2012 R2 Datacenter (64 ビット版)。コンパイラー・オプション: -O3 -Qipo -QxSSE4.2 (SSE4.2 の場合)、または -O3 -Qipo -QxCORE-AVX2 (AVX2 の場合)。詳細は、http://www.intel.com/performance (英語) を参照してください。

高速なコードの迅速な開発

インテル® コンパイラー

C++ の明示的なベクトル化: SIMD パフォーマンス

```
… … …
#pragma simd vectorlength(8)
  for (int x = x0; x < x1; ++x) {
    float div = coef[0] * A_cur[x]
              + coef[1] * ((A_cur[x + 1] + A_cur[x - 1])
              + (A_cur[x + Nx] + A_cur[x - Nx])
              + (A_cur[x + Nxy] + A_cur[x - Nxy]))
              + coef[2] * ((A_cur[x + 2] + A_cur[x - 2])
              + (A_cur[x + sx2] + A_cur[x - sx2])
              + (A_cur[x + sxy2] + A_cur[x - sxy2]))
              + coef[3] * ((A_cur[x + 3] + A_cur[x - 3])
              + (A_cur[x + sx3] + A_cur[x - sx3])
              + (A_cur[x + sxy3] + A_cur[x - sxy3]))
              + coef[4] * ((A_cur[x + 4] + A_cur[x - 4])
              + (A_cur[x + sx4] + A_cur[x - sx4])
              + (A_cur[x + sxy4] + A_cur[x - sxy4]));
    A_next[x] = 2 * A_cur[x] - A_next[x] + vsq[s+x] * div;
  }
… … …
```

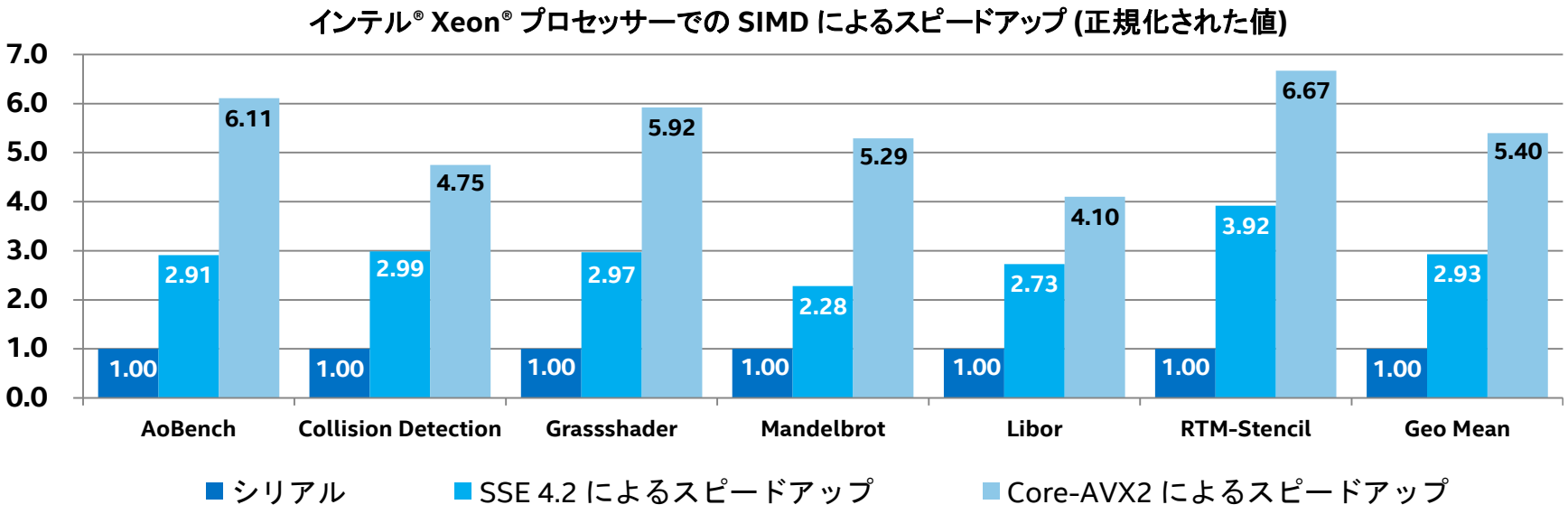


システム構成: インテル® Xeon® プロセッサー E3-1270 v3 @ 3.50GHz (4 コア、ハイパースレッディング有効)、32GB RAM、L1 キャッシュ 256KB、L2 キャッシュ 1MB、L3 キャッシュ 8MB、Windows Server* 2012 R2 Datacenter (64 ビット版)。コンパイラー・オプション: -O3 -Qipo -QxSSE4.2 (SSE4.2 の場合)、または -O3 -Qipo -QxCORE-AVX2 (AVX2 の場合)。詳細は、http://www.intel.com/performance (英語) を参照してください。

高速なコードの迅速な開発

インテル® コンパイラー

OpenMP* 4.0 SIMD/インテル® Cilk™ Plus による C++ の明示的なベクトル化: SIMD パフォーマンス



システム構成: インテル® Xeon® プロセッサー E3-1270 v3 @ 3.50GHz (4 コア、ハイパースレッディング有効)、32GB RAM、L1 キャッシュ 256KB、L2 キャッシュ 1MB、L3 キャッシュ 8MB、Windows Server* 2012 R2 Datacenter (64 ビット版)。コンパイラー・オプション: -O3 -Qipo -QxSSE4.2 (SSE4.2 の場合)、または -O3 -Qipo -QxCORE-AVX2 (AVX2 の場合)。
詳細は、http://www.intel.com/performance (英語) を参照してください。

高速なコードの迅速な開発

インテル® コンパイラー

C++ の並列化: インテル® Cilk™ Plus キーワードの利用

修正前: シリアルコード

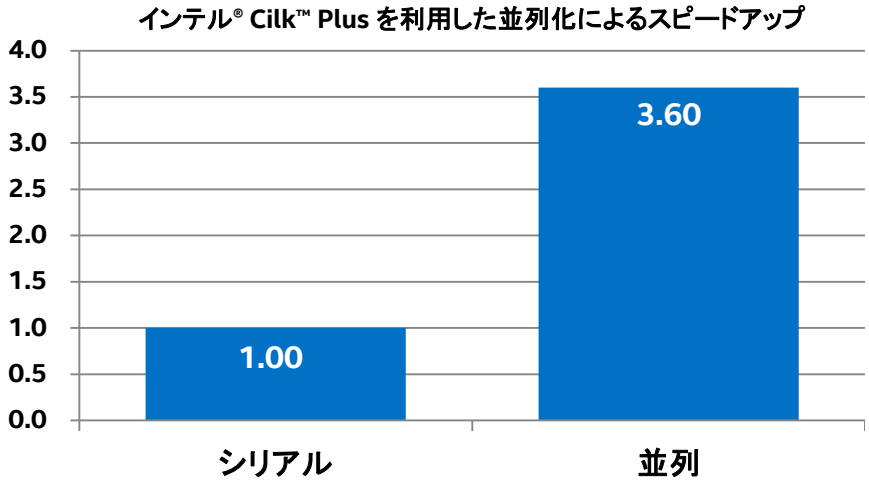
```
void Search( const Board& b ) {
  …
     } else {
     Search( Board(b,i,0) );
     Search( Board(b,i,1) );
     }}}
```

30.2 秒

修正後: 並列コード (fork/join によるタスク並列化)

```
void Search( const Board& b ) {
  …
     } else {
     cilk_spawn Search( Board(b,i,0) );
     Search( Board(b,i,1) );
     }}}
```

8.4 秒



システム構成: インテル® Core™ i7-4770K プロセッサー (開発コード名: Haswell) @ 3.50GHz (4 コア、ハイパースレッディング有効)、16GB RAM、12M キャッシュ、Ubuntu* (64 ビット)。

完全なコードは、次の Web サイトからダウンロードできます。
http://www.isus.jp/article/idz/parallel/cilk-plus-solver-for-a-chess-puzzle-or-how-i-learned-to-love-rejection

ベンチマーク情報の詳細: http://www.intel.com/performance (英語)

高速なコードの迅速な開発

法務上の注意書きと最適化に関する注意事項

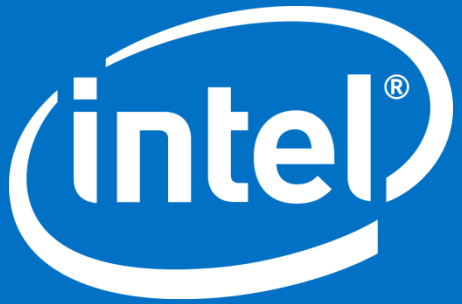
本資料の情報は、現状のまま提供され、本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他、知的財産権の侵害への保証を含む) をするものではありません。

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。

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®

参考情報

スピーカー: 顧客に応じて適切なスライドを使用すること

インテル® Xeon Phi™ コプロセッサーでのフォワード・スケーリング

インテル® Cilk™ Plus の配列表記または #pragma simd

- 毎回ベクトル化
- インテル® Xeon プロセッサーとインテル® Xeon Phi™ コプロセッサーで最適な SIMD 命令を利用

配列表記: (推奨)

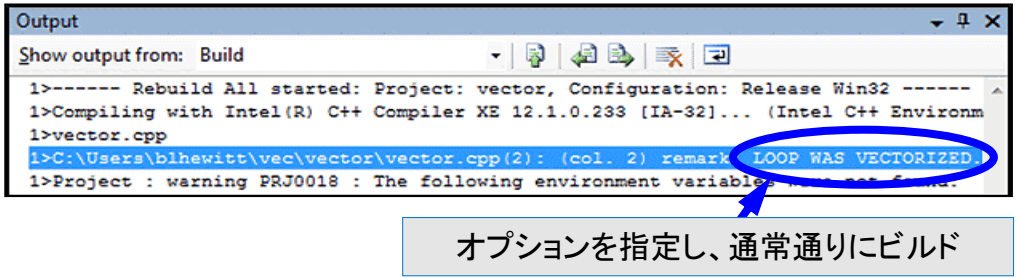
```
a[0:n] = a[0:n] + b[0:n];
```

#pragma simd: (使用には注意が必要)

```
#pragma simd
for (i=0; i<n; i++){
 a[i] = a[i] + b[i];
}
```

自動ベクトル化

- 最初に試すべき最も簡単な方法
- ガイド付き自動並列化 (GAP) はベクトル処理を増やすための変更をアドバイス



オプションを指定し、通常通りにビルド

新しいハードウェア向けのソフトウェアの再作業を最小限に

最適化に関する注意事項

最高レベルのパフォーマンス
インテル® C++ コンパイラー

- インテル® Cilk™ Plus: 並列化と分かりやすい構文により優れたパフォーマンスを提供する、簡単に使える配列表記の言語拡張
- パフォーマンスの向上を管理
  - コードのより多くの領域に適用される強化されたベクトル化と自動ベクトル化 (ビルドログで違いを確認してみてください)
  - 低オーバーヘッドのループおよび関数プロファイルにより hotspot とスレッドを追加すべき個所を特定
- 優れたパフォーマンスを提供する生産性ツールであるガイド付き自動並列化が、コンパイラーにより自動ベクトル化/自動並列化されるようにコード変更をアドバイス
- Visual C++* との互換性を強化するため、より多くの C++ 0x および C99 標準機能をサポート
- インテル® SSE 4.2 命令を使用する最適化された文字列組込み関数により、パフォーマンスをさらに向上

プロシージャー間の最適化 (IPO)

- クロスモジュールの最適化
- IPO はシームレスなプロセス。ほとんどの最適化は実際にはリンクフェーズで行われる。
- IPO の利点
  - 多くの頻繁に使用される小/中規模の関数 (特にループで呼び出されるもの) を最適化
  - 関数のインライン展開
    - 引数のセットアップ、呼び出しの分岐/リターンのオーバーヘッドを排除
    - ほかの最適化の可能性 (const prop、DCE、&c)
- 不要コードの排除、効率的なレジスター利用
- エイリアス解析の向上による自動ベクトル化とループ変換の向上
- 場合によっては、ビルド時間とバイナリーサイズが増えることがある

自動ベクトル化

- 自動ベクトル化は SIMD/DLP の可能性を調査
  - インテル® SSE およびインテル® AVX 命令によりシーケンシャル操作を自動ベクトル化
  - ソースコードの大幅な変更はなし
  - 分かりやすく、デバッグと保守も簡単
  - コンパイラー/プロセッサーの将来を考慮
- ターゲット・プロセッサー向けに最適化されたコード
  - インテル® プロセッサーおよび互換プロセッサーに対応
  - 異なるプロセッサー環境をサポート
- プロセッサー固有の最適化
  - 特定のインテル® プロセッサー向け
  - 例: インテル® Core™ i7 プロセッサーの場合は /QxSSE4.2 を使用

「ループがベクトル化されました。」ビルドメッセージ

```
subroutine quad(len,a,b,c,x1,x2)
  real(4) a(len),b(len), c(len), x1(len), x2(len), s

  do i=1,len
    s = b(i)**2 - 4.*a(i)*c(i)
    if (s.ge.0.) then
      x1(i) = sqrt(s)
      x2(i) = (-x1(i) - b(i)) *0.5 / a(i)
      x1(i) = ( x1(i) - b(i)) *0.5 / a(i)
    else
      x2(i)=0.
      x1(i)=0.
    endif
  enddo
end

> ifort -c -vec-report2 quad.f90
quad.f90(4): (列 3) リマーク: ループがベクトル化されました。
```

IA-32 およびインテル® 64 向けコードで優れた結果が得られる
インテル® Xeon Phi™ コプロセッサーでは追加のチューニングが必要

自動並列化

- 可能な場合、コードのシリアル領域をマルチスレッド・コードに自動変換
  - シリアルコードのワークシェアに適した領域を特定
  - 正しい並列実行を確認するためデータフロー解析を実行
  - マルチスレッド・コード向けにデータを分割
- 並列ランタイムサポートは OpenMP* と同様の機能を提供
  - ループ反復の変更に伴う細かい処理
  - スレッドのスケジューリング
  - 同期
- "/Qparallel" オプションで有効になる

IA-32 およびインテル® 64 向けコードで優れた結果が得られる
インテル® Xeon Phi™ コプロセッサーでは追加のチューニングが必要

何が必要かコンパイラーがアドバイス
ガイド付き自動並列化 (GAP)

- 目的
  - 効率良く、簡単にアプリケーションを並列化する方法
  - ビルトインのコンパイラー・テクノロジーを利用して並列化にかかる時間を短縮
- GAP とは?
  - 再コンパイルしたときに、ベクトル化、並列化、データ変換によりコードが自動的に最適化されるように、開発者にコード変更をアドバイスするコンパイラー・ベースのアナライザー
  - 既存の自動ベクトル化と自動並列化テクノロジーをベースにしている
- GAP は次のことには対応していない
  - コードの解析とスレッド化候補の特定 (インテル® Advisor を使用)
  - スレッドの正当性検証 (インテル® Inspector XE を使用)
  - パフォーマンス/hotspot 解析 (インテル® VTune™ Amplifier XE を使用)

GAP によるアドバイスのセマンティクスを確認するのは開発者の責任
IA-32 およびインテル® 64 向けコードで優れた結果が得られる
インテル® Xeon Phi™ コプロセッサーでは追加のチューニングが必要

GAP の利用に関して

- 最適化レベルを /O2 以上に設定する必要がある
  - GAP の出力はさまざまなメッセージであって .exe ではない
  - コマンドライン・オプションまたは Visual Studio* IDE のどちらでも動作
  - IPO や PGO は不要だが、オプションに基づいてアドバイスが代わる可能性がある
  - ユーザーは GAP によるアドバイスの一部または全部を適用するか選択可能。ただし、1 つのループに対して複数のメッセージが出力された場合は、適切に最適化されるようにそのループに対するすべてのアドバイスを適用する必要がある。
- ユーザーがファイルやルーチンの "ホット" な領域を指定可能
  - その場合、ホットな領域のアドバイスのみに制限される
  - デフォルトではコンパイル単位全体に対するアドバイスが出力される
- 次のようなアドバイスが出力される
  - 新しい特性を有効にするソース変更
  - ループに対するプラグマの追加 (セマンティクスを満たす場合)
  - 新しいオプションの追加

IA-32 およびインテル® 64 向けコードで優れた結果が得られる
インテル® Xeon Phi™ コプロセッサーでは追加のチューニングが必要

最適化レポート

- GAP: ガイド付き自動並列化
  - /Qguide オプションは並列化を有効にするソース変更をアドバイス
  - 実際にコードは生成せず、解析とアドバイスのみを提供
- その他のレポート
  - /Qvec-report: ベクトル化されたループ、ベクトル化されなかったループとその理由を出力
  - /Qpar-report
  - /Qopt-report: さまざまな最適化のレポートを出力
  - 詳細は icl -help reports で確認

IA-32 およびインテル® 64 向けコードで優れた結果が得られる
インテル® Xeon Phi™ コプロセッサーでは追加のチューニングが必要

高レベルの最適化 (HLO)

- -O3 (Windows* では /O3) で有効になる
  - 自動ベクトル化により /O2 よりも強力なデータ依存性の解析が行われる
  - ソースコードの特性 (ループと配列) を調査
  - ループ変換の最良のチャンス
- ループ変換を実行
  - ループ分配
  - ループ交換
  - ループ融合
  - ループアンロール
  - データ・プリフェッチ
  - PGO ベースのループアンロール
  - など

並列プログラミング・モデル

- インテル® Cilk™ Plus (C/C++ 言語拡張)
  - キーワードベース: cilk_for、cilk_spawn、cilk_sync
  - 配列のベクトル表記
  - SIMD 対応関数 (ベクトル関数)
- インテル® TBB
  - タスク用の C++ コンテナー
  - 柔軟な粒度設定
  - 洗練されたビルトインのタスク・スケジューリング