インテル® C++ Composer XE 2011 Windows* 版

― 入門ガイド ―

エクセルソフト株式会社
www.xlsoft.com

Rev. 1.2 (2011/05/03)

ー 目次 ー

1．はじめに

インテル® C++ Composer XE 2011（以下、本製品）は、インテル® C++ コンパイラー12.0 に加えて、数値演算ライブラリー（MKL）、マルチメディア向けライブラリー（IPP）またアプリケーションの並列化ライブラリー（TBB）が含まれています。本ドキュメントでは、インテル® C++ コンパイラー 12.0 について記述します。

本ドキュメントでは、製品に含まれるサンプルプログラムを使用して、コマンドラインからのコンパイル、および Microsoft* Visual Studio* 統合開発環境 (IDE) からのビルド手順を説明しています。インテル® C++ コンパイラーの基本動作を確認するとともに、インストール後の動作検証を行うことができます。また、これらの手順説明の中で、インテル® C++ コンパイラー 12.0 に含まれるいくつかの最適化オプションを使用してパフォーマンスの検証も行っていますので、オプションの内容や設定方法なども習得することができます。さらに、その他の最適化オプションや機能の紹介も加えています。最後に、本製品を使用する際の、関連・追加情報もいくつか記載されていますので、必要に応じて内容を参照してください。

なお、本ドキュメントでは、インテル® C++ Composer XE 2011（日本語版）を使用し、以下のデフォルトのインストールフォルダーにインストールしている環境を使用しています。

C:¥Program Files¥Intel¥ComposerXE-2011¥

また、本ドキュメントでは Visual Studio 2008 および Visual Studio 2010 のバージョンを使用しています。Visual Studio 2005 のバージョンを使用している場合は、Visual Studio 2008 の内容を参照してください。

本ドキュメントで使用しているシステム情報は以下のとおりです。

- プロセッサー：インテル® Core™ 2 Quad CPU Q6600 2.40GHz
- 搭載メモリ：4.00 GB（3.25 GB 使用可能）
- OS：Windows 7 Professional x86

  ※オペレーティング・システムに x64 システムをご使用の場合は、本ドキュメントのなかで“Program Files”を“Program Files (x86)”と読み直してください。

本ドキュメントに記載されていない詳細な内容は、製品ドキュメントを参照してください。

2．サンプルプログラムと作業準備

本ドキュメントでは、本製品に含まれる以下のサンプルプログラムを使用します。サンプルプログラムは Zip 形式で圧縮されています。

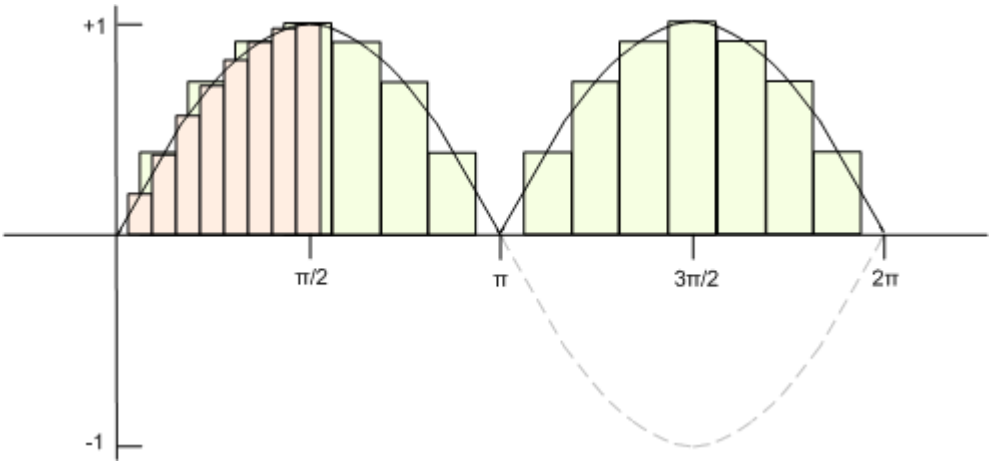
C:¥Program Files¥Intel¥ComposerXE-2011¥Samples¥ja_JP¥C++¥optimize.zip
または、
C:¥Program Files¥Intel¥ComposerXE-2011¥Samples¥en_US¥C++¥optimize.zip

> Note：本製品では、フォルダー名“ja_JP”は日本語ファイル用フォルダー、“en_US”は英語ファイル用フォルダーを意味しています。

このサンプルプログラムを適当なユーザーフォルダーにコピーして解凍してください。
ここでは、C ドライブに“temp”フォルダーを作成し、サンプルプログラムをコピーして解凍しています。
C:¥temp¥optimize.zip

解凍が完了すると optimize フォルダーが作成され、そのフォルダー内にいくつかのファイルが解凍されます。
本ドキュメントでは、ソースファイル“int_sin.c”をサンプルプログラムとして使用します。

本サンプルプログラムは、1 サイクル 2πラジアンの正弦曲線の絶対値を積分する数値演算プログラムです。次の図は、計算に使用される方法を示しています。この方法は、曲線と上辺の中央部分が一致するように長方形を連続的に追加します。長方形の数が増えると (長方形の幅が狭くなると)、計算される領域は 4 (4.0) に近づきます。この図では、2^4 内点と 2^5 内点の最初の 8 片におけるイメージを示しています。



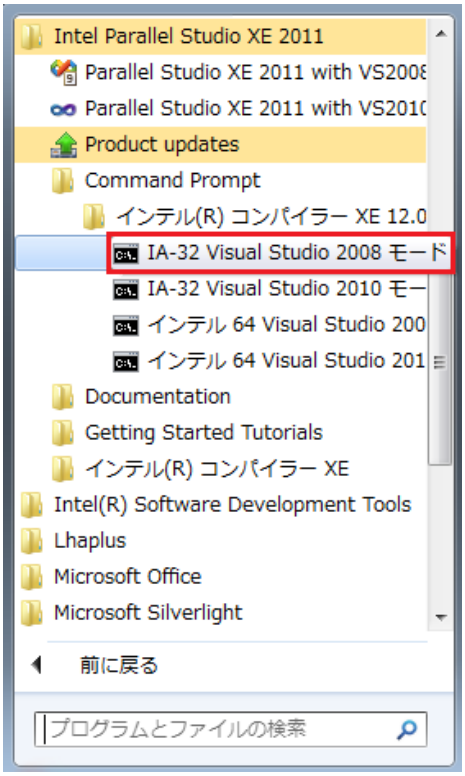
このサンプルプログラムをコンパイルして実行し、計算結果が既知の正しい値である 4.0 に収斂するかどうかをチェックします。また、サンプルプログラムの計算処理の開始と終了時に時間関数（clock）がコールされ、計算にかかった経過時間（プロセッサ時間）を測定しています。

3. コマンドラインからのコンパイル方法

インテル® C++ コンパイラーをコマンドラインから実行する場合は、“icl”コマンドを使用します。以下にコマンドラインからのコンパイル手順を記します。

1. Windows [スタート] メニューから [プログラム] - [Intel Parallel Studio XE 2011] - [Command Prompt] - [インテル(R) コンパイラー XE 12.0 Update 1] - [IA-32 Visual Studio 2008 モード]を選択して、インテル® C++ コンパイラー専用コマンドプロンプトを開きます。このコマンドプロンプトでは起動時に以下のバッチファイルが実行され、コンパイルに必要な環境変数（PATH、LIB、INCLUDE 等）の設定が自動で行われます。

   "C:¥Program Files¥Intel¥ComposerXE-2011¥bin¥ipsxe-comp-vars.bat" ia32 vs2008



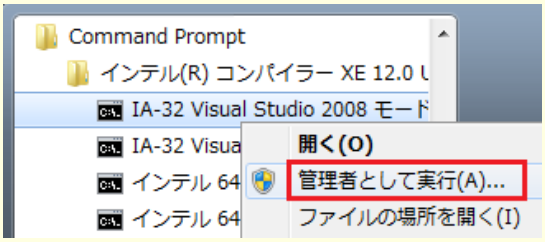
このコマンドプロンプトのモードの種類には、使用する「インテル® C++ コンパイラーのアーキテクチャー」と「Visual Studio のバージョン」の組み合わせによって、以下のようなパターンがあります。

- [IA-32 Visual Studio 2005 モード]
- [IA-32 Visual Studio 2008 モード]
- [IA-32 Visual Studio 2010 モード]
- [インテル 64 Visual Studio 2005 モード]
- [インテル 64 Visual Studio 2008 モード]
- [インテル 64 Visual Studio 2010 モード]

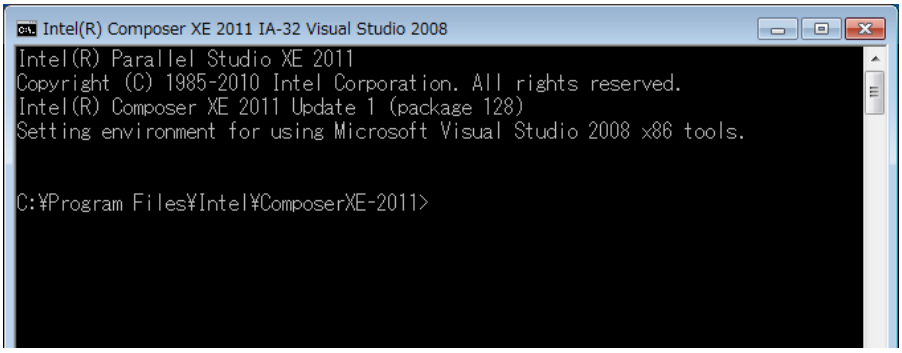
それぞれのモードで適切な環境変数が設定されます。

本ドキュメントでは、IA-32 用インテル C++ コンパイラーと Visual Studio 2008 の組み合わせのモードを使用しています。

ご注意：コンパイル処理が正常終了しない場合は、下図のようにショートカットを右クリックして表示されるメニューから [管理者として実行] を選択して再度お試しください。



図：インテル® C++ コンパイラー専用コマンドウィンドウ



2. 表示されるコマンドプロンプトに、まずは以下のように icl コマンドを実行してみましょう。この実行でコンパイラーのバージョン情報などが表示されていれば、icl コマンドへのパスが確認されたことになります。

```
> icl
```

3. 次にマイクロソフト・リンカー“link”コマンドを実行してみましょう。この実行で link コマンドの使用方法などが表示されることを確認してください。このマイクロソフト・リンカーは、icl コマンドによってコールされます。

```
> link
```

ご注意： link コマンドの実行が正常に行われない場合は、Visual Studio などのビルド環境が正しくインストールされていない可能性があります。製品リリースノートを参照して、本製品のサポートするビルド環境を確認し、本製品をアンインストールした後、正しいビルド環境を構築してから改めて本製品をインストールしなおしてください。

Note： icl コマンドは、内部でインテル® C++ コンパイラー本体（mcpcom.exe）をコールしてコンパイル処理を行い、その後マイクロソフト・リンカー（link.exe）をコールしてリンク処理を行います。このように icl コマンドはビルド工程をハンドルするツールであり、一般的にインテル® C++ コンパイラー・ドライバーと呼ばれます。

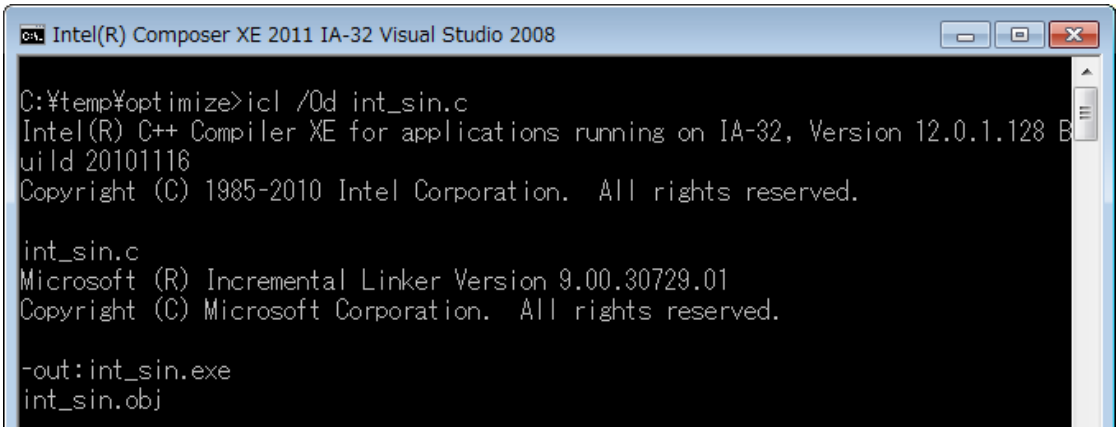
4. カレント・ディレクトリーを int_sin.c サンプルプログラムが存在するフォルダーまで移動します。

```
> cd c:¥temp¥optimize
```

３－１．コンパイル（最適化オプションなし）

最初に、最適化オプションを使用しないでコンパイルし、パフォーマンスの基準を確認します。次のようにインテル® C++ コンパイラーを実行してサンプルプログラムのコンパイルを行ってください。

```
> icl /Od int_sin.c
```



Note：インテル® C++ コンパイラーは、デフォルトでいくつかの最適化オプションが有効になっています。そのため、最適化なしでコンパイルするためには、オプション（/Od）を付加してデフォルトの最適化オプションを無効にする必要があります。なお、/Od などの “O” は大文字アルファベットのオーです。これは、Optimization（最適化）の頭文字を意味しています。

また、以下のように /Zi デバッグ・オプションを使用しても構いません。この場合もデフォルトの最適化オプションが無効になり、かつデバッグ情報が組み込まれます。

```
> icl /Zi int_sin.c
```

３－２．実行/プログラムの検証

実行ファイルは、サンプルプログラムと同じディレクトリーに “int_sin.exe” という名前で生成されます。次のようにプログラムを実行します。

```
> int_sin.exe
```

本サンプルプログラムは、内点の数が増えると、計算値が 4.0 に近く (または等しく) なります。プログラムを実行すると、次のような出力結果が表示されます。

Number of Interior Points	Computed Integral
4	3.141593e+000
8	3.792238e+000
16	3.948463e+000
32	3.987141e+000
64	3.996787e+000
128	3.999197e+000
256	3.999799e+000
512	3.999950e+000
1024	3.999987e+000
2048	3.999997e+000
4096	3.999999e+000
8192	4.000000e+000
16384	4.000000e+000
32768	4.000000e+000
65536	4.000000e+000
131072	4.000000e+000
262144	4.000000e+000
524288	4.000000e+000
1048576	4.000000e+000
2097152	4.000000e+000
4194304	4.000000e+000
8388608	4.000000e+000
16777216	4.000000e+000
33554432	4.000000e+000
67108864	4.000000e+000

Application Clocks = 8.299000e+003

３－３．コンパイル（最適化オプションあり）

インテル® C++ コンパイラーには多くの最適化オプションが用意されています。これらの最適化オプションを使用してプログラムのパフォーマンスを向上させることができます。

インテル® C++ コンパイラーには、デフォルトで以下の最適化オプションが含まれます。

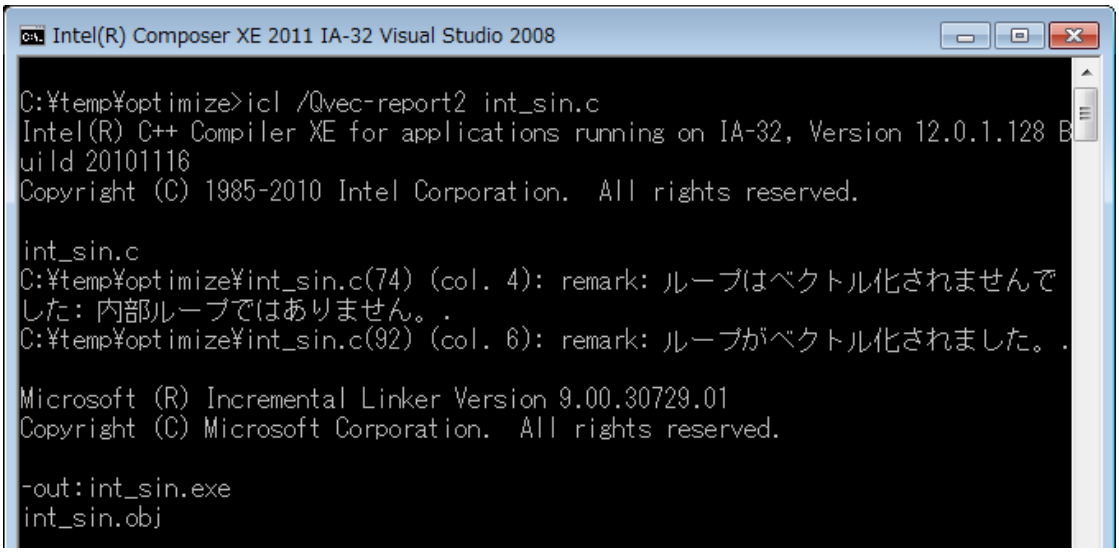
- /O2　　・・・ 速度重視の最適化オプション
- /arch:SSE2・・・ SSE2 命令を搭載したプロセッサーに特化したコードを生成するオプション

つまり以下のように、特にオプションを指定しないでコンパイルした場合は、上記のデフォルト最適化オプションが有効となります。

```
> icl int_sin.c
```

このデフォルトの２つのオプションを使用することにより、コンパイラーがループ処理に対してベクトル化を実装しようとします。すべてのループがベクトル化の対象となるわけではありませんが、ベクトル化されたループ処理には SIMD コードが実装され、パフォーマンスが飛躍的に向上する場合があります。また、インテル® C++ コンパイラーには、このベクトル化処理の結果を表示させるオプションが用意されており、どのループがベクトル化されたのかを確認することができます。それでは、以下のように /Qvec-report オプションを指定してコンパイルを実行し、出力内容を確認してください。

```
> icl /Qvec-report2 int_sin.c
```



上記の出力内容より、int_sin.c ファイルの 74 行目のループはベクトル化が適用されていませんが、92 行目のループに対してはベクトル化が実装されていることが確認できます。なお /Qvec-report オプションに指定されている数字は出力レベルを意味しており 0 から 5 までの数字を指定することができます。詳細は製品ドキュメントを参照してください。

では作成されたプログラムを実行してパフォーマンスを確認してみましょう。

３－４．実行/パフォーマンスの比較

次のように、最適化された int_sin プログラムを実行します。

> int_sin.exe

最適化を行わなかった場合と、実行時間（プロセッサ時間）を比較します。約 4.5 倍も実行速度が向上していることが確認できます。

```
                :
                :
---------------------------------------
    16777216        |    4.000000e+000    |
---------------------------------------
    33554432        |    4.000000e+000    |
---------------------------------------
    67108864        |    4.000000e+000    |

    Application Clocks     = 1.840000e+003
```

Note：このデフォルト最適化オプション /arch:SSE2 を使用して作成した実行バイナリーは、SSE2 命令を搭載しないプロセッサー上では動作しません。現在ではほとんどのプロセッサーがこの命令を所有していますが、例えば、インテル® Pentium III などの SSE2 命令を持たない古いプロセッサー上で実行させる場合は、以下のように /arch:IA32 オプションを指定してコンパイルする必要があります。この指定で、デフォルトの /arch:SSE2 オプションは、/arch:IA32 オプションに上書きされ、x86/x87 命令を使用した汎用コードが生成されます。

> icl /arch:IA32 int_sin.c

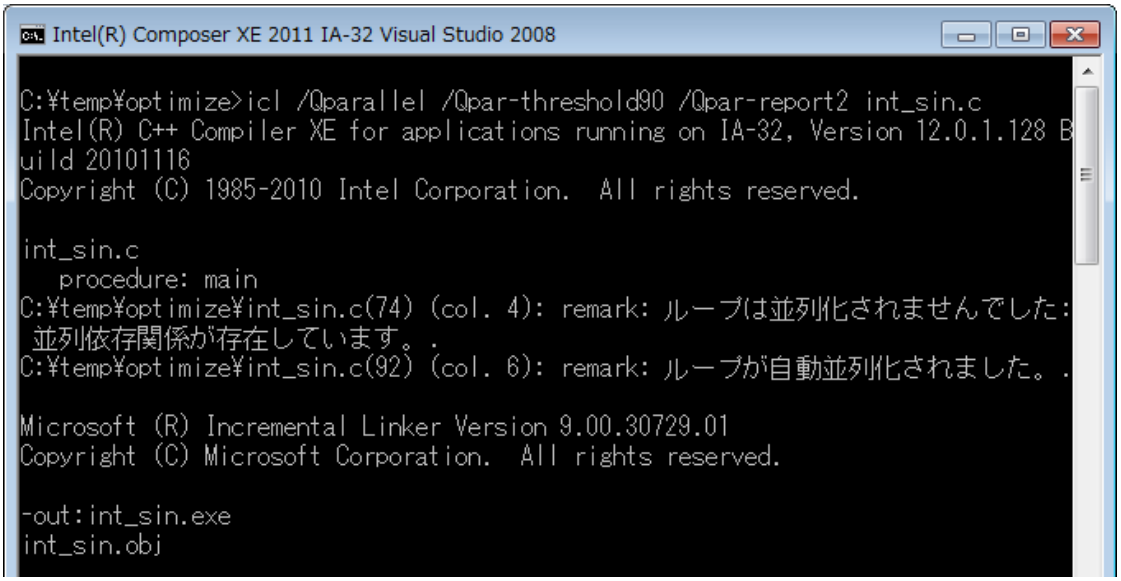
３－５．コンパイル（並列化オプションあり）

ここでは、インテル® C++ コンパイラーの自動並列化オプション（/Qparallel）を適用します。このオプションはプログラム内のループ処理に対してマルチスレッドを実装し、各スレッドに処理を分散させて効率よくプログラムを実行するための機能です。この自動並列化の機能では、コンパイラーはコンパイル時に各ループ処理に対して分析を行い、ループの実行回数や処理の大きさ、複雑性などをチェックして、安全に並列化が実装できるループ、また並列化によって高い効果が見込まれるループに対してのみ、この機能を適用します。それではこの機能を使用してサンプルプログラムをコンパイルしますが、ここでも結果レポートを出力させるオプション（/Qpar-report）を指定して次のようにコンパイルします。なお、/Qpar-report に対しても出力レベルを設定することができます。指定可能な出力レベルの範囲は 0 から 3 です。

> icl /Qparallel /Qpar-report2 int_sin.c

自動並列化機能のレポートを見ると、サンプルプログラム（int_sin.c）内の２つのループは並列化されていないことが確認できます。まず、74 行目のループは処理に依存関係が存在するためコンパイラーは安全に並列化を実施することができず、また 92 行目のループはループ内の計算量が不足しているため並列化の効果が見込まれないと判断されたようです。一般的に依存関係が存在するループを並列化するにはソースコードを見直す必要がありますが、92 行目のループのように効率性能により並列化対象とならなかったループに対しては、コンパイラーの性能評価の閾値を調節するオプション（/Qpar-threshold）を使用して並列化の実装を再度試みることができます。このオプションには閾値（0 から 100）を指定する必要があり、デフォルトでは 100 の値が設定されています。この閾値を下げることでコンパイラーの性能チェックによる制限を弱めることができます。ここでは、この閾値に 90 を設定して再度以下のようにコンパイルを実行します。

```
> icl /Qparallel /Qpar-threshold90 /Qpar-report2 int_sin.c
```



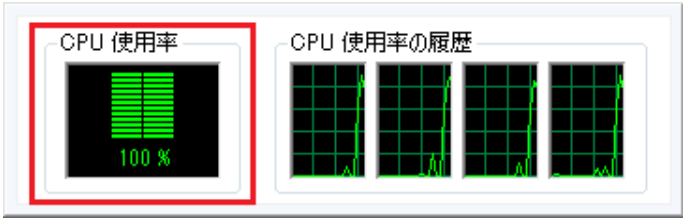
結果レポートを確認すると、今度は 92 行目のループが自動並列化されたことが分かります。それでは作成されたバイナリーを実行してパフォーマンスを見てみましょう。

３－６．実行/パフォーマンスの比較

自動並列化が施された int_sin.exe プログラムを実行し、前回の結果と比較してください。自動並列化することにより、ここでは、さらに 2.7 倍近い速度向上が得られていることが確認できます。また実行に際して、[タスク マネージャー] を起動して CPU 使用率もチェックしてみますと、CPU 使用率は 100% を示しており、すべてのコアを使用して処理が行われていることが確認できます。

```
> int_sin.exe
                :
                :
                :
                :
-------------------------------------
     8388608        |    4.000000e+000    |
-------------------------------------
    16777216        |    4.000000e+000    |
-------------------------------------
    33554432        |    4.000000e+000    |
-------------------------------------
    67108864        |    4.000000e+000    |

    Application Clocks      = 6.860000e+002
```



本サンプルプログラム（int_sin.c）を自動並列化すると実行時間が短すぎて CPU 使用率を観察しづらくなります。その場合は、サンプルプログラム内の以下のループ回数を 27 から 30 程度に変更してお試しください。

```
// for (j=2;j<27;j++)
    for (j=2;j<30;j++)
    { …
```

Note：自動並列化されたループはマルチスレッド化され、マルチコアやマルチプロセッサーの CPU リソースを効率よく使用して高いパフォーマンスが期待できます。本ドキュメントでは、Intel® Core™ 2 Quad プロセッサーを使用しており、４つのコアを使用してこの自動並列化されたサンプルプログラムを実行していますが、それ以上のコア数やプロセッサー数を搭載したシステムでは更なるパフォーマンス向上が見込まれる可能性があります。

Note：コマンドラインでコンパイル、リンクを実行すると、リンクされるランタイムライブラリーはデフォルトで、静的ライブラリー（/MT）が指定されます。これを動的ライブラリーに変更する場合は、/MD オプションを指定する必要があります。インテル® C++ コンパイラーの自動並列化を使用する場合は、指定されるランタイムライブラリーによりパフォーマンスが異なる場合があります。一度、以下のオプションで実行速度を検証することをお勧めします。

```
> icl /Qparallel /Qpar-threshold90 /MD int_sin.c
```

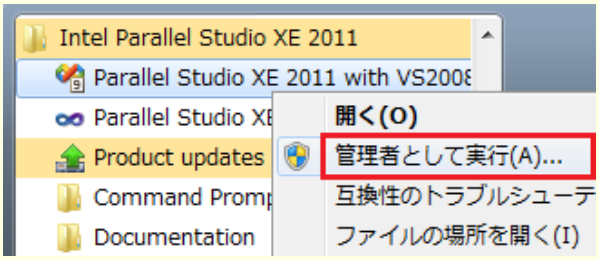
4． Microsoft Visual Studio IDE からのビルド方法

本章では、インテル® C++ コンパイラーを Microsoft Visual Studio 環境で使用する手順を説明します。本ドキュメントで使用しているサンプルプログラムには Visual Studio のプロジェクトが含まれていますが、ここではあえてプロジェクトを新規作成し、ビルド環境の設定を行います。また本製品は、使用する Microsoft Visual Studio のバージョンによって多少操作方法が異なりますので、ここでは Visual Studio 2008（以下、VS2008）と Visual Studio 2010（以下、VS2010）からのビルド手順について説明します。Visual Studio 2005 については Visual Studio 2008 と操作手順は同じですので Visual Studio 2008 からのビルド方法を参照してください。それでは以下に、Microsoft Visual Studio 環境からのビルド手順を記します。

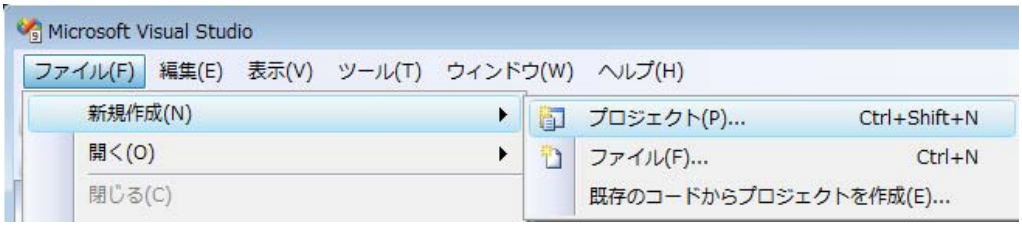
4－1． Visual Studio 2008 からのビルド

1. まず、Windows [スタート] メニューから [Intel Parallel Studio XE 2011] – [Parallel Studio XE 2011 with VS2008] を起動します。または、同じく [スタート] メニューから [Microsoft Visual Studio 2008] – [Microsoft Visual Studio 2008] を起動しても構いません。

ご注意：ビルドが正常終了しない場合は、下図のようにショートカットを右クリックして表示されるメニューから [管理者として実行] を選択して再度お試しください。

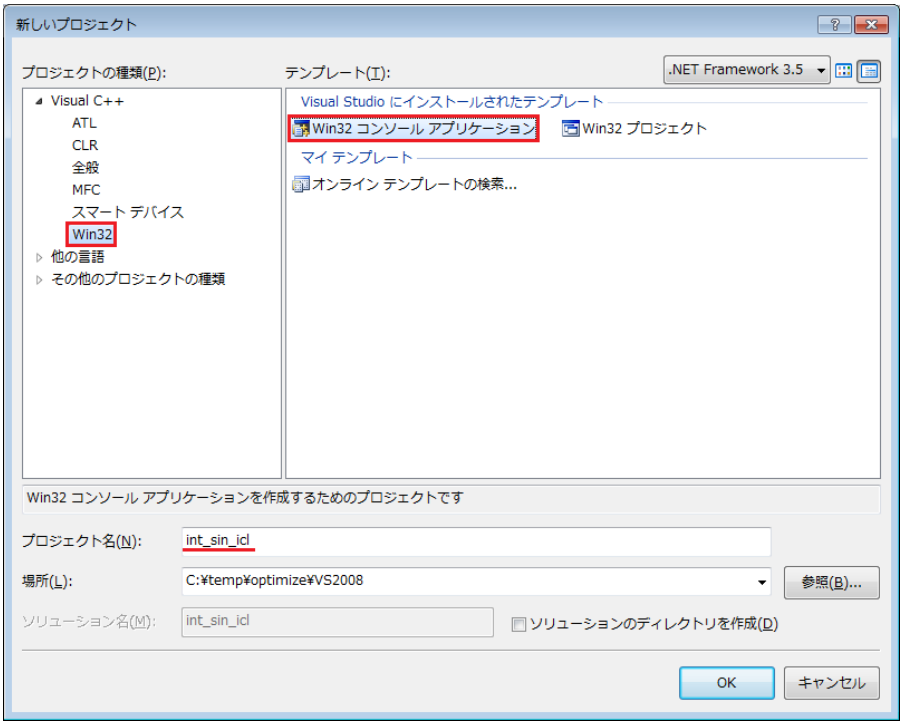


2. 新規プロジェクトを作成します。VS2008 のメニューから、[ファイル] - [新規作成] - [プロジェクト] を選択して [新しいプロジェクト] ダイアログを表示します。



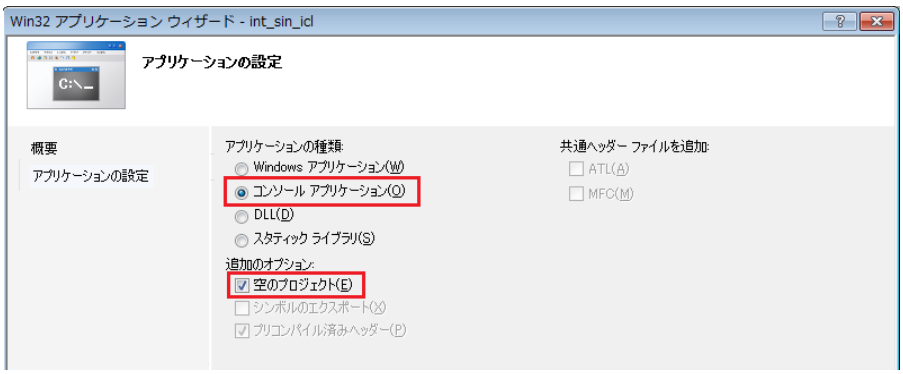
下図に示すように [プロジェクトの種類] で [Win32] を選択し、[テンプレート] で [Win32 コンソールアプリケーション] を選択します。プロジェクト名として int_sin_icl を指定して [OK] ボタンをクリックします。なお、プロジェクトを作成する“場所”は任意で構いませんが、ここでは、“C:¥temp¥optimize¥VS2008” を指定しています。

図：新しいプロジェクト

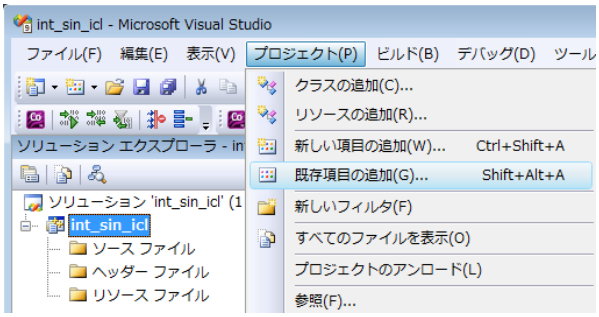


[アプリケーションの設定] 画面では、下図のように [空のプロジェクト] を選択してください。

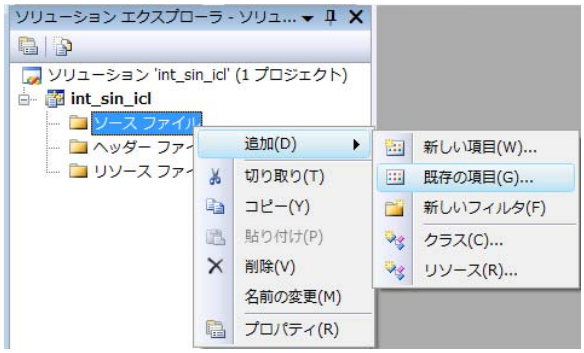
図：アプリケーションの設定



3. 作成したプロジェクトにサンプルコード（int_sin.c）を追加します。メニューから [プロジェクト] - [既存項目の追加...] を選択するか、または [ソリューション エクスプローラ] から “ソースファイル” を右クリックして表示されるメニューから [追加] - [既存の項目] を選択します。



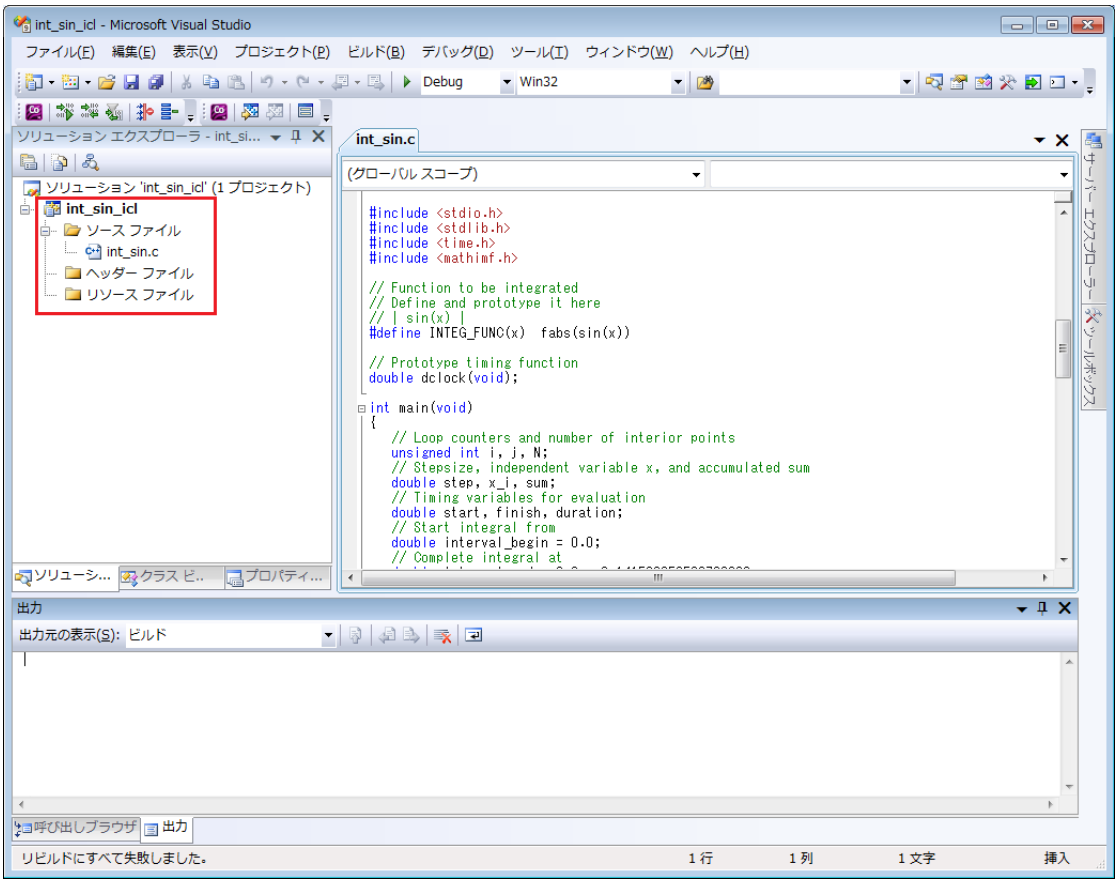
または、



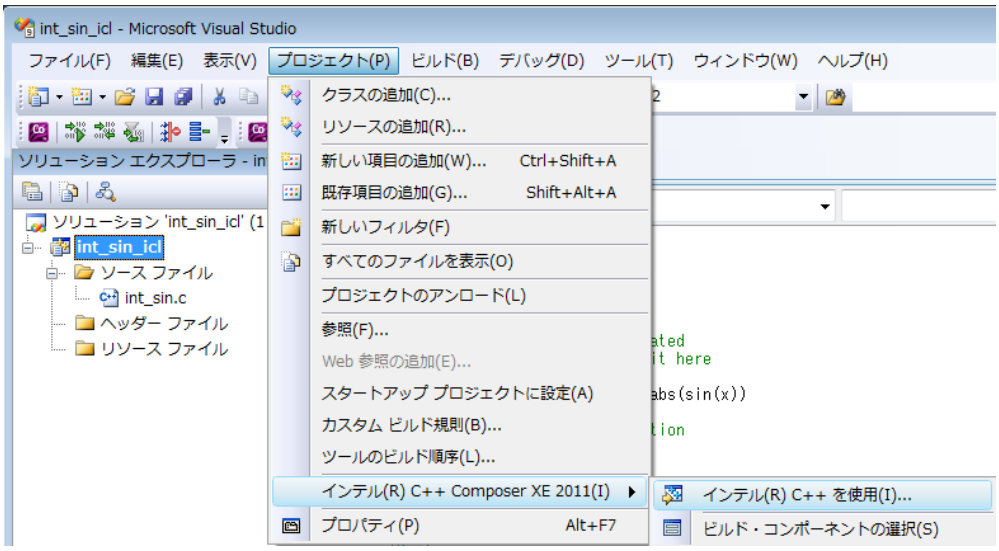
表示される [既存項目の追加] ダイアログで以下のサンプルコードを選択して [追加] ボタンをクリックします。

C:¥temp¥optimize¥int_sin.c

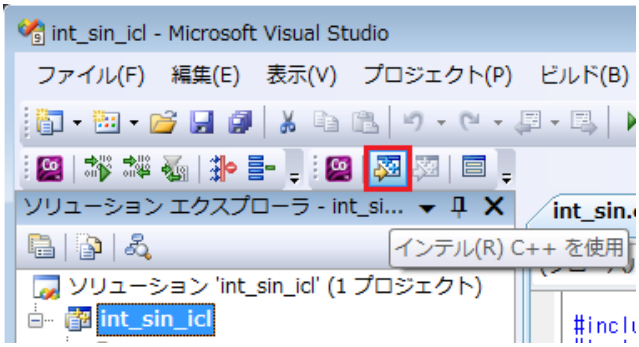
4. 新しいプロジェクト int_sin_icl の “ソースファイル” に、サンプルコード “int_sin.c” が追加されたことを確認します。



5. 次に、使用するコンパイラーの切り替えを行います。 [プロジェクト] メニューから [インテル(R) C++ Composer XE 2011] - [インテル(R) C++ を使用] を選択するか、またはインテル(R) C++ ツールバーの [インテル(R) C++ を使用] ボタンをクリックします。

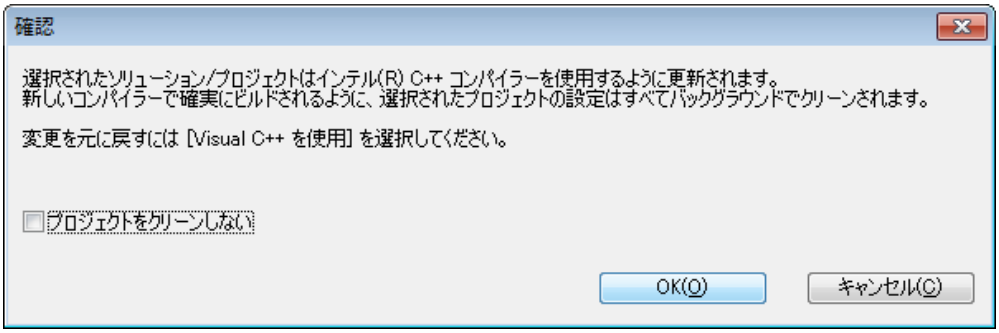


または



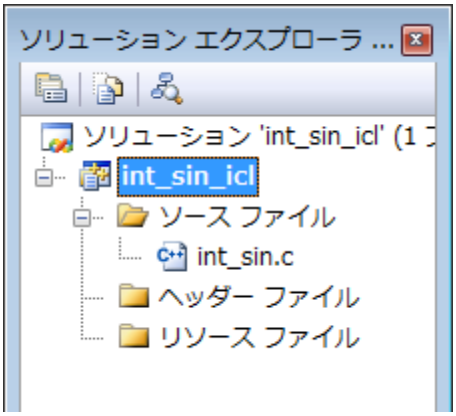
> Note： [ソリューション エクスプローラ] からプロジェクトを右クリックして表示されるメニューから、[インテル(R) C++ Composer XE 2011] - [インテル(R) C++ を使用] を選択して使用するコンパイラーを切り替えることもできます。

表示される [確認] ダイアログで [OK] をクリックします。デフォルトでは切り替え時にプロジェクトのクリーンが実行されます。

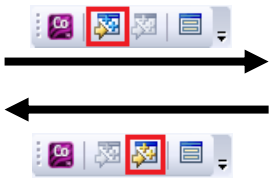


切り替えが成功すると、[ソリューション エクスプローラ] にインテル® C++ プロジェクトが追加されます。このコンパイラーの切り替えは Microsoft Visual C++ コンパイラーと インテル® C++ コンパイラー間で自由に行うことができます。また、この切り替えにより、インテル® C++ プロジェクト・ファイル (.icproj) が作成されインテル® C++ コンパイラーの設定内容が管理されます。なお、既存の Microsoft Visual C++ プロジェクト・ファイル（.vcproj）の内容を変更することはありません。

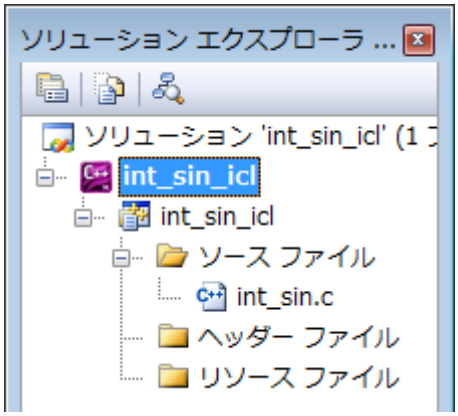
図：Microsoft Visual C++ プロジェクト



コンパイラーの切り替え



図：インテル® C++ プロジェクト

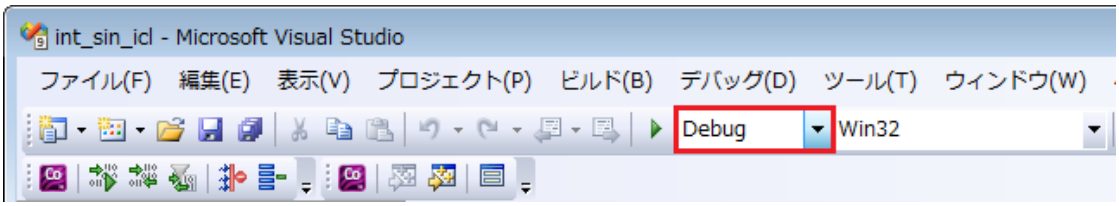


Note：インテル® C++ コンパイラーでは、.NET プロジェクトのようなマネージドコードを生成するプロジェクトをサポートしていないため、これらのプロジェクトに対するコンパイラーの切り替えはできません。

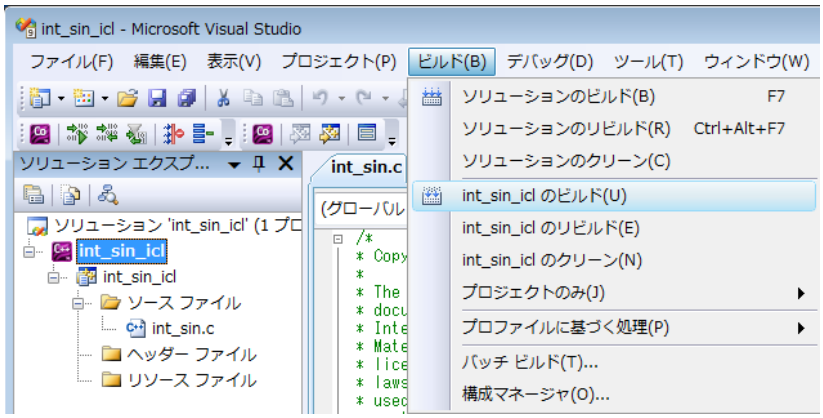
4－1－1．ビルド（最適化オプションなし）

まず、最適化オプションなしでビルドを行います。次の手順を実行します。

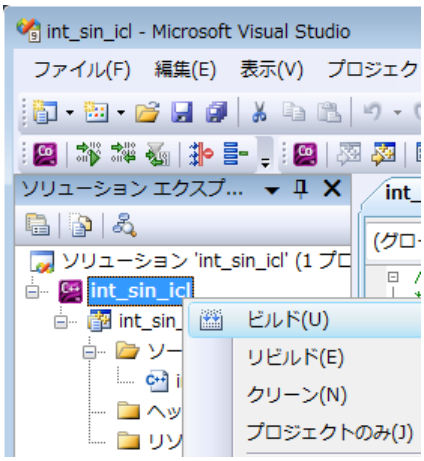
1. プロジェクトの構成が、"Debug" 構成であることを確認してください。

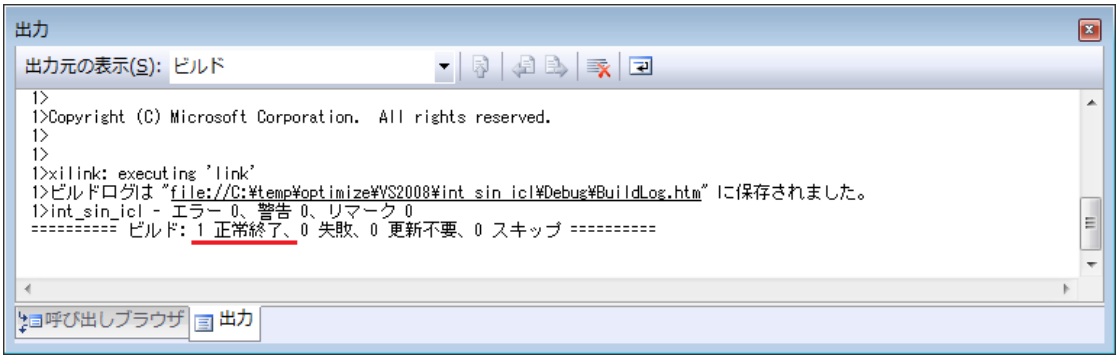


2. 次にプロジェクトのビルドを行います。VS2008 のメニューから、[ビルド] - [int_sin_icl のビルド] を選択するか、または [ソリューション エクスプローラ] からプロジェクトを右クリックして表示されるメニューから、[ビルド] を選択します。ビルドが完了するとビルド結果が表示されるので、正常終了していることを確認してください。



または

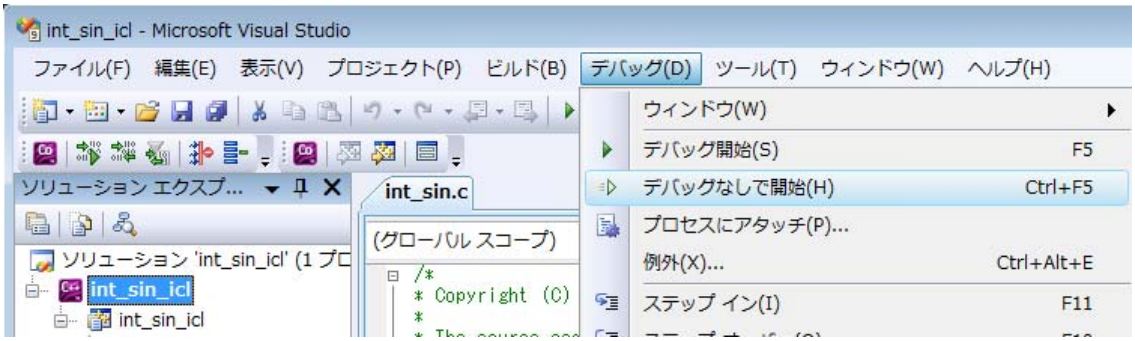


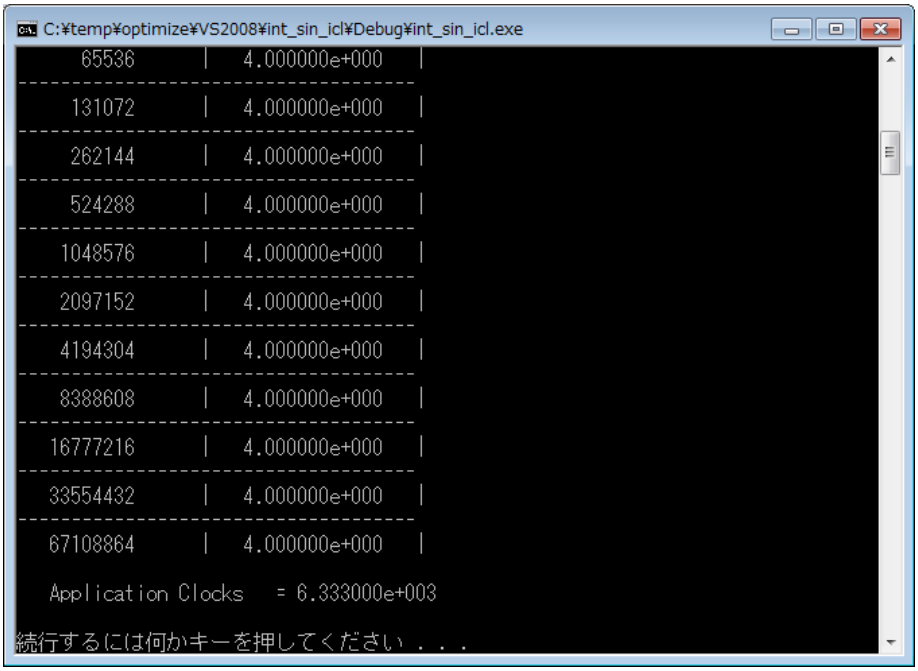


Note : プロジェクトには通常、Debug 構成と Release 構成という 2 種類のプロジェクト構成（ビルド設定環境）が用意されています。一般的に開発中のプロジェクトは Debug 構成で作業を行い、開発が完了した製品を Release 構成でビルドします。デフォルトのプロジェクト構成は Debug 構成で、プロジェクトは最適化なしで、シンボリック・デバッグ情報付きでビルドされます。これはコマンドラインから、 icl /Od /Zi int_sin.c と入力した場合とほぼ同じです。

４－１－２．実行/プログラムの検証

1. VS2008 メニューから、[デバッグ] - [デバッグなしで開始] を選択します。コマンドウィンドウにプログラムの実行結果が表示されます。



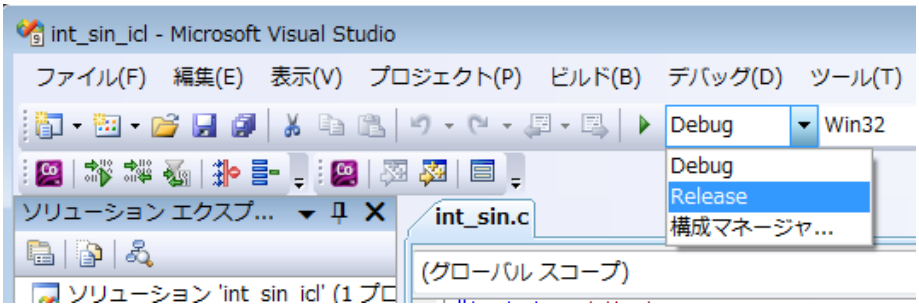


2. プログラム実行にかかった CPU 時間をメモします。

４－１－３．ビルド（最適化オプションあり）

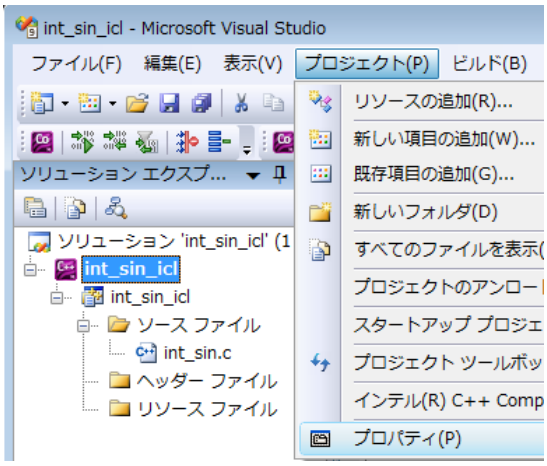
次に、最適化オプションを使用してビルドを行います。次の手順を実行します。

1. プロジェクトの構成を、”Release” 構成に変更してください。

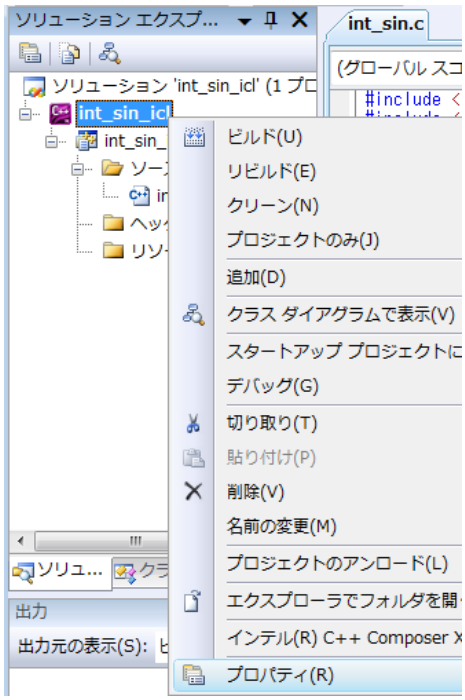


“Release” モードのプロジェクト構成では、インテル® C++ コンパイラーのデフォルトの最適化オプション（/O2 および /arch:SSE2）が有効となります。プロジェクトの [プロパティ ページ] で確認してみましょう。

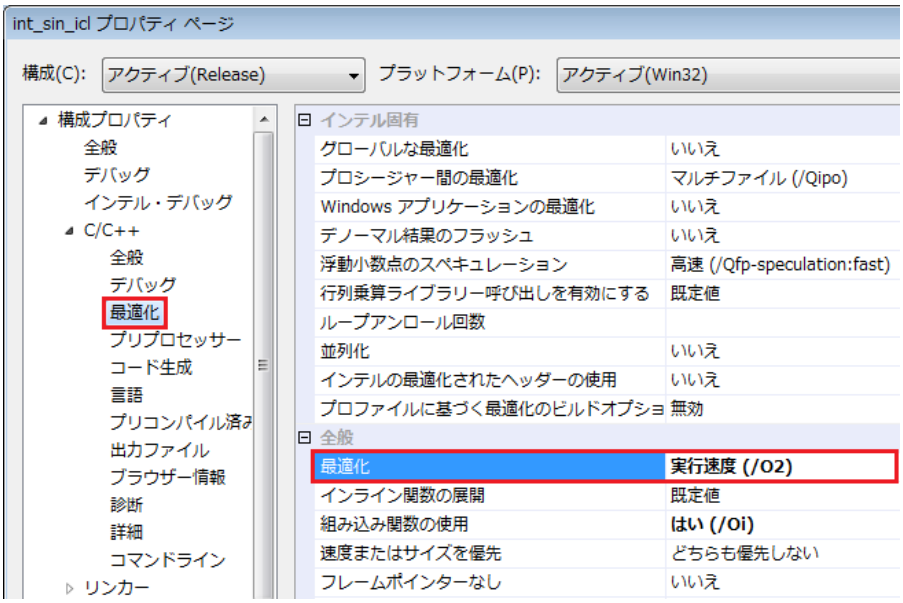
2. VS2008 メニューから [プロジェクト] - [プロパティ] を選択します。または、[ソリューション エクスプローラ] からプロジェクト “int_sin_icl” を右クリックして [プロパティ] を選択します。



または

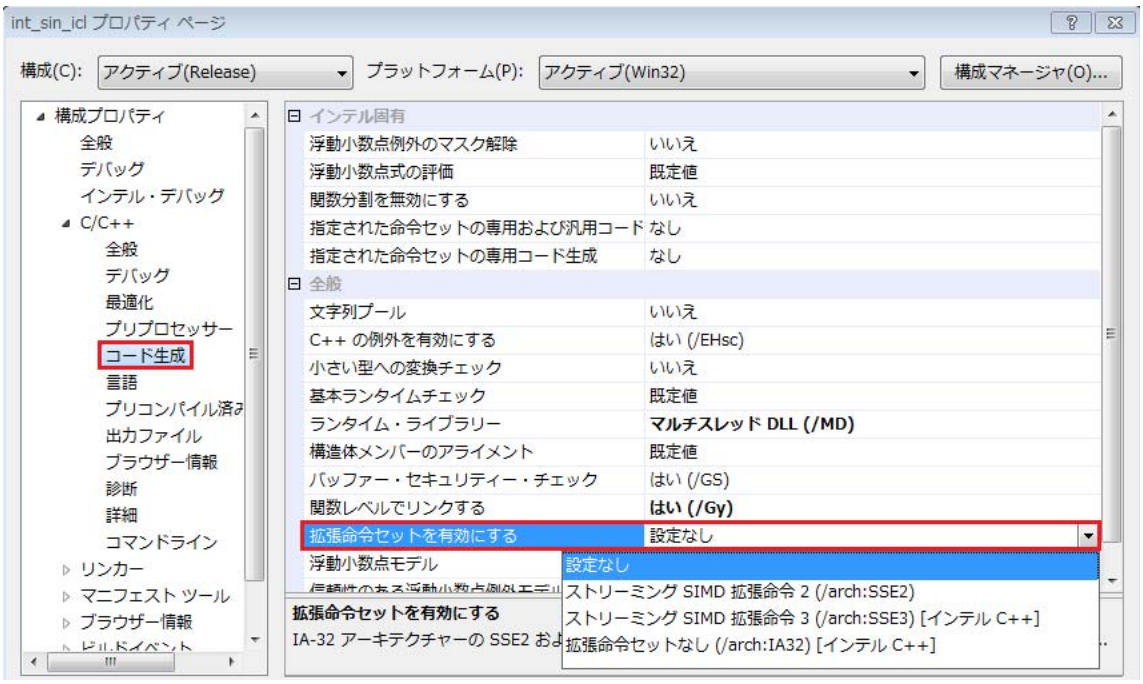


3. 表示される [プロパティ ページ] の左のペインから [構成プロパティ] - [C/C++] - [最適化] を選択して、[最適化] が “実行速度（/O2）” に設定されていることを確認します。

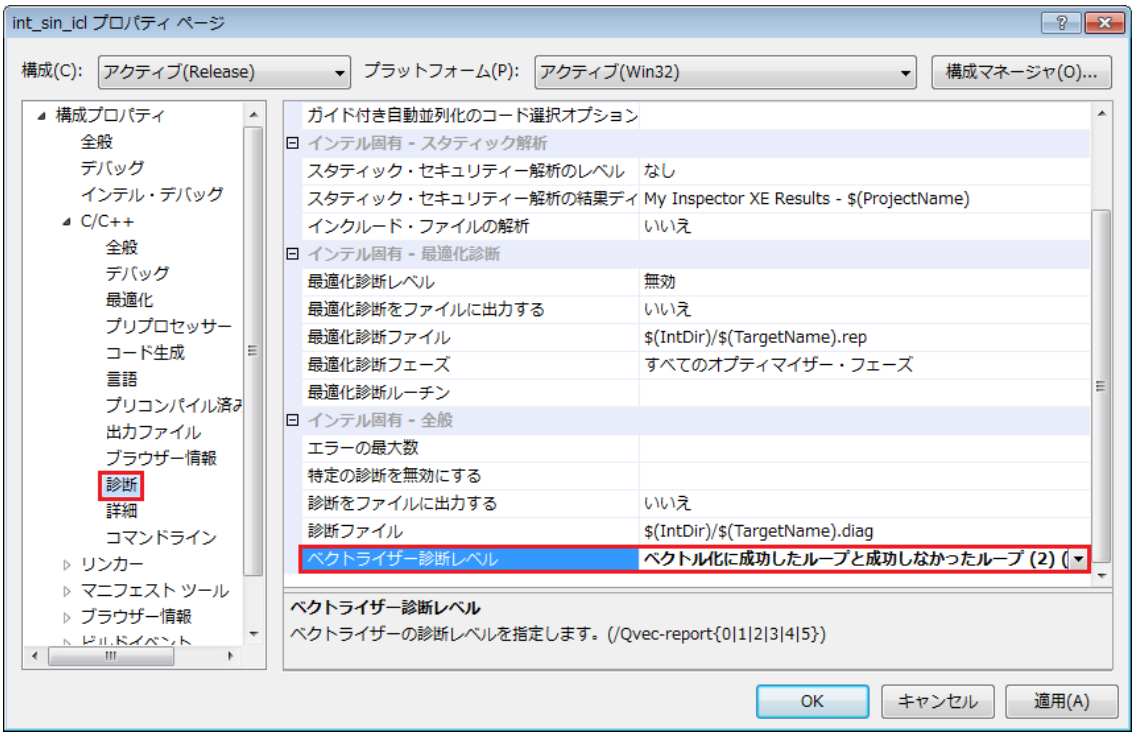


Note：前節の Debug 構成では、この最適化の値は “無効（/Od）” に設定されています。

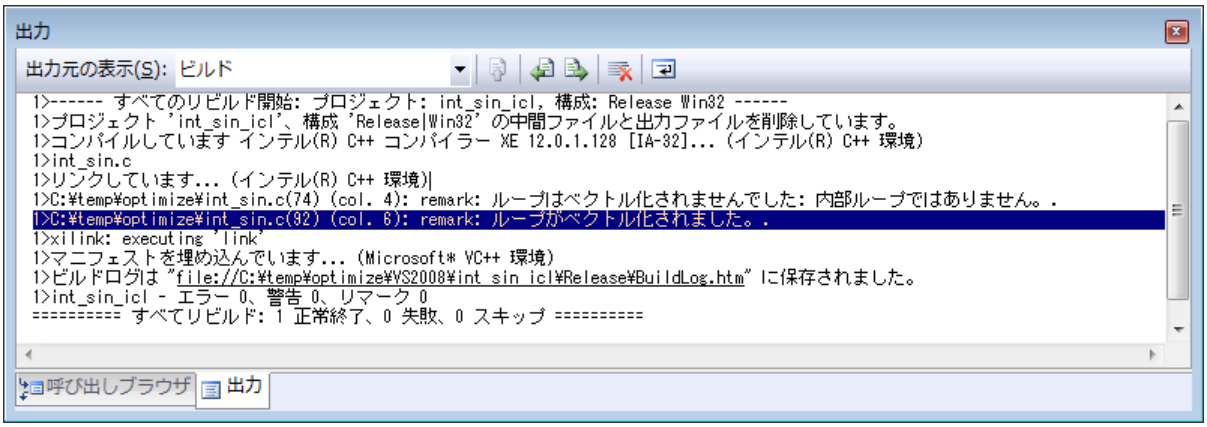
また、/arch:SSE2 のオプションは [構成プロパティ] - [C/C++] - [コード生成] の [拡張命令セットを有効にする] の項目に存在します。この項目の設定値は “設定なし” と指定されていますが、/O2 が指定されている場合は、コンパイラーはデフォルトで /arch:SSE2 オプションを有効にします。このオプションを無効にする場合は、 “拡張命令セットなし（/arch:IA32）[インテル C++]” を選択します。



また、/arch:SSE2 オプションの効果を確認するために、「３－３．コンパイル（最適化オプションあり）」で説明した /Qvec-report オプションを指定します。[構成プロパティ] - [C/C++] - [診断] から [ベクトライザー診断レベル] の項目に、[ベクトル化に成功したループと成功しなかったループ（２）（/Qvec-report2）] を選択します。

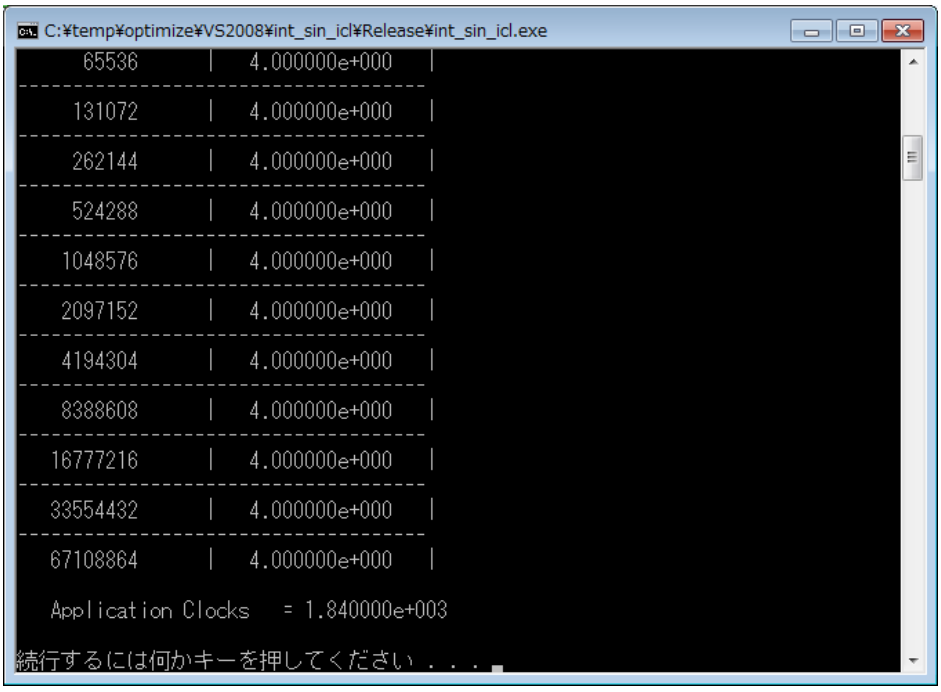


4. VS2008 のメニューから [ビルド] - [int_sin_icl のビルド] を選択して、“Release” 構成で int_sin_icl プロジェクトをビルドします。 表示されるレポート内容を確認してベクトル化の適用状況を確認します。



４－１－４．実行/パフォーマンスの比較

1. VS2008 のメニューから、[デバッグ] - [デバッグなしで開始] を選択します。コマンドウィンドウに最適化されたプログラムの実行結果が表示されます。

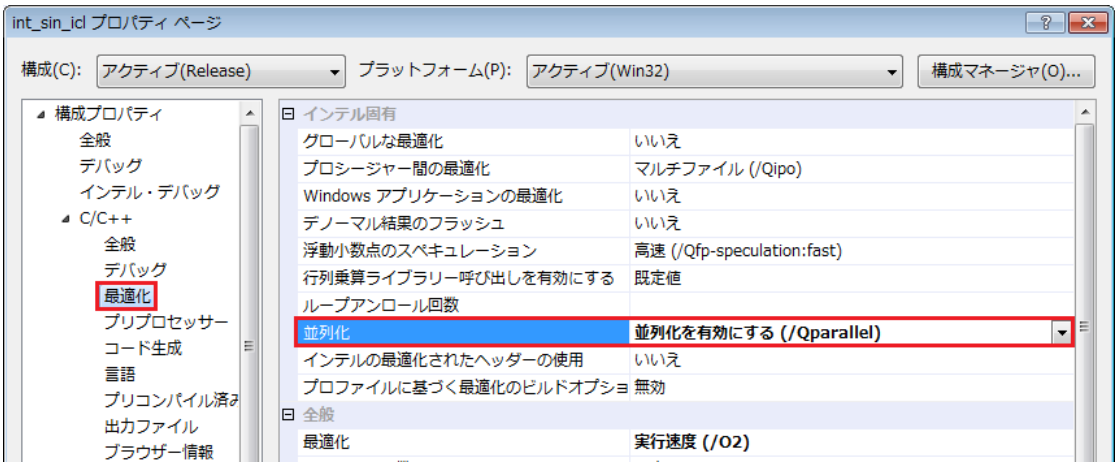


2. 最適化を行った場合の CPU 時間をメモして、最適化を行わなかった場合と比較します。ここでの結果では、約 3.5 倍の速度向上が確認できます。

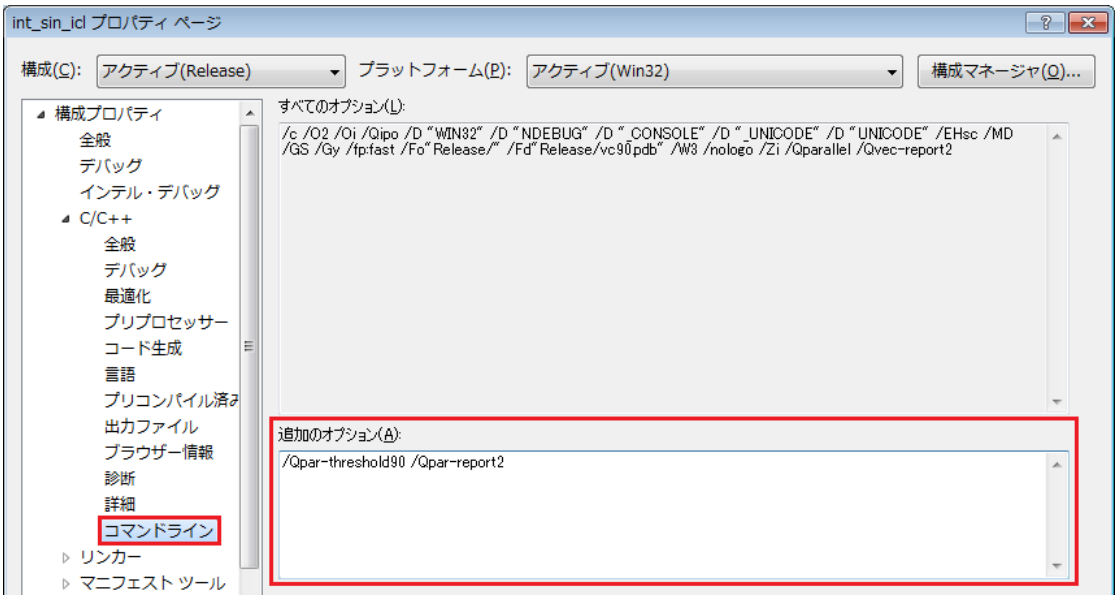
４－１－５．ビルド（並列化オプションあり）

コマンドライン同様、ここでも自動並列化オプション(/Qparallel)を使用してパフォーマンスを見ていきます。「３－５．コンパイル（並列化オプションあり）」の章で説明したとおり、/Qpar-threshold オプションが必要になります。また、結果レポートを表示させる /Qpar-report オプションも同様に指定します。

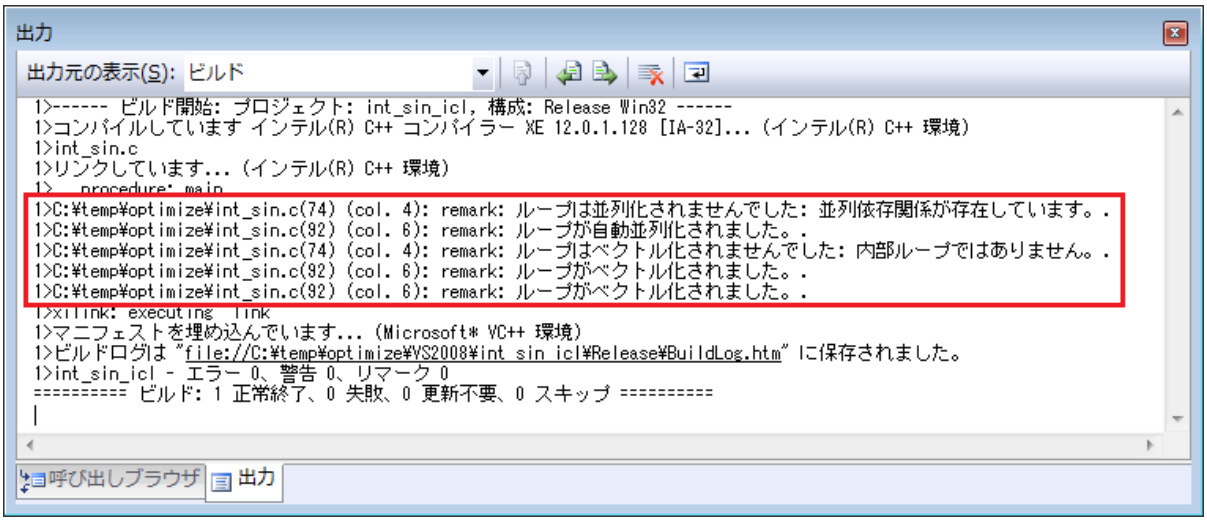
1. まず、プロジェクトの [プロパティ ページ] を開き、[構成プロパティ] - [C++] - [最適化] を選択して下図のように [並列化] の値を“並列化を有効にする（/Qparallel）”に設定します。



また、/Qpar-threshold と /Qpar-report オプションは、[プロパティ ページ] に指定する項目がないので、[構成プロパティ] - [C++] - [コマンドライン] の [追加のオプション] 欄に手書きで記入します。以下のように、/Qpar-threshold90 /Qpar-report2 と２つのオプションをスペースで区切って入力してください。

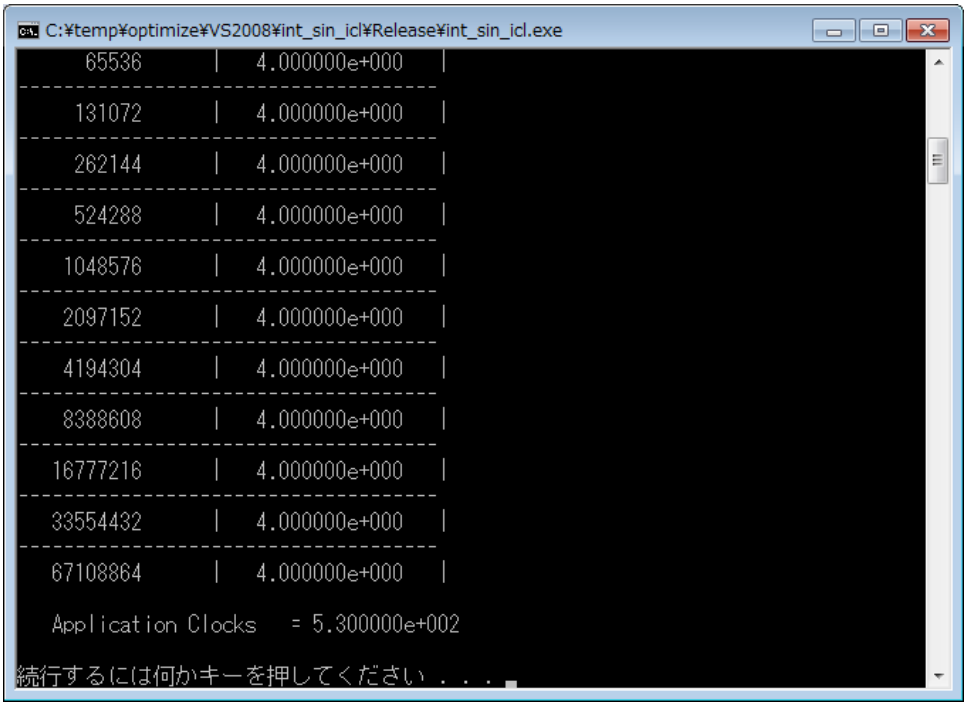


2. オプションの設定が完了したら、[ビルド] - [int_sin_icl のビルド] を選択して、int_sin_icl プロジェクトをビルドします。 結果レポートを確認してください。



４－１－６．実行/パフォーマンスの比較

1. VS2008 のメニューから、[デバッグ] - [デバッグなしで開始] を選択してプログラムを実行します。



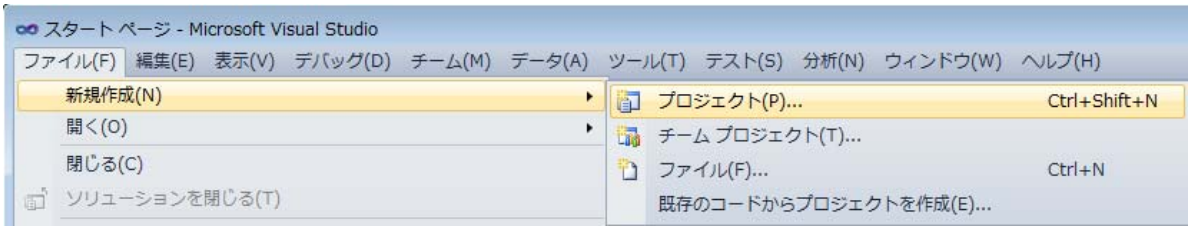
2. 並列化を行った場合の CPU 時間をメモして、結果を比較します。並列化を行わなかった結果と比較すると、ここでは約 3.5 倍のパフォーマンスが得られています。

4－2．Visual Studio 2010 からのビルド

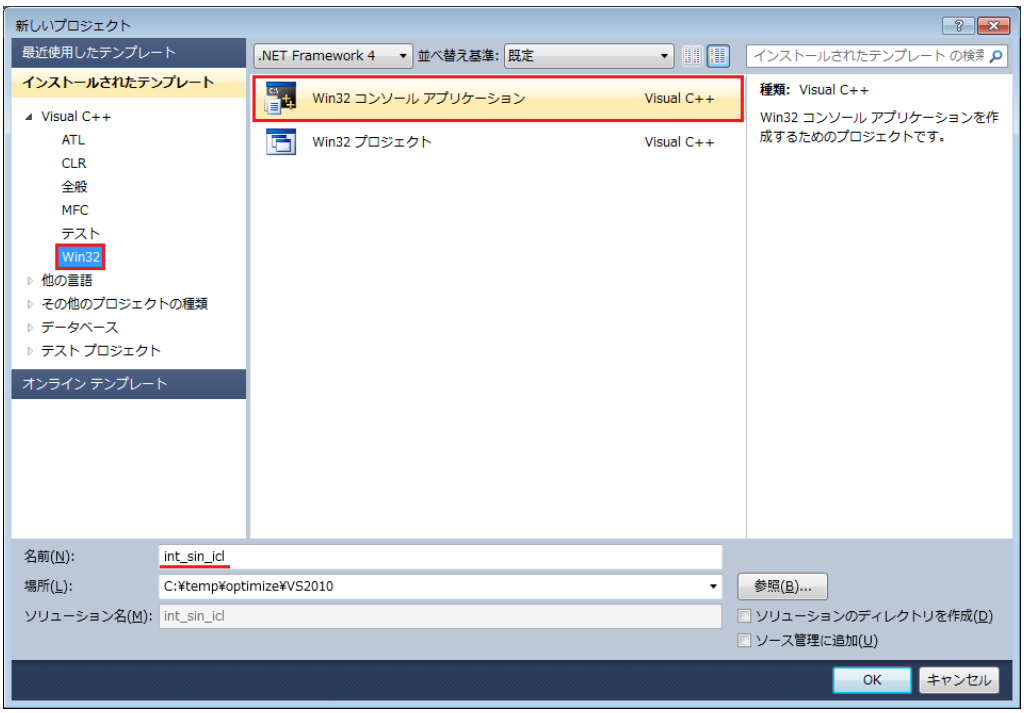
1. まず、Windows [スタート] メニューから [Intel Parallel Studio XE 2011] - [Parallel Studio XE 2011 with VS2010] を起動します。または、同じく [スタート] メニューから [Microsoft Visual Studio 2010] - [Microsoft Visual Studio 2010] を起動しても構いません。

ご注意：ビルドが正常終了しない場合は、下図のようにショートカットを右クリックして表示されるメニューから [管理者として実行] を選択して再度お試しください。

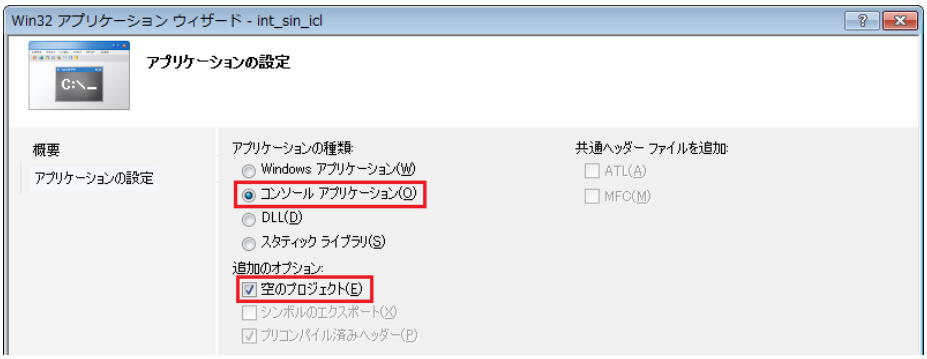
2. 新規プロジェクトを作成します。VS2010 のメニューから、[ファイル] - [新規作成] - [プロジェクト] を選択して [新しいプロジェクト] ダイアログを表示します。



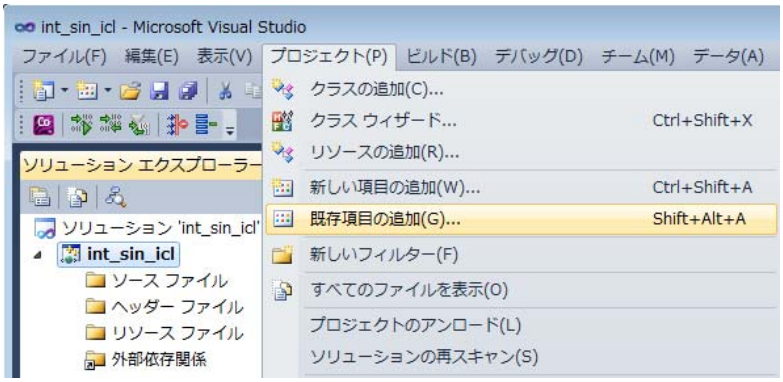
下図に示すように [Visual C++] - [Win32] を選択し、表示される [Win32 コンソールアプリケーション] テンプレートを選択します。プロジェクト名として int_sin_icl を指定して [OK] ボタンをクリックします。なお、プロジェクトを作成する“場所”は任意で構いませんが、ここでは、“C:¥temp¥optimize¥VS2010” を指定しています。



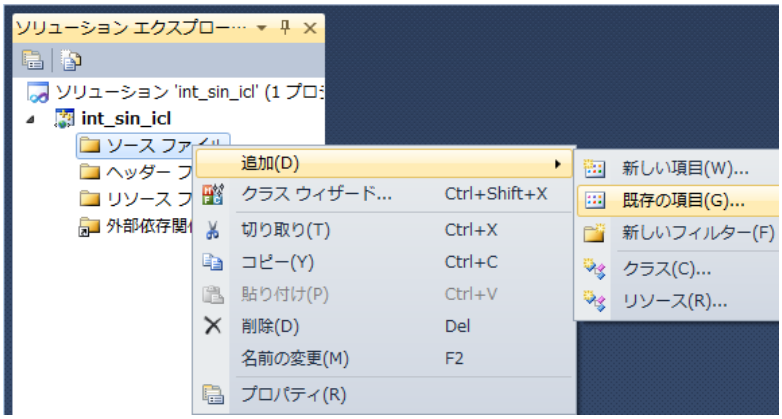
[アプリケーションの設定] 画面では、下図のように [空のプロジェクト] を選択してください。



3. 作成したプロジェクトにサンプルコード（int_sin.c）を追加します。メニューから [プロジェクト] - [既存項目の追加...] を選択するか、または [ソリューション エクスプローラ] から “ソースファイル” を右クリックして表示されるメニューから [追加] - [既存の項目] を選択します。



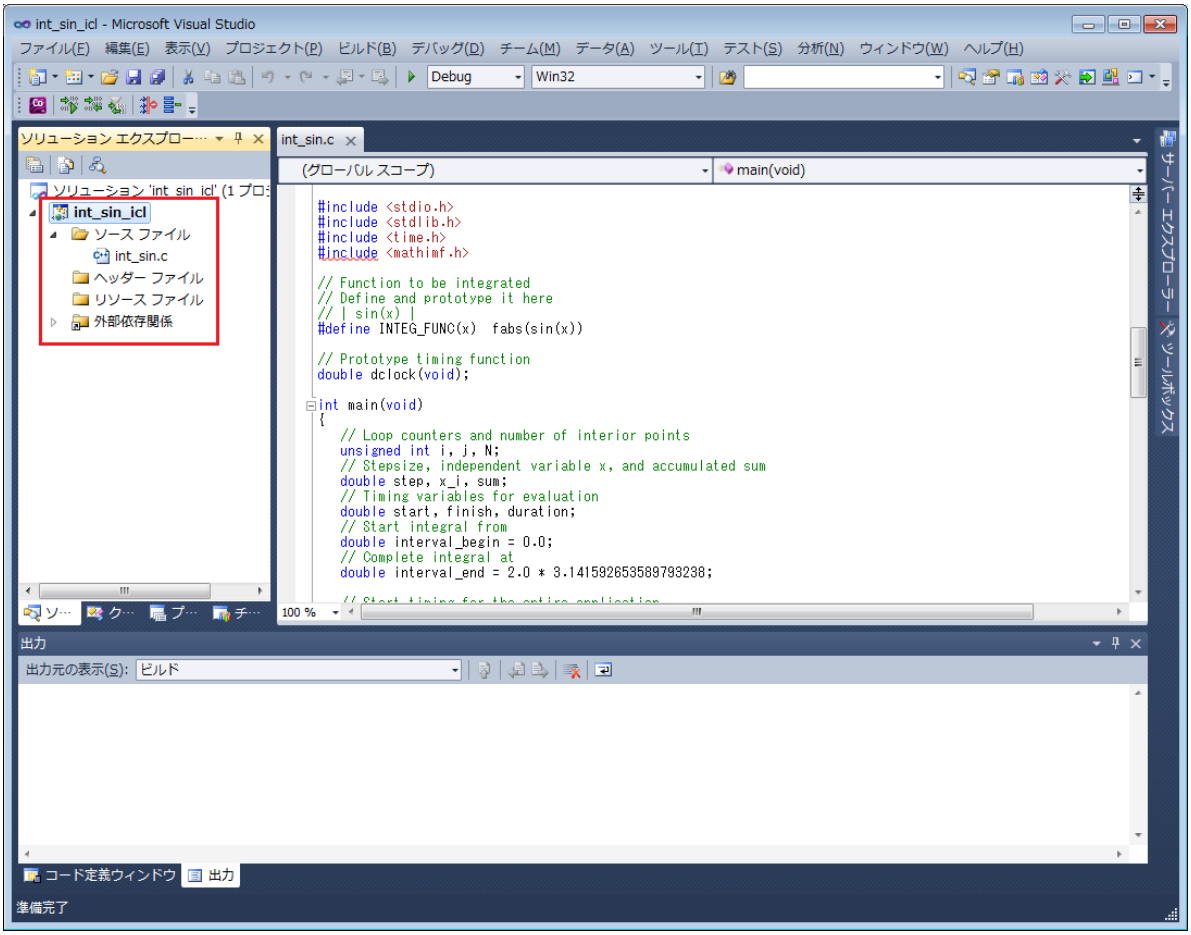
または、



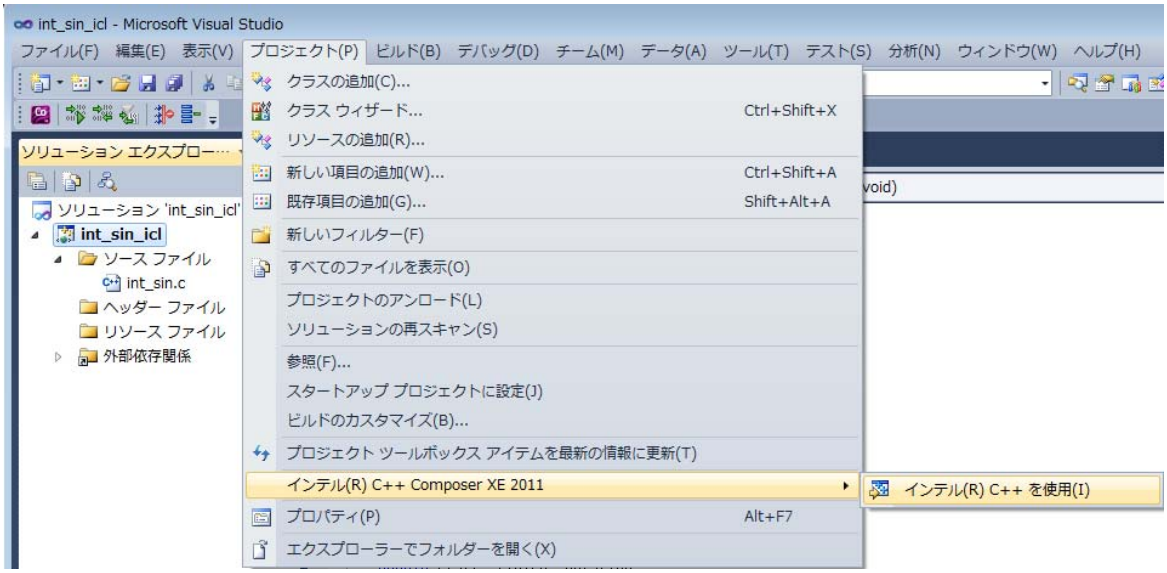
表示される [既存項目の追加] ダイアログで以下のサンプルコードを選択して [追加] ボタンをクリックします。

C:¥temp¥optimize¥int_sin.c

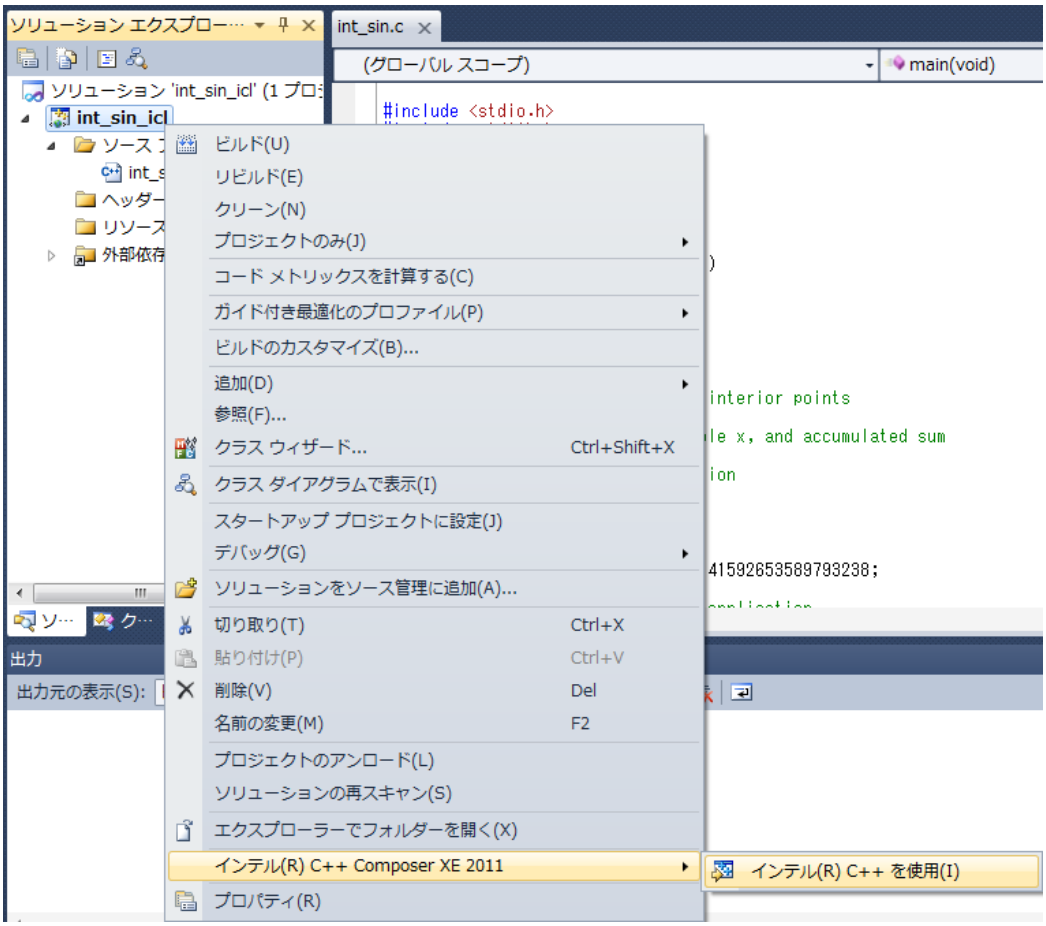
4. 新しいプロジェクト int_sin_icl の“ソースファイル”に、サンプルコード “int_sin.c” が追加されたことを確認します。



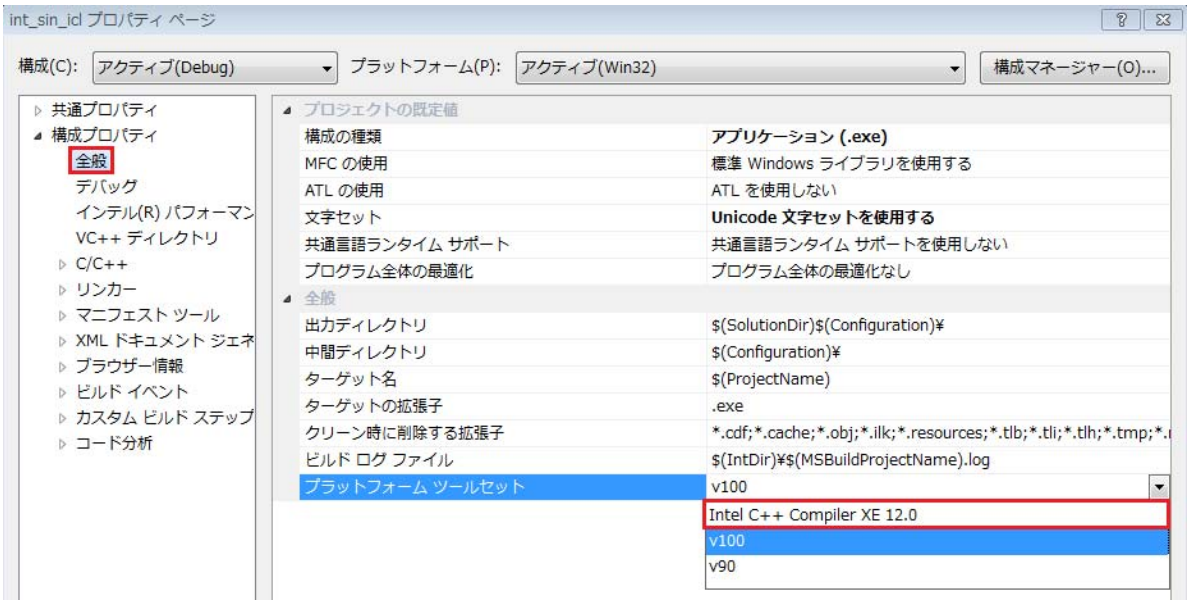
5. 次に、使用するコンパイラーの切り替えを行います。プロジェクトで使用するコンパイラーにインテル® C++ コンパイラーを指定するには、[プロジェクト] メニューから [インテル(R) C++ Composer XE 2011] - [インテル(R) C++ を使用] を選択するか、または [ソリューション エクスプローラ] からプロジェクトを右クリックして表示されるメニューから、[インテル(R) C++ Composer XE 2011] - [インテル(R) C++ を使用] を選択します。



または、

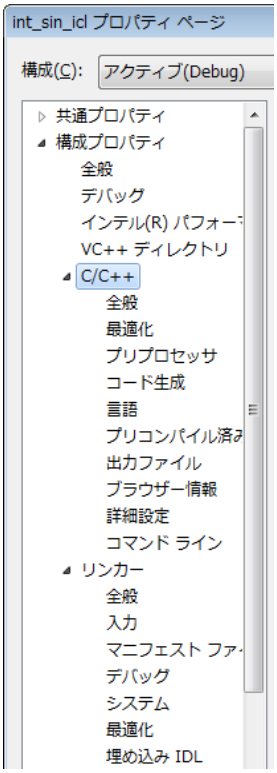


また、プロジェクトの [プロパティ ページ] からプロジェクトの構成単位に使用するコンパイラーを切り替えることもできます。インテル® C++ コンパイラーを使用する場合は、[構成プロパティ] - [全般] から [プラットフォーム ツールセット] の値を“v100”から“Intel C++ Compiler XE 12.0”に変更します。

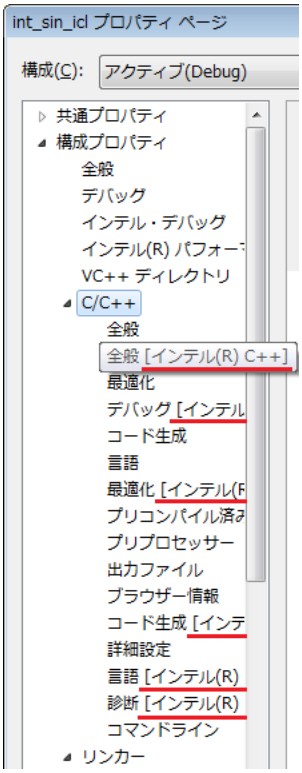


Note：インテル® C++ コンパイラーのプロパティページ情報は、Microsoft Visual C++ コンパイラー用のプロジェクト・ファイル（.vcxproj）に追記されます。

Visual C++ Compiler：



Intel C++ Compiler：

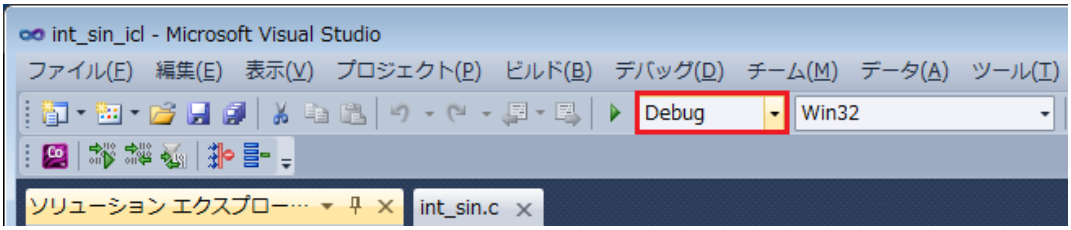


左の図のように、使用するコンパイラーをインテル® C++ コンパイラーに切り替えると、プロジェクトのプロパティページの左ペインの項目に [インテル(R) C++] の文字が付いたエントリーが追加されます。
このエントリーには、インテル® C++ コンパイラー固有のオプションがまとめられています。

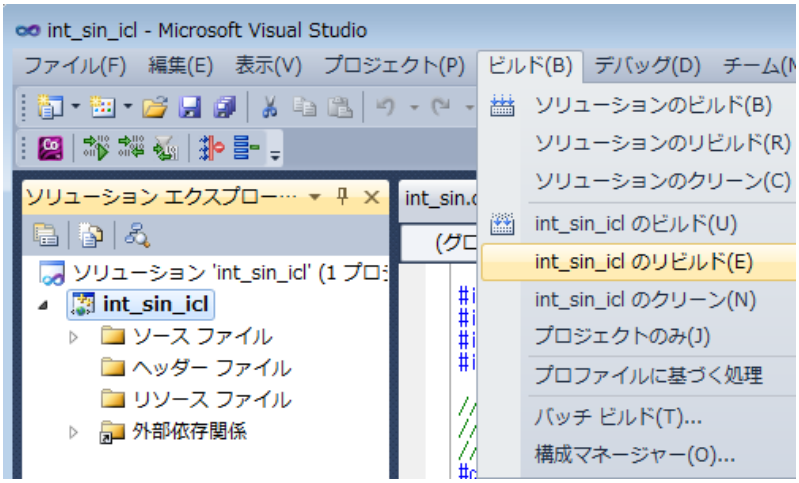
４－２－１．ビルド（最適化オプションなし）

まず、最適化オプションなしでビルドを行います。次の手順を実行します。

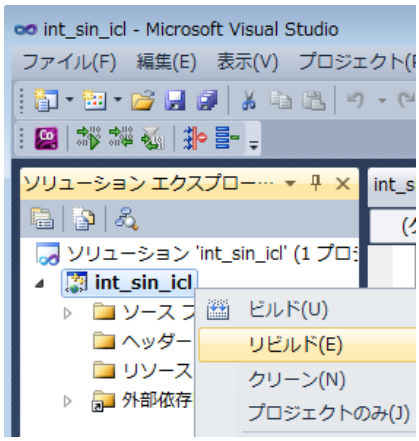
1. プロジェクトの構成が、”Debug” 構成であることを確認してください。



2. 次にプロジェクトのリビルドを行います。VS2010 のメニューから、[ビルド] - [int_sin_icl のリビルド] を選択するか、または [ソリューション エクスプローラ] からプロジェクトを右クリックして表示されるメニューから [リビルド] を選択します。リビルドが完了するとビルド結果が表示されるので、正常終了していることを確認してください。



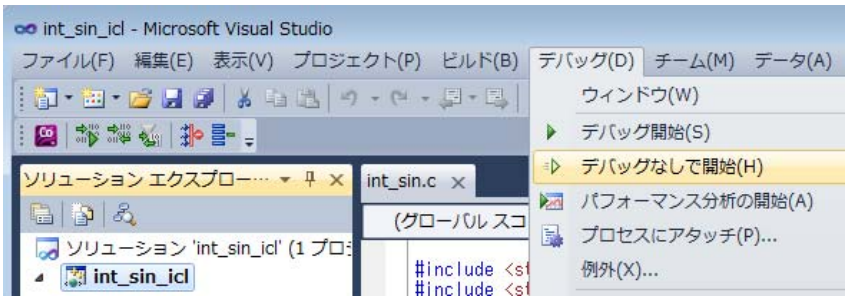
または

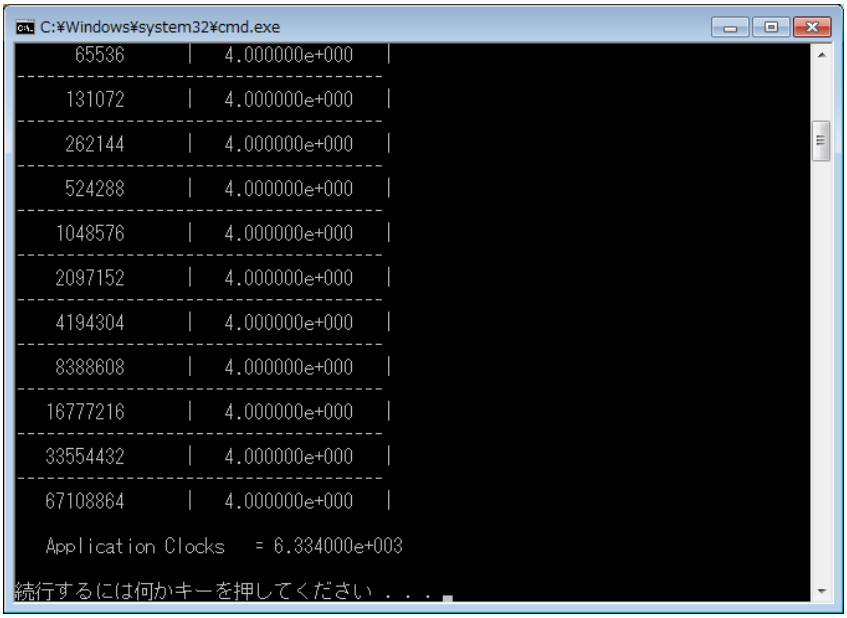


Note：デフォルトの Debug 構成では、最適化なし、デバッグ情報付きでビルドが実行されます。これはコマンドラインから、 icl /Od /Zi int_sin.c と入力した場合とほぼ同じです。

４－２－２．実行/プログラムの検証

1. VS2010 メニューから、[デバッグ] - [デバッグなしで開始] を選択します。コマンドウィンドウにプログラムの実行結果が表示されます。



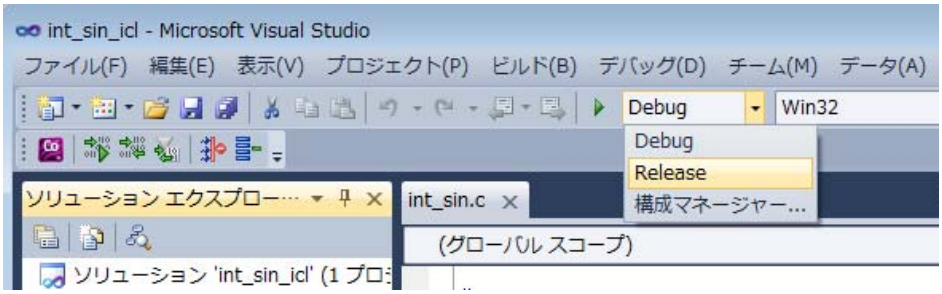


2. プログラム実行にかかった CPU 時間をメモします。

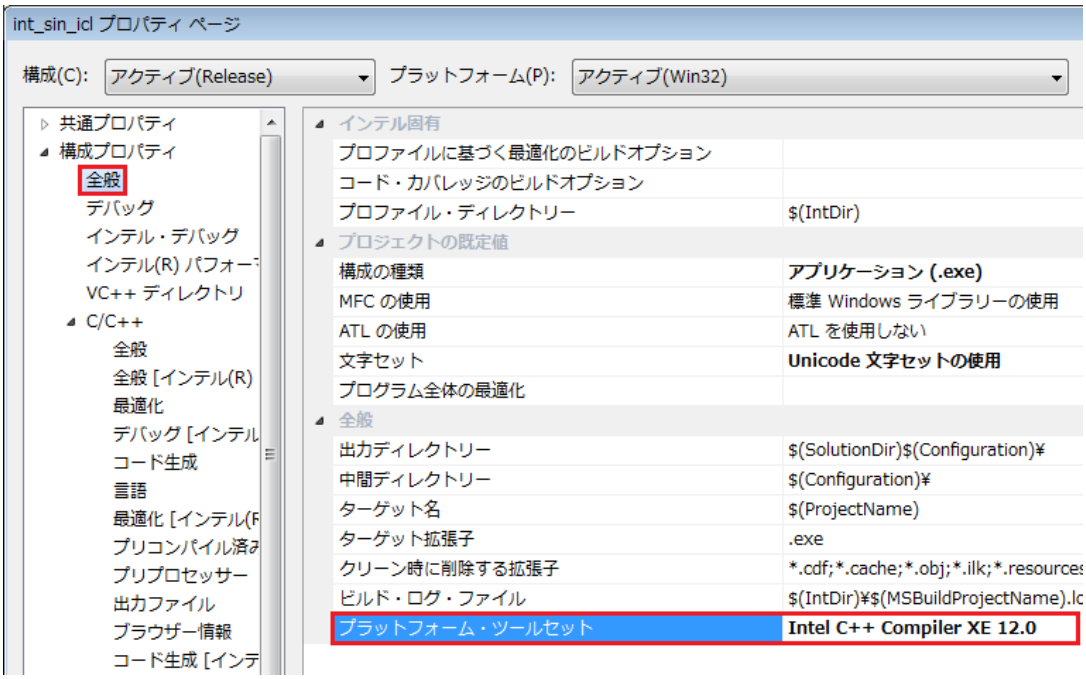
４－２－３．ビルド（最適化オプションあり）

次に、最適化オプションを使用してビルドを行います。次の手順を実行します。

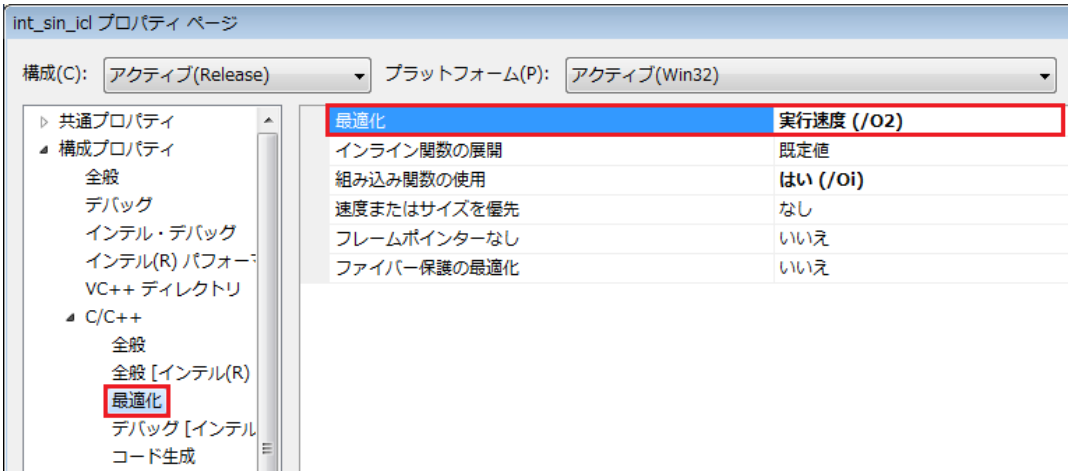
1. プロジェクトの構成を、”Release” 構成に変更してください。



2. プロジェクトの [プロパティ ページ] を開いて、使用するコンパイラーとしてインテル® C++ コンパイラーが指定されているか確認します。



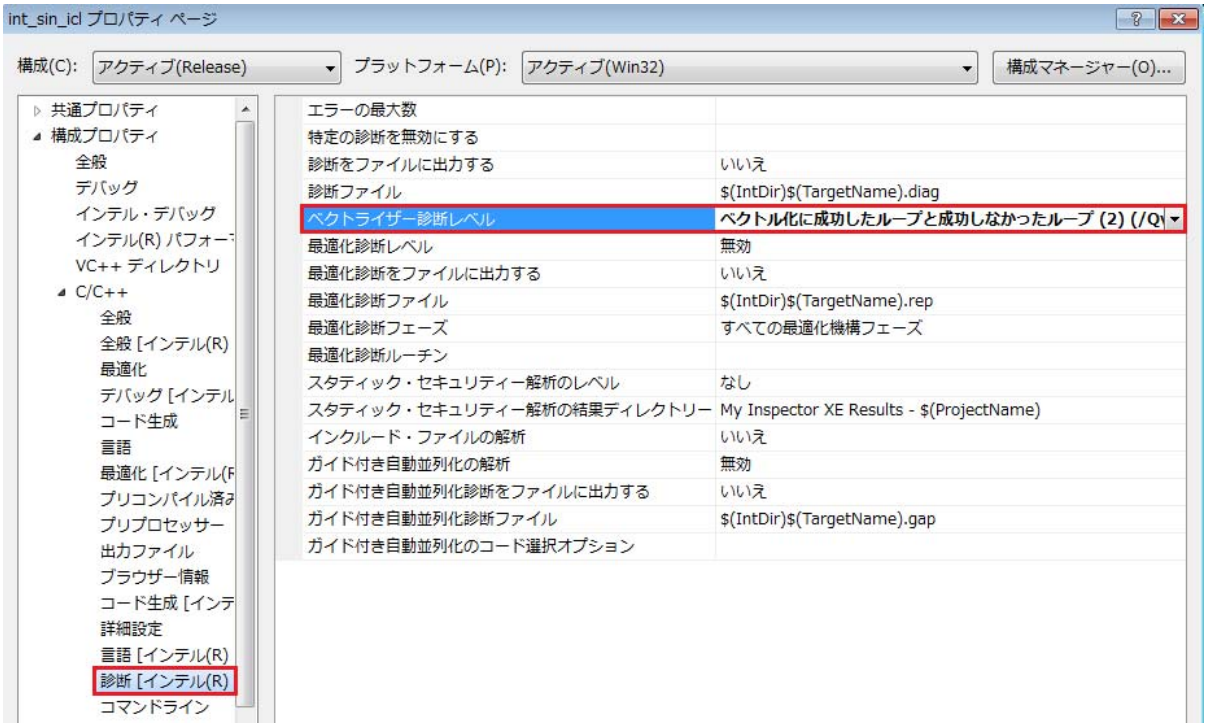
3. “Release” モードのプロジェクト構成では、インテル® C++ コンパイラーのデフォルトの最適化オプション（/O2 および /arch:SSE2）が有効となります。プロジェクトの [プロパティ ページ] で確認してみましょう。まず、[構成プロパティ] - [C/C++] - [最適化] を選択して、[最適化] が “実行速度 (/O2)” に設定されていることを確認します。



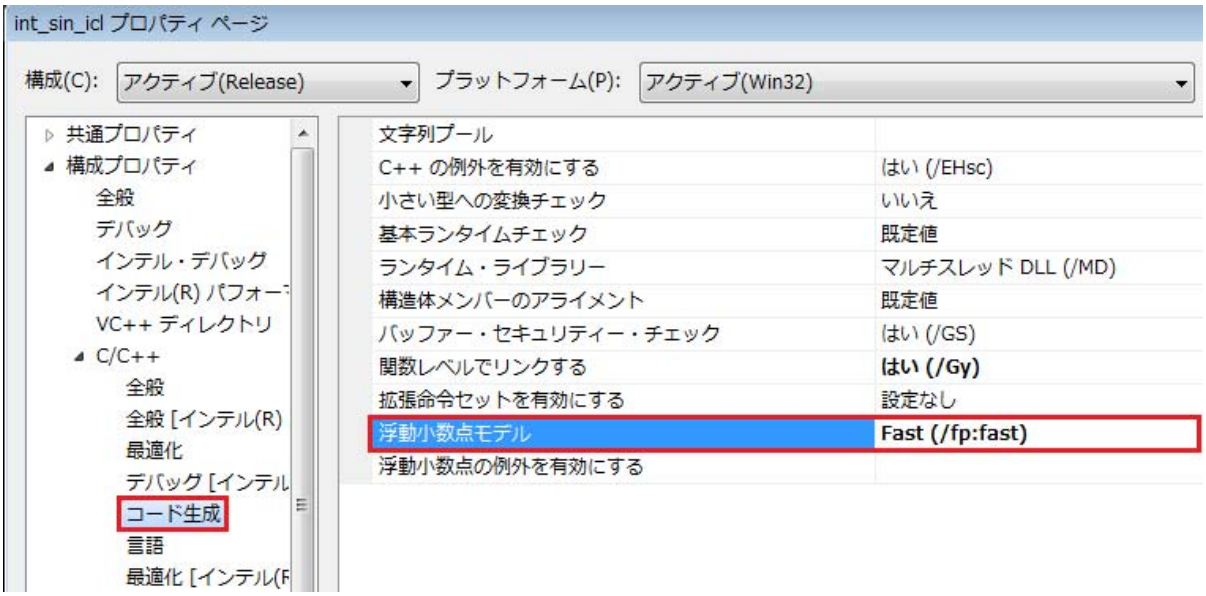
Note：前節の Debug 構成では、この最適化の値は “無効（/Od）” に設定されています。

また、/arch:SSE2 のオプションは [構成プロパティ] - [C/C++] - [コード生成] の [拡張命令セットを有効にする] の項目に存在します。この項目の設定値は“設定なし”と指定されていますが、/O2 が指定されている場合は、コンパイラーはデフォルトで /arch:SSE2 オプションを有効にします。このオプションを無効にする場合は、“拡張命令セットなし（/arch:IA32）”を選択します。

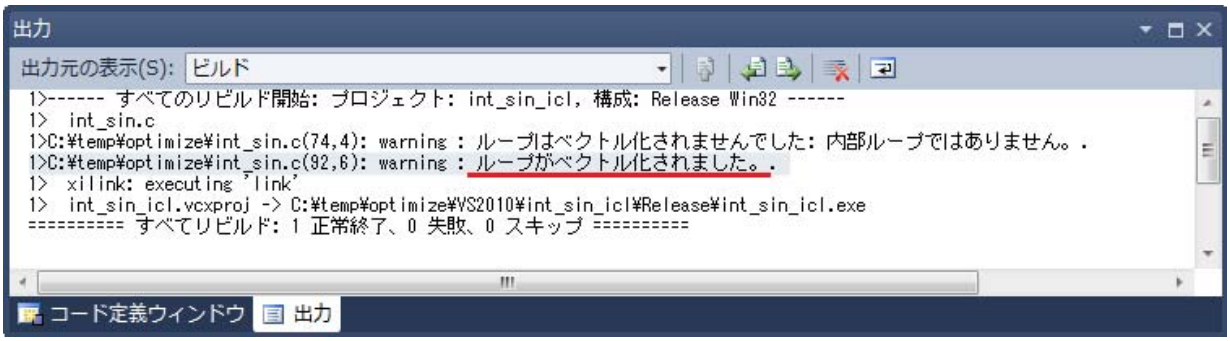
また、/arch:SSE2 オプションの効果を確認するために、「３－３．コンパイル（最適化オプションあり）」で説明した /Qvec-report オプションを指定します。[構成プロパティ] - [C/C++] - [診断[インテル(R) C++]] から [ベクトライザー診断レベル] の項目に、[ベクトル化に成功したループと成功しなかったループ（２）（/Qvec-report2）] を選択します。



それから最後に、[構成プロパティ] - [C/C++] - [コードの生成] から [浮動小数点モデル] の値をデフォルトの “Precise（/fp:precise）” から “Fast（/fp:fast）” に変更します。この変更をすることにより、インテル® C++ コンパイラーが本サンプルコードのループ処理に対してベクトル化を適用することができるようになります。

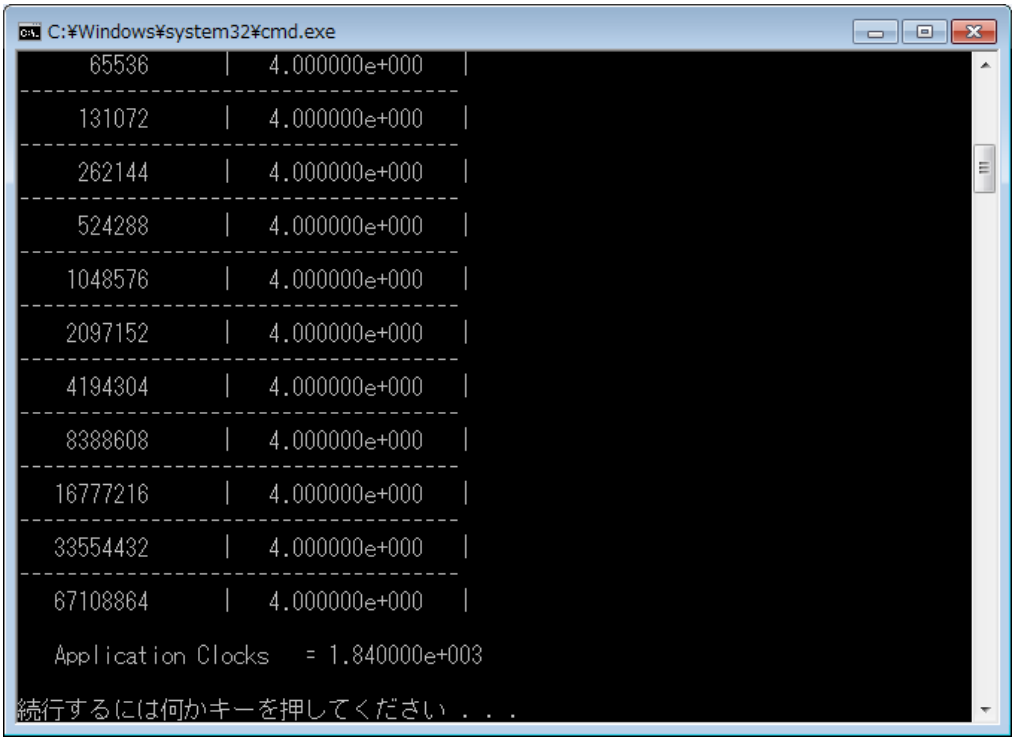


4. VS2010 のメニューから [ビルド] - [int_sin_icl のリビルド] を選択して、“Release” 構成で int_sin_icl プロジェクトをリビルドします。 表示されるレポート内容を確認してベクトル化の適用状況を確認します。



４－２－４．実行/パフォーマンスの比較

1. VS2010 のメニューから、[デバッグ] - [デバッグなしで開始] を選択します。コマンドウィンドウに最適化されたプログラムの実行結果が表示されます。

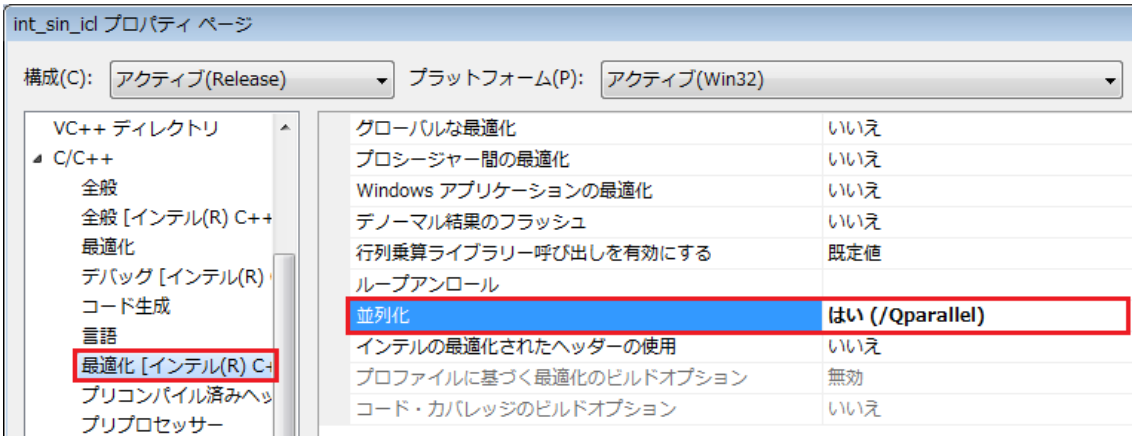


2. 最適化を行った場合の CPU 時間をメモして、最適化を行わなかった場合と比較します。ここでの結果では、約 3.5 倍の速度向上が確認できます。

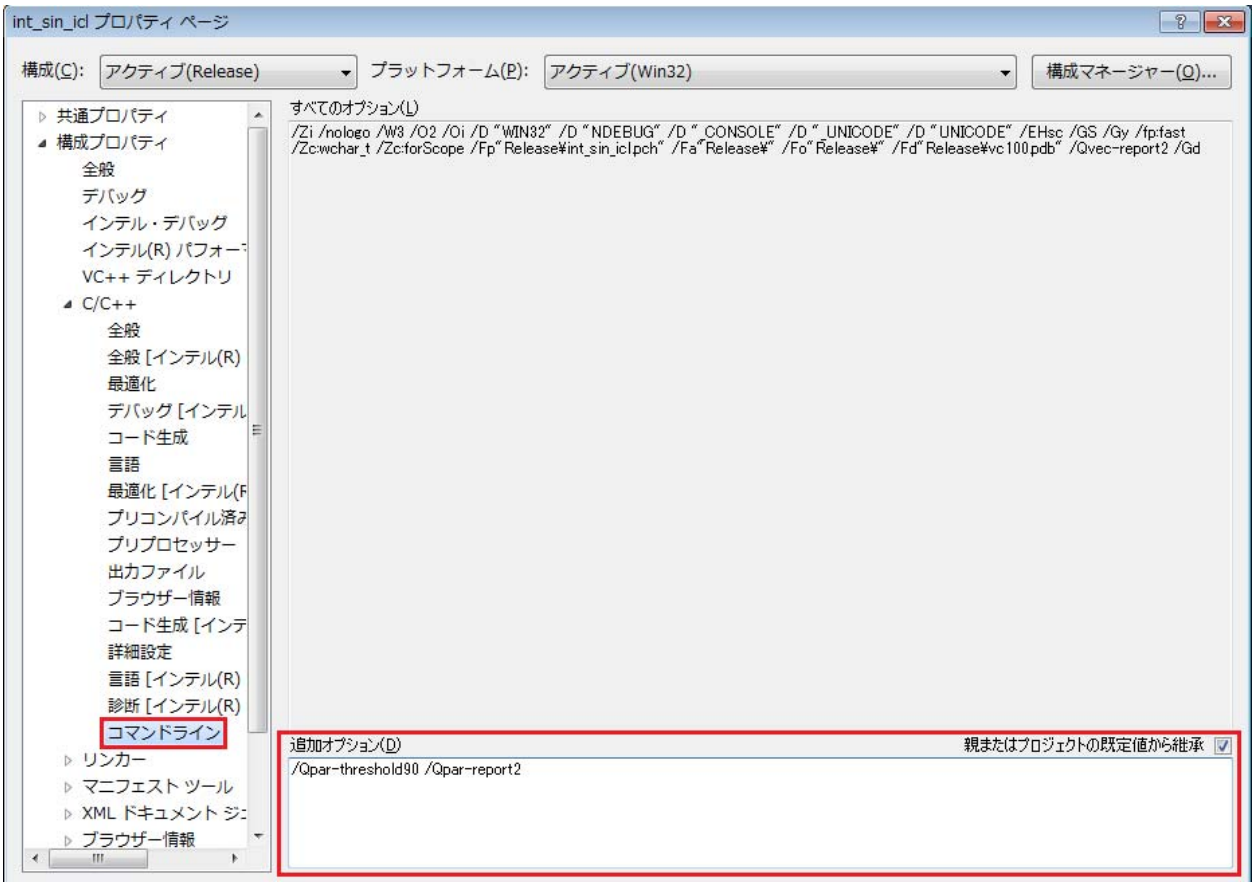
４－２－５．ビルド（並列化オプションあり）

コマンドライン同様、ここでも自動並列化オプション(/Qparallel)を使用してパフォーマンスを見ていきます。「３－５．コンパイル（並列化オプションあり）」の章で説明したとおり、/Qpar-threshold オプションが必要になります。また、結果レポートを表示させる /Qpar-report オプションも同様に指定します。

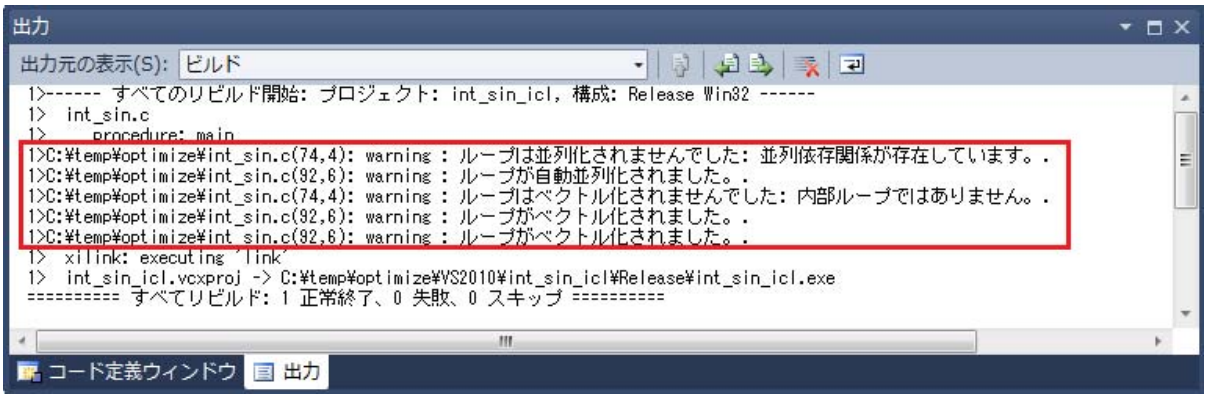
1. まず、プロジェクトの [プロパティ ページ] を開き、[構成プロパティ] - [C++] - [最適化[インテル(R) C++]] を選択して下図のように [並列化] の値を“はい（/Qparallel）”に設定します。



また、/Qpar-threshold と /Qpar-report オプションは、[プロパティ ページ] に指定する項目がないので、[構成プロパティ] - [C++] - [コマンドライン] の [追加のオプション] 欄に手書きで記入します。以下のように、/Qpar-threshold90 /Qpar-report2 と 2 つのオプションをスペースで区切って入力してください。

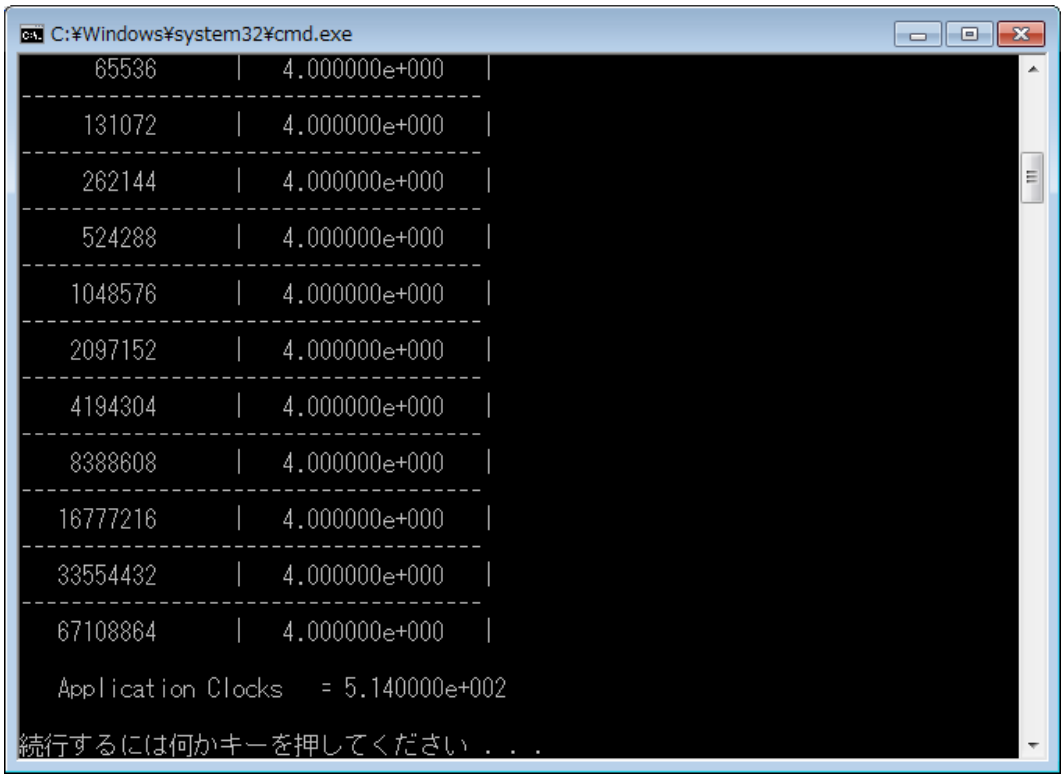


2. オプションの設定が完了したら、[ビルド] - [int_sin_icl のリビルド] を選択して、int_sin_icl プロジェクトをリビルドします。 結果レポートを確認してください。



４－２－６．実行/パフォーマンスの比較

1. VS2010 のメニューから、[デバッグ] - [デバッグなしで開始] を選択してプログラムを実行します。



2. 並列化を行った場合の CPU 時間をメモして、結果を比較します。並列化を行わなかった結果と比較すると、ここでは約 3.6 倍のパフォーマンスが得られています。

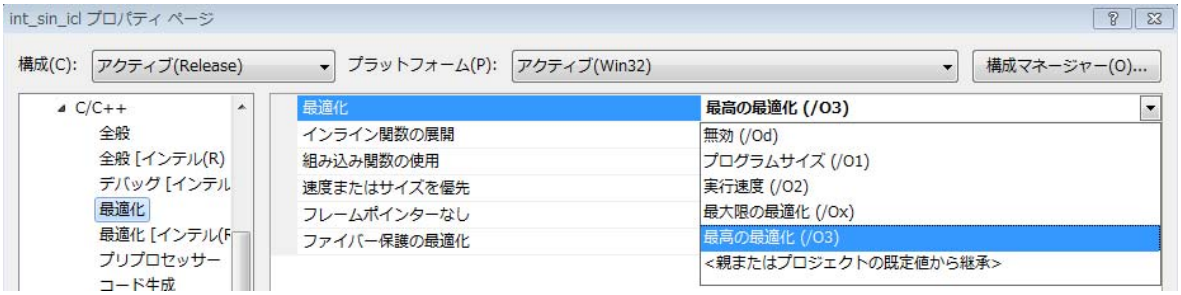
5．主な最適化オプション

前章では、サンプルプログラムを使用してインテル® C++ コンパイラーの基本的な使用方法といくつかの最適化オプションとともにパフォーマンスの検証を行いました。ここではさらにインテル® C++ コンパイラーの所有する主要な最適化オプションを紹介します。これらのオプションは、特にインテルプロセッサー上でより高いパフォーマンスが得られるように設計されています。また、これらのオプションを組み合わせることによりさらに最適化が実装される場合があります。今度はご自身のプログラムでパフォーマンスを検証してください。

5－1．高レベルな最適化（HLO）

前章では一般的な最適化オプションとして“/O2”を使用しました。インテル® C++ コンパイラーでは、その上の最適化オプションとして“/O3”があります。このオプションは、“/O2”の効果に加えてさらに強力なループ変換などを行います。後述する自動ベクトル化オプションはこのオプションと併用することで、より詳細なベクトル化分析ができるようになります。ただし、本オプションで逆にパフォーマンスが落ちるケースもありますので注意が必要です。このオプションは、多数の浮動小数点演算や大量のデータを処理するループが存在するアプリケーションに有効です。

コンパイル例：> icl /O3 /QxHost file.cpp
IDE からの使用：[構成プロパティ] － [C/C++] － [最適化] → [最適化]



5－2．プロシージャー間の最適化（IPO）

このオプション（/Qipo）はいわゆるプログラム全体の最適化を行う機能です。この機能では、コンパイラーはコンパイル時にすべてのソースコードを解析し、一度擬似オブジェクトを作成した上で最適化された本来のオブジェクトを生成します。Visual Studio 上でのビルドでは、この擬似オブジェクトを処理するために xilink というツールでリンク処理が実行されます。ソースファイル単位の最適化とは対照に、IPO 機能を使用した場合コンパイラーはプログラム全体の構成を把握でき、ソースファイル間における効果的な最適化を実施することができます。この IPO 機能を使用することにより、関数のインライン展開や定数伝播、エイリアス解析、

不要な処理の削除、スタックフレームのアライメントなど、その他多数のコード最適化が可能となります。このオプションも後述する自動ベクトル化オプションのヘルプとなります。なお、この IPO 機能を単一ソースファイル内に限定したい場合は、“/Qip”というオプションを使用します。

コンパイル例： > icl /Qipo /QxHost file1.cpp file2.cpp file3.cpp

IDE からの使用：

（VS2005/2008 の場合）

[構成プロパティ] ー [C/C++] ー [最適化] → [プロシージャー間の最適化]

または、

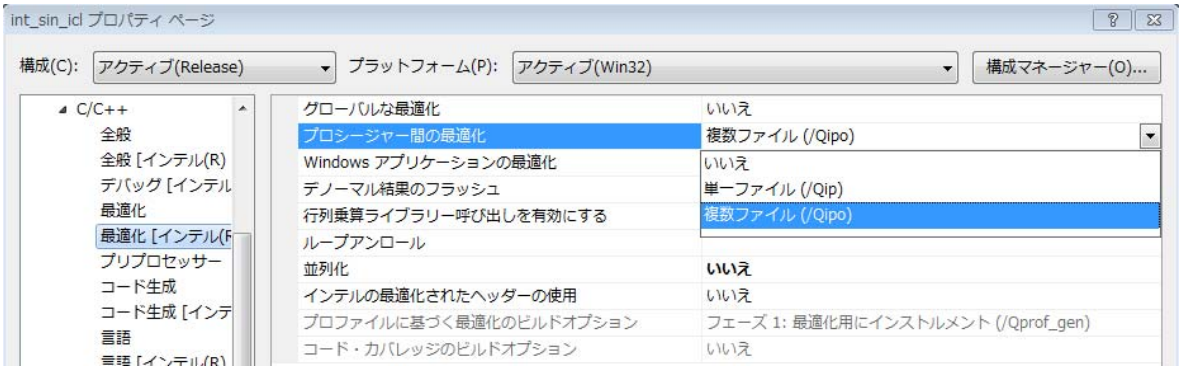
[構成プロパティ] ー [全般] → [プログラム全体の最適化]

（VS2010 の場合）

[構成プロパティ] ー [C/C++] ー [最適化[インテル(R) C++]] → [プロシージャー間の最適化]

または、

[構成プロパティ] ー [全般] → [プログラム全体の最適化]



5－3．プロファイルに基づく最適化（PGO）

この最適化機能は、アプリケーションの実行プロファイル情報に基づく最適化手法となります。実際にプログラムを実行させてプログラム動作の傾向を判断して最適化を行います。この PGO 機能を使用して最適化を行うには、以下の３つのフェーズを行う必要があります。

- フェーズ１：インストルメンテーション・コンパイル

  /Qprof-gen オプションを指定してプログラムをビルドし、プロファイル情報生成ロジックが埋め込まれた検証用アプリケーションを作成する

- フェーズ２：実行

  フェーズ１で生成した検証用アプリケーションを実行する。一回の実行で一つの動的プロファイル情報ファイル（.dyn）が生成される。この実行は何度行ってもよいが、なるべく同様な傾向の結果が得られるような実行を行う。

➢ フェーズ３：フィードバック・コンパイル

フェーズ２で生成した動的プロファイル情報ファイル（.dyn）を反映させるため、/Qprof-use オプションを使用してプログラムをビルドし、最適化されたアプリケーションを作成する。また、/Qipo オプションや自動ベクトル化オプションなどの最適化オプションも同時に指定することにより、さらに効果的なアプリケーションが作成できる。

この PGO 機能を使用することにより、コードレイアウトを見直して命令キャッシュ問題を軽減、コードサイズの縮小や分岐予測ミスの減少などの効果を得ることができます。

コンパイル例：

フェーズ１（インストルメント）> icl /Qprof-gen /Femyapp.exe file1.cpp file2.cpp file3.cpp

フェーズ２（アプリの実行） > myapp.exe

フェーズ３（フィードバック） > icl /Qprof-use /Qipo /QxHost /Femyapp.exe file1.cpp file2.cpp file3.cpp

IDE からの使用：

（VS2005/2008 の場合）

[構成プロパティ] ー [C/C++] ー [最適化] → [プロファイルに基づく最適化のビルドオプション]

または、

[構成プロパティ] ー [全般] → [プロファイルに基づく最適化のビルドオプション]

または、

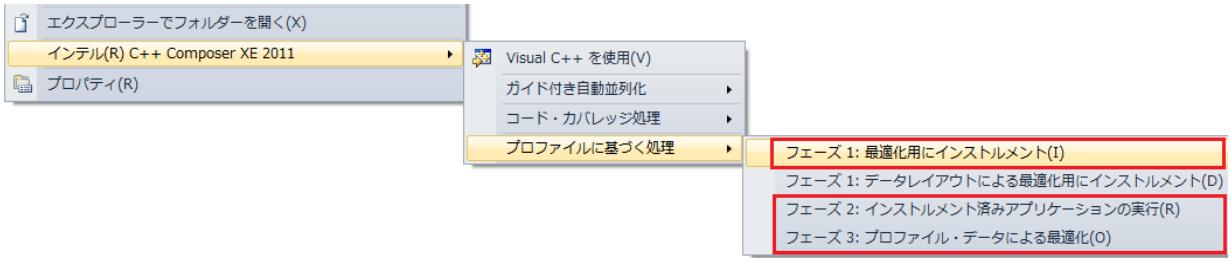
プロジェクトを右クリックして表示されるメニューから [インテル(R) C++ Composer XE 2011] ー [プロファイルに基づく処理]

（VS2010 の場合）

[構成プロパティ] ー [全般] → [プロファイルに基づく最適化のビルドオプション]

または、

プロジェクトを右クリックして表示されるメニューから [インテル(R) C++ Composer XE 2011] ー [プロファイルに基づく処理]



５－４．自動ベクトル化

ベクトル化とは、スカラ演算から SIMD 演算に変換して処理効率のよいコードを実装する技術です。インテル® C++ コンパイラーの自動ベクトル化機能を使用することにより、プログラム内のループ処理に対してベクトル化分析が行われ、ベクトル化が可能であると判断されたループに対して、SSE 命令を駆使した効果的な SIMD 演算コードを生成することができます。前章で、/arch:SSE2 というベクトル化オプションを使用しましたが、このオプションはインテルプロセッサーに特化したものではなくインテル互換プロセッサーでも動作するコードを生成します。ここでは、主にインテルプロセッサー用に特化した自動ベクトル化オプション（/Qx および /Qax）について説明します。

インテル® C++ コンパイラーの提供する自動ベクトル化のオプションは、以下のように SSE 命令のバージョンごとに分かれており、指定された SSE 命令のバージョン用に最適なコードを生成します。

/arch: 系： /arch:IA32、/arch:SSE、/arch:SSE2、/arch:SSE3、/arch:SSSE3、/arch:SSE4.1、/arch:AVX

/Qx 系： /QxSSE2、/QxSSE3、/QxSSSE3、/QxSSE3_ATOM、/QxSSE4.1、/QxSSE4.2、/QaxAVX、/QxHost

/Qax 系： /QaxSSE2、 /QaxSSE3、 /QaxSSSE3、 /QaxSSE4.1、 /QaxSSE4.2、 /QxAVX

/Qx 系のオプションは、指定した SSE バージョンを所有するプロセッサーに特化したコードを生成します。つまり、たとえば /QxSSE4.2 を指定して作成された実行バイナリーは、SSE4.2 命令を搭載しないプロセッサー（例：Intel Core 2 Duo など）では動作しません。SSE4.2 以上の命令セットを搭載するプロセッサー上で実行させる必要があります。また /QxHost オプションは、コンパイラーが実行される開発システム（ホストマシン）のプロセッサーが持つ最新の SSE 命令用のベクトル化オプションを自動で選択してくれる便利なオプションです。たとえば、SSSE3 を搭載するインテル® Core™2 Duo プロセッサーなどのシステムでコンパイルした場合は、/QxHost オプションは /QxSSSE3 オプションに置き換えられます。一般的に /QxHost オプションは、開発システムが実行環境となる場合に使用されるオプションです。

一方、/Qax 系のオプションは、指定された SSE バージョン用のコードと汎用 SSE コードの複数のコードを生成しようとします。コンパイラーは、指定された SSE バージョンのコードの生成が有益と見なした場合のみ、指定された SSE バージョン用のコードを生成し、そして汎用 SSE コードも生成します。そうでない場合は、汎用 SSE コードのみの生成となります。この汎用 SSE コードは、デフォルトでは /arch:SSE2 レベルの SSE コードが生成されます。たとえば /QaxSSE4.2 と指定した場合、/QxSSE4.2 レベルのコードの生成が有益であるとみなされた場合は /QxSSE4.2 コードと、汎用コードとして /arch:SSE2 レベルのコードの２つが生成されます。そしてプログラムの実行において、使用プロセッサーの情報を取得して実行すべきパスを自動で切り替えます（自動ディスパッチ）。また、作成する汎用 SSE コードは変更することもできます。デフォルトでは /arch:SSE2 でしたが、たとえば /QxSSSE3 と明示的に指定した場合は、作成される汎用コードは /QxSSSE3 レベルとなります。なお、この /Qax 系オプションを使用した場合は、バイナリーサイズが /Qx 系オプションよりも大きくなる可能性があります。

では以下に、自動ベクトル化オプションのコンパイル例を示します。

コンパイル例：

> icl /QxSSE4.2 file.cpp … SSE4.2 に特化したコードを生成。SSE4.2 以上の CPU で動作可能。

> icl /QaxSSE4.2 file.cpp … SSE4.2 コードの生成が有益と見なされた場合は、SSE4.2 に特化したコードと SSE2(/arch:SSE2) の汎用コードを生成。そうでない場合は、汎用コードのみ。

> icl /QaxSSE4.2 /QxSSSE3 file.cpp … SSE4.2 コードの生成が有益と見なされた場合は、SSE4.2 に特化したコードと SSSE3 の汎用コードを生成。そうでない場合は、汎用コードのみ。

> icl /QaxAVX /arch:IA32 file.cpp … AVX コードの生成が有益と見なされた場合は、AVX に特化したコードと x86/x87 命令の汎用コードを生成。そうでない場合は、汎用コードのみ。

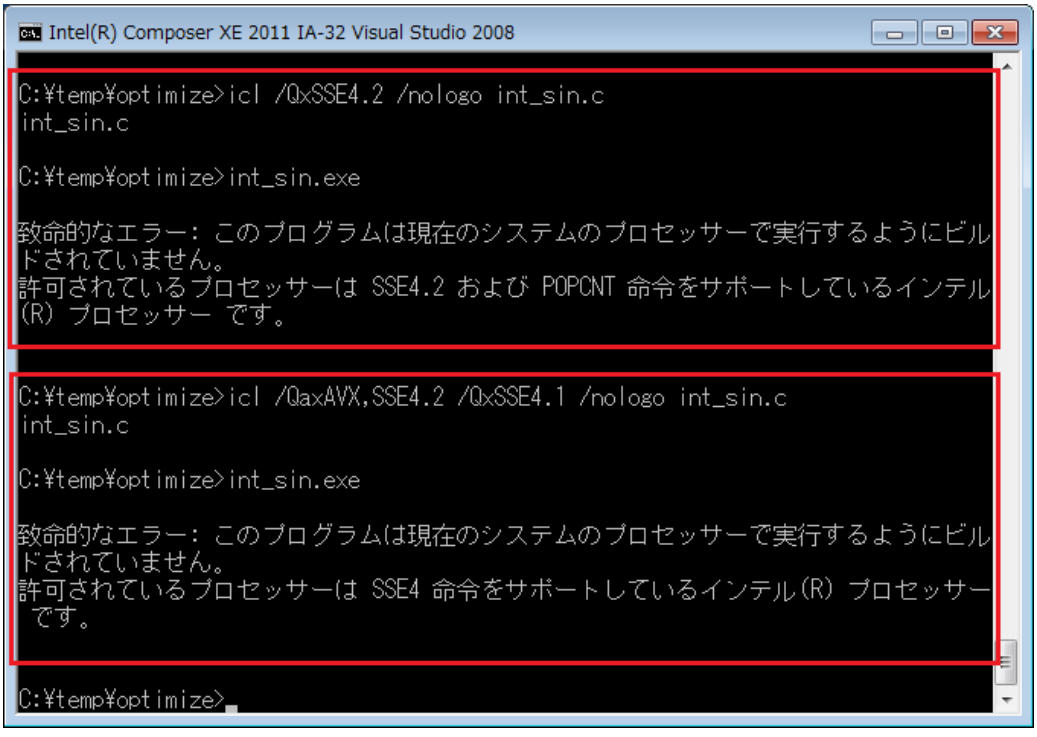
> icl /QaxAVX,SSE4.2 /QxSSSE3 file.cpp … AVX と SSE4.2 のコードの生成が有益と見なされた場合は、AVX と SSE4.2 に特化したコードと SSSE3 の汎用コードを生成。そうでない場合は汎用コードのみ。

> icl /QaxSSE4.2,SSE4.1 /QxAVX file.cpp … /QxAVX によって上書き。AVX に特化したコードを生成。

> icl /QxSSE4.2 /QxSSE4.1 file.cpp … /QxSSE4.1 によって上書き。SSE4.1 に特化したコードを生成

Note：自動ベクトル化機能を有効にするには“/O2”レベル以上の最適化オプションが必要です。

なお、SSE 命令バージョンによる実行エラーの例も以下に示します。本ドキュメントで使用しているシステムのプロセッサーは、インテル® Core™ 2 Quad で、最高で SSSE3 命令までを搭載しています。



IDE からの使用：

（VS2005/2008 の場合）

/Qx 系オプション

[構成プロパティ] ー [C/C++] ー [コード生成] → [指定された命令セットの専用コード生成]

/Qax 系オプション

[構成プロパティ] ー [C/C++] ー [コード生成] → [指定された命令セットの専用および汎用コード生成]

（VS2010 の場合）

/Qx 系オプション

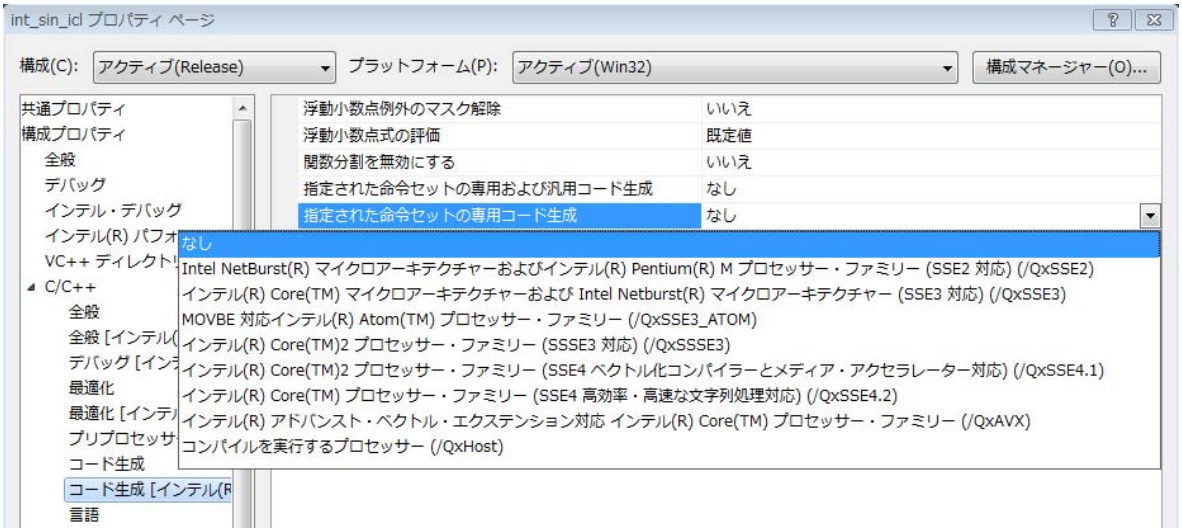
[構成プロパティ] ー [C/C++] ー [コード生成 [インテル(R) C++]] →

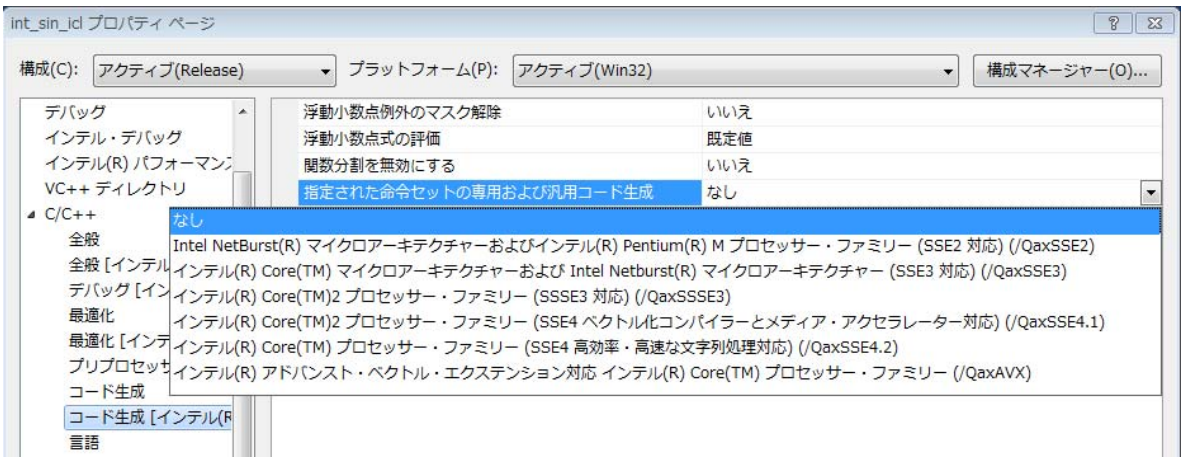
[指定された命令セットの専用コード生成]

/Qax 系オプション

[構成プロパティ] ー [C/C++] ー [コード生成 [インテル(R) C++]] →

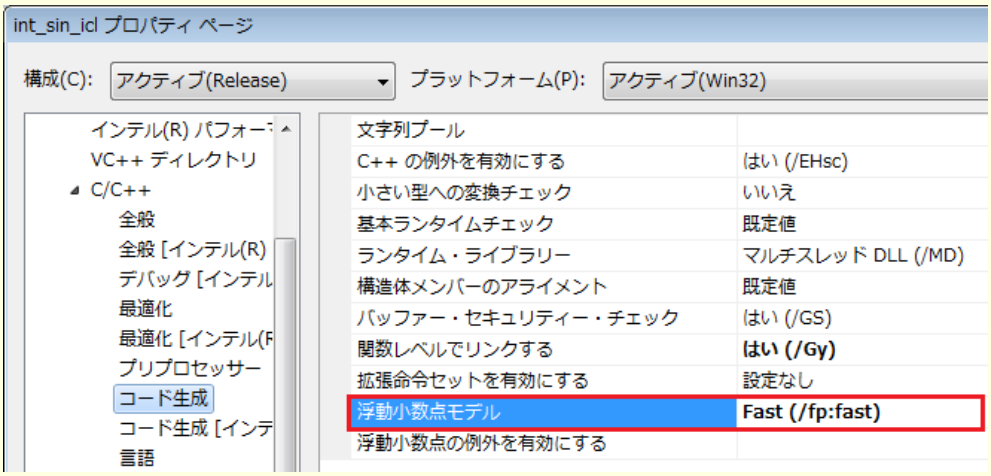
[指定された命令セットの専用および汎用コード生成]





Note：自動ベクトル化はすべてのループ処理に対して適用されるわけではありません。ベクトライザーによる分析でベクトル化が可能であると判断されたループのみが対象となります。対象外と見なされるループ処理の例として、ループの反復においてデータや処理に依存関係がある場合や、ループ処理内でデータアクセスが連続でなかったり、関数をコールしていたり、複数のポインター間で共通のメモリー領域を参照（ポインターエイリアス）していたり、またループの途中でループから抜けたり、十分な処理サイズがない場合などがあげられます。しかしこれらの問題は、他の最適化オプションと併用することで解決される場合があります。たとえば、IPO 機能を使用することにより、ループ回数、アライメント、データ依存などのループに関する情報が明確になり、また関数がインラインされることでベクトライザーがベクトル化し易い状態を作り出します。それから /O3 オプションもループ変換等を行うことでベクトライザーのヘルプとなる場合があります。また、/Oa や /Qalias-args- オプションを使用して、ポインターエイリアスが存在しないことを断言することができます。しかし、これらのオプションはプログラマーの責任で指定することになります。その他の解決方法として、コード内に #pragma（#pragma ivdep、#pragma loop count、#pragma vector always など）やキーワード（restrict など）を追記して特定の内容をベクトライザーに直接指示することもできます。

また、VS2010 では、浮動小数点モデルがデフォルトで /fp:precise に設定されており、ベクトル化するためには以下のように /fp:fast に変更する必要があります。



自動ベクトル化の結果レポートは、前の章で説明したとおり、/Qvec-report オプションが使用できます。

５－５．自動並列化

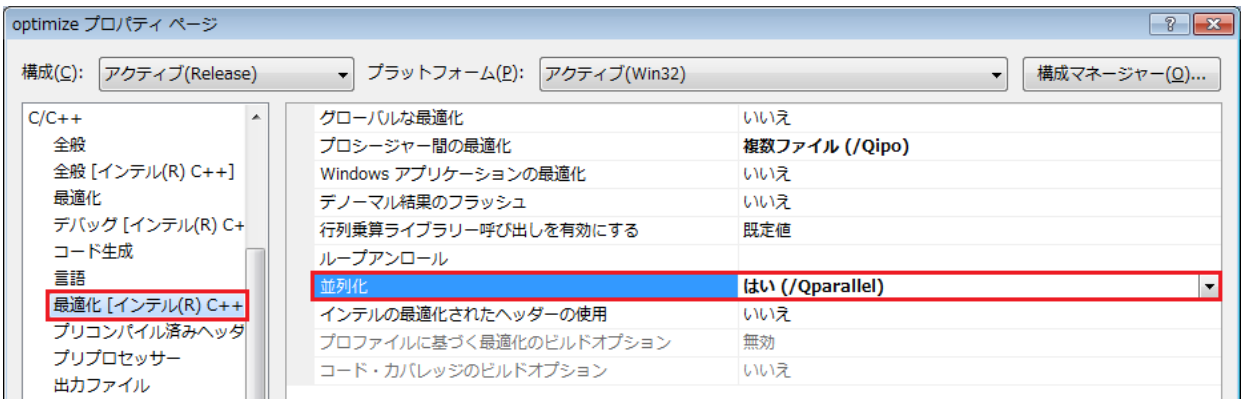
自動並列化の機能は前章で既に何度か紹介していますが、本機能はコンパイラーによって並列化が可能とみなされたループに対してマルチスレッドによる並列処理を実装するものです。コンパイラーはコンパイル時にソースコード上のループ処理に対して分析を行いループの実行回数や処理の大きさ、複雑性などをチェックし、安全に並列化が実装できるループ、また並列化によって高い効果が見込まれるループに対してのみ、この機能を適用します。通常、並列化が可能であるとみなされるループの条件として以下のようなものがあります。

- ループの反復回数がコンパイル時に認識できること（while ループは対象外）
- ループ依存問題（イテレーション間で同一メモリーアドレスへのアクセス）がないこと
- ループへのジャンプやループからのジャンプがないこと。
- 複雑な処理を行う関数をコールしていないこと
- 十分な処理サイズがあること

なお、本機能は OpenMP として実装されます。そのため OpenMP で使用するランタイム関数や環境変数などを使用することができます。また本機能を使用した場合は、並列化適用の有無にかかわらず OpenMP のランタイムライブラリー（libiomp5md.dll）が必要になりますのでご注意ください。
それでは以下に、本機能のオプションと利用方法について記します。なお、この自動並列化オプションと共に IPO および自動ベクトル化オプションも同時に使用することをお勧めします。

/Qparallel　　… 自動並列化オプション
/Qpar-report[n]　… 結果レポート表示。 n の値は 0 から 3（n 省略時のデフォルト値は 1）
/Qpar-threshold[n]　… 並列化実装の効果に関する閾値。n の値は 0 から 100（※本オプションについては「３－５．コンパイル（並列化オプションあり）」を参照してください）

コンパイル例：　> icl /Qipo /QxHost /Qparallel /Qpar-threshold90 /Qpar-report2 file1.cpp file2.cpp file3.cpp
IDE からの使用：
（VS2005/2008 の場合） [構成プロパティ] ー [C/C++] ー [最適化] → [並列化]
（VS2010 の場合）　　[構成プロパティ] ー [C/C++] ー [最適化[インテル(R) C++]] → [並列化]



５－６．ガイド付き自動並列化（GAP）

インテル® C++ コンパイラーの自動ベクトル化や自動並列化機能には、結果レポートを表示させるオプション（/Qvec-report[n] および /Qpar-report[n]）があります。このオプションに高い表示レベルを指定して、ベクトル化や並列化が適用されなかった場合の理由を表示させることが可能ですが、実際「どのようにすればベクトル化や並列化が適用されるのか」という具体的な解決策を提供してくれるわけではありません。ここで紹介するガイド付き自動並列化（GAP）機能は、ソースコード修正のアドバイスや推奨オプションの提案、また Pragma などのキーワードを提供してプログラムの並列化（自動ベクトル化/自動並列化）に役立つ詳細な情報を表示する診断ツールです。

> Note：インテル® C++ コンパイラーでは、SIMD 演算によるベクトル化とマルチスレッドによる並列化の両方を“並列処理”、“自動並列化”として位置づけています。

この GAP 機能を使用するには、次の /Qguide オプションを使用します。

/Qguide:[n] … n は診断レベルで 1 から 4 の数字を指定。指定しない場合は 4 がデフォルト。

GAP 機能にはこの /Qguide の他にもいくつか関連オプションがありますが、IDE からの操作でこれらの関連オプションは自動設定されるので、ここではこのメインオプションのみの紹介とします。

なお、GAP 機能の使用に際し、以下の条件があります。

- 自動ベクトル化同様、/O2 レベル以上の最適化オプションが必要です。
- /Qparallel オプションを指定しない場合、GAP はベクトル化に関するアドバスのみ表示します。
- GAP 機能を使用したビルドでは、オブジェクトファイルや実行形式ファイルは生成されません。

また、GAP が表示するすべての診断メッセージの詳細やソースコード修正例などは、メッセージ ID 単位で、インテル® C++ コンパイラーのドキュメントに記載されています。

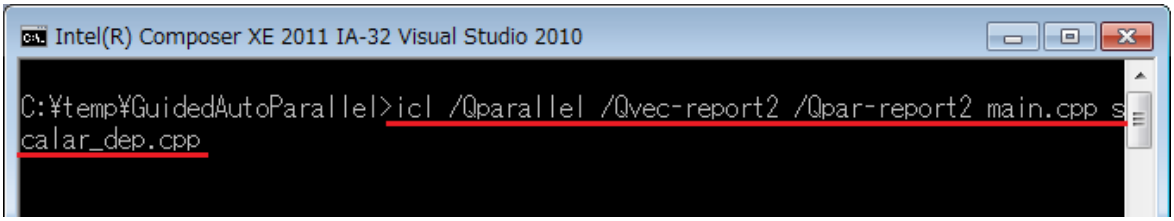
それでは、GAP 機能の使用方法を説明します。説明にあたって本製品に付属する以下のサンプルプログラムを使用し、本機能の効果も検証します。サンプルプログラムを適当なフォルダーに解凍してください。

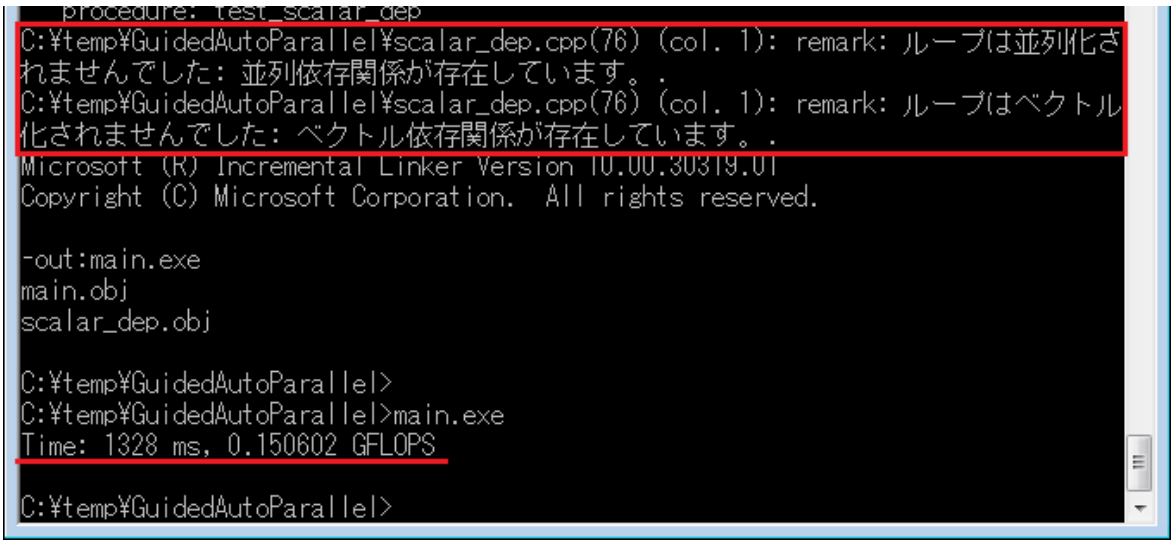
C:¥Program Files¥Intel¥ComposerXE-2011¥Samples¥en_US¥C++¥GuidedAutoParallel.zip

＜コマンドラインからの使用＞

まず、GAP 機能を使用する前に、本サンプルコードの自動ベクトル化と自動並列化の適用状況を確認します。以下のようにコンパイルします。

> icl /Qparallel /Qvec-report2 /Qpar-report2 main.cpp scalar_dep.cpp

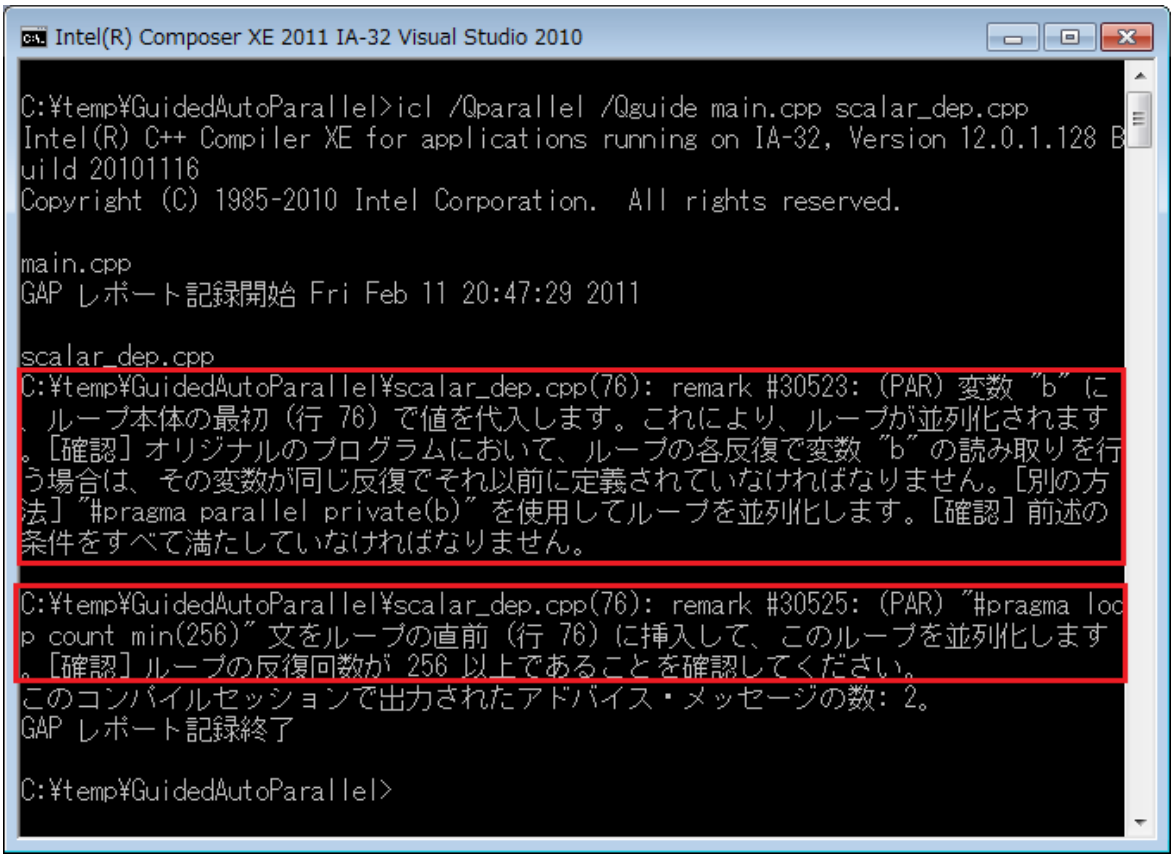




コンパイル結果より、scalar_dep.cpp ファイルの 76 行目のループ（本サンプルコードのメイン処理部）が、依存関係が存在するため自動ベクトル化および自動並列化が適用されていないことが確認できます。作成された実行ファイル（main.exe）を実行して処理時間を確認します。

それでは、GAP 機能を使用してみます。以下のようにコンパイルします。

> icl /Qparallel /Qguide main.cpp scalar_dep.cpp



GAP から、scalar_dep.cpp ファイルの 76 行目のループに対して 2 つの診断レポートが表示されています。まず、変数“b”に関する問題で、ループ本体の最初で値を代入するか、“#pragma parallel private(b)”キーワー

ドの使用が提案されています。これらの提案で変数“b”に関するループ反復間の依存問題は解決しますが、GAP メッセージの“[確認]”にあるようにこの修正が妥当かどうかをきちんと確認する必要があります。また 2 つ目のメッセージでは“#pragma loop count min(256)”キーワードを使用して、コンパイラーに対してループの最低反復回数情報を提供することにより自動並列化が実装できるようアドバイスを提供しているのが分かります。このループの反復回数は 10000 回ありますのでこのキーワードを安全に使用できます。
それでは、これら 2 つのキーワードを以下のようにソースコード（scalar_dep.cpp）に追加してみます。

```
for (i=0; i<n; i++) {
      if (A[i] > 0) {b=A[i]; A[i] = 1 / A[i]; }
      if (A[i] > 1) {A[i] += b;}
  }
```

修正

```
#pragma parallel private(b)
#pragma loop count min(256)
for (i=0; i<n; i++) {
      if (A[i] > 0) {b=A[i]; A[i] = 1 / A[i]; }
      if (A[i] > 1) {A[i] += b;}
  }
```

ソースコードの修正が完了したら、再度以下のコマンドでコンパイルし実行して結果を確認します。

> icl /Qparallel /Qvec-report2 /Qpar-report2 main.cpp scalar_dep.cpp

```
C:¥temp¥GuidedAutoParallel¥scalar_dep.cpp(77) (col. 1): remark: ループが自動並列
化されました。.
C:¥temp¥GuidedAutoParallel¥scalar_dep.cpp(77) (col. 1): remark: ループがベクトル
化されました。.
C:¥temp¥GuidedAutoParallel¥scalar_dep.cpp(77) (col. 1): remark: ループがベクトル
化されました。.
Microsoft (R) Incremental Linker Version 10.00.30319.01
Copyright (C) Microsoft Corporation.  All rights reserved.

-out:main.exe
main.obj
scalar_dep.obj

C:¥temp¥GuidedAutoParallel>main.exe
Time: 186 ms, 1.07527 GFLOPS

C:¥temp¥GuidedAutoParallel>
```

GAP の診断レポートのアドバイスをソースコードに反映することにより、今度は自動ベクトル化および自動並列化が適用されていることが分かります。また、結果は前回の約 7 倍の速度向上が確認できています。

Note : GAP を IPO 機能と共に使用した場合、診断メッセージ内容が異なる場合があります。実際、本サンプルコードで以下のように /Qipo オプションを追加した場合は、“#pragma loop count min(256)”に関するメッセージが表示されなくなります。

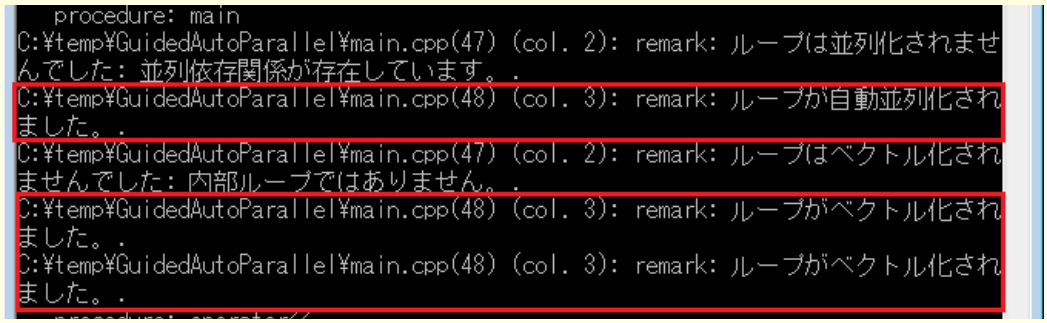
> icl /Qipo /Qparallel /Qguide main.cpp scalar_dep.cpp

これは、IPO 機能によりコンパイラーはファイル間の情報を取得できるので、このループの反復回数が 10000 回であることを認識できるようになるからです。つまりこの場合、ソース

コードに“#pragma parallel private(b)”のみを追加し、以下のように /Qipo オプションと共にコンパイルすることで自動ベクトル化および自動並列化が適用されるようになります。

> icl /Qipo /Qparallel /Qvec-report2 /Qpar-report2 main.cpp scalar_dep.cpp

以下、コンパイル結果（最適化レポート）です。



上記最適化レポートでは、main.cpp ファイルに対して結果が表示されています。これは、IPO 機能によって scalar_dep.cpp ファイル内の test_scalar_dep() 関数がインライン展開されているからです。そのため最適化レポートでは、main.cpp ファイル内の test_scalar_dep() 関数をコールしている 48 行目に対して最適化の結果が表示されています。このように IPO 機能を使用してコンパイラーに広い視野を持たせることにより最適化しやすい状況を作り出すことができるようになります。

＜IDE からの使用＞

IDE からこの GAP 機能を使用する場合は、以下の複数の指定方法が可能です。

- プロジェクトの [プロパティ ページ] から
- Visual Studio の [ツール] メニューから
- コンテキストメニューから

《プロジェクトの [プロパティ ページ] から使用する場合》

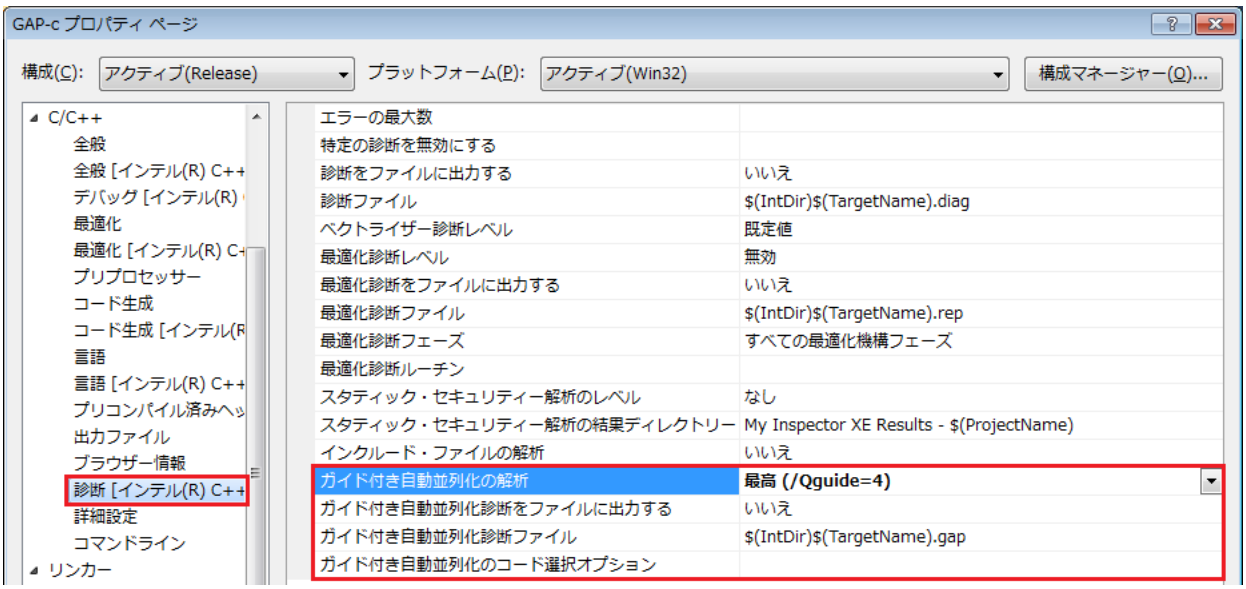
（VS2005/2008 の場合）

[構成プロパティ] ー [C/C++] ー [診断] → [インテル固有・ガイド付き自動並列化]

（VS2010 の場合）

[構成プロパティ] ー [C/C++] ー [診断 [インテル(R) C++]] → （ガイド付き自動並列化の項目）

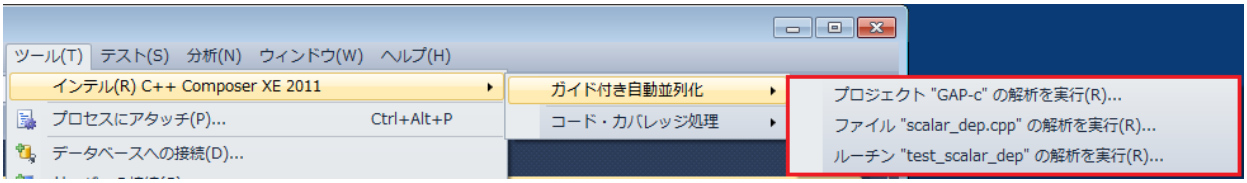
Note：/Qguide の GAP メインオプションに加えて、診断結果の出力先の指定など GAP 関連オプションも設定することができます。



《Visual Studio の [ツール] メニューまたはコンテキストメニューから使用する場合》

これらの方法で GAP を使用すると、GAP が提供するその他の機能を利用し易くなります。

まず、[ツール] メニューから使用する場合は、[ツール] - [インテル(R) C++ Composer XE 2011] - [ガイド付き自動並列化] から表示される項目を選択します。

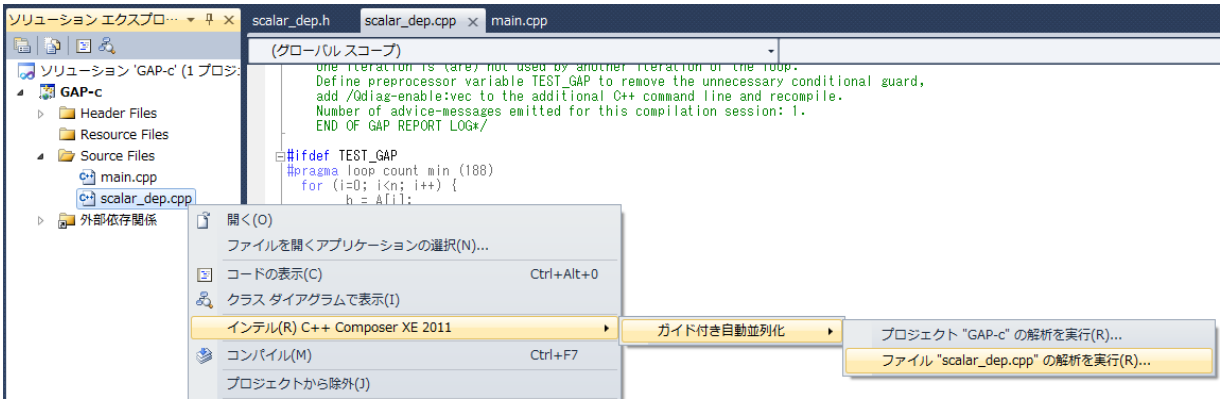


この項目には GAP による診断対象の範囲が表示されます。この表示される範囲内容はマウスでフォーカスされている箇所によって異なります。たとえば「ソリューション エクスプローラー」内のあるプロジェクトがフォーカスされていればそのプロジェクト全体が診断範囲となり、ソースコードにフォーカスされていればそのソースコードが診断範囲となります。またソースコード上のある関数（内）にフォーカスされていればその関数が、関数内のある領域がハイライトされていればその行の処理が GAP 診断の対象となります。

コンテキストメニューから GAP を使用する場合は、この診断ターゲットの指定がもっと直感的になります。この方法では、ターゲットをマウスで右クリックして表示されるメニューから [インテル(R) C++ Composer XE 2011] - [ガイド付き自動並列化] を選択します。ここに表示される内容は選択したターゲットによって異なります。ではそのいくつかの例を以下に示します。

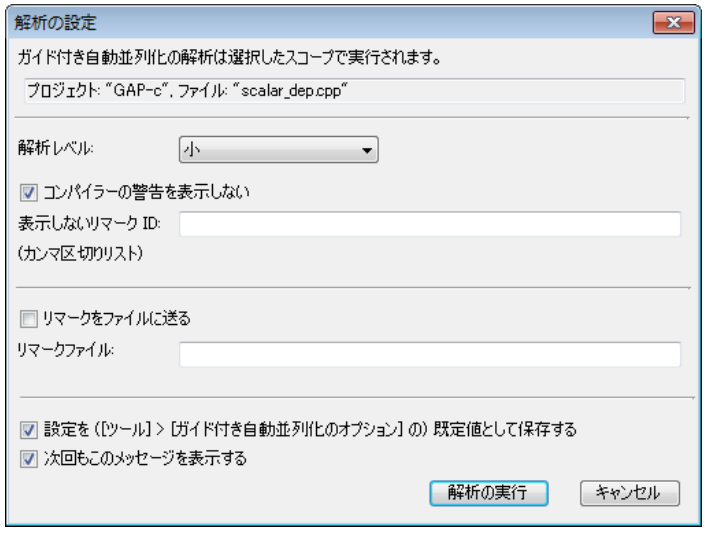
> **ご注意：** この方法で GAP を使用する場合は、プロジェクトの [プロパティ ページ] の /Qguide オプションの設定値が“無効”になっていることを確認してください。また、その他の GAP 関連オプションも確認することをお勧めします。

（ソースコードを選択した場合）



（ソースコード内のある領域を選択した場合）

また、[ツール] メニューまたはコンテキストメニューから GAP 機能を選択すると右図の [解析の設定] ダイアログが表示されます。このダイアログでは /Qguide オプションのレベルに相当する“解析レベル”の指定や GAP の診断結果（リマーク）をファイルに送る設定などが可能となります。なお、このダイアログの内容は VS の [ツール] メニューから [オプション] を選択し [インテル(R) C++] - [ガイド付き自動並列化] で設定されている内容が規定値となります。



ご注意：この [オプション] から [インテル(R) C++] - [ガイド付き自動並列化] の内容は、GAP 機能をプロジェクトの [プロパティ ページ] から使用する場合は無視されます。

5－7．スタティック・セキュリティー解析（SSA）

このSSA機能は、ソフトウェアの脆弱性を除去しセキュリティー攻撃などに強いアプリケーションの作成、また実行中の予期しない不具合を回避することを目的としています。本機能を使用するにはインテル® Parallel Studio XE のライセンスが必要となり、インテル® C++ コンパイラーで分析を行い、分析結果はインテル® Inspector XEを使用して閲覧する手順となっています。

本機能には 250 種類以上のレポート内容が存在し、以下に示すような言語やプログラムロジック、メモリーやスレッドに関するエラーの分析が行われます。

- バッファーオーバーフローやメモリー違反
- 未初期化変数またはオブジェクトの使用
- メモリーリーク
- ポインターや動的メモリー領域の不正使用
- 未確認入力データの使用
- 演算オーバーフローとゼロ除算
- 不必要なコードや重複コード
- ストリング、メモリー、フォーマットライブラリーの不正使用
- 一貫性のないオブジェクト宣言
- OpenMP 指示語やインテル Cilk Plus 言語機能の誤使用
- C++ および Fortran 言語の使用問題

また、以下に SSA 機能を使用する際の条件や注意事項、関連情報などを記載します。

- SSA の分析を行うシステムには、インテル® C++ コンパイラーが必要となる。
- SSA の分析結果を表示するシステムには、インテル® C++ コンパイラーおよびインテル® Inspector XE が必要となる。
- SSA 機能を使用するためには、コンパイル時にエラーがあってはならない。
- SSA 機能を使用したビルドでは、実行形式ファイルは作成されない。
- SSA 機能を使用したビルドでは、それぞれのソースファイルの擬似オブジェクトが生成され、分析結果はリンク時に生成される（この動作は IPO 機能と類似する）。
- SSA 機能では、コンパイラーに mcpcom.exe ではなく、svcpcom.exe が使用される。

上記内容にある通り SSA では擬似オブジェクトを生成するため、既存の正規オブジェクトを上書きしたくない場合は、SSA 用に新しいビルド構成を作成することをお勧めします。新しいビルド構成は [ビルド] - [構成マネージャー] を選択して [構成マネージャー] ダイアログを表示し、「アクティブ ソリューション構成」から “<新規作成...>” を選択して作成することができます。

SSA 機能を使用するためには以下のコンパイルオプションを使用します。

/Qdiag-enable:sc{[1|2|3]} … 数字は分析レベルで、内容は以下の通り。（デフォルトの分析レベルは 2）

1：致命的エラー　2：すべてのエラー　3：すべてのエラーおよび警告

また、ヘッダーファイルも分析対象とする場合は、以下のオプションを追加します。

/Qdiag-enable:sc-include

この他にも SSA にはいくつかの関連オプションが存在しますが、ここではこれらの SSA メインオプションのみの紹介とします。

コンパイル例：

> icl /Qdiag-enable:sc3 /Qdiag-enable:sc-include file1.cpp file2.cpp file3.cpp

IDE からの使用：

（VS2005/2008 の場合）

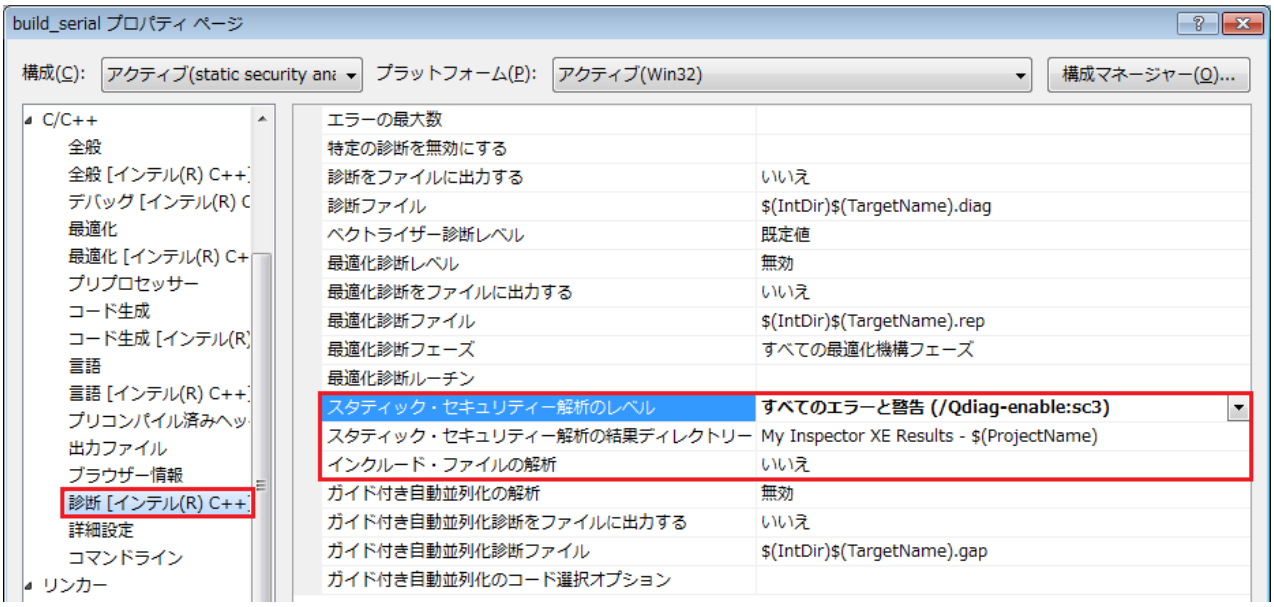
[構成プロパティ] ー [C/C++] ー [診断] → [スタティック・セキュリティー解析のレベル]

[構成プロパティ] ー [C/C++] ー [診断] → [インクルード・ファイルの解析]

（VS2010 の場合）

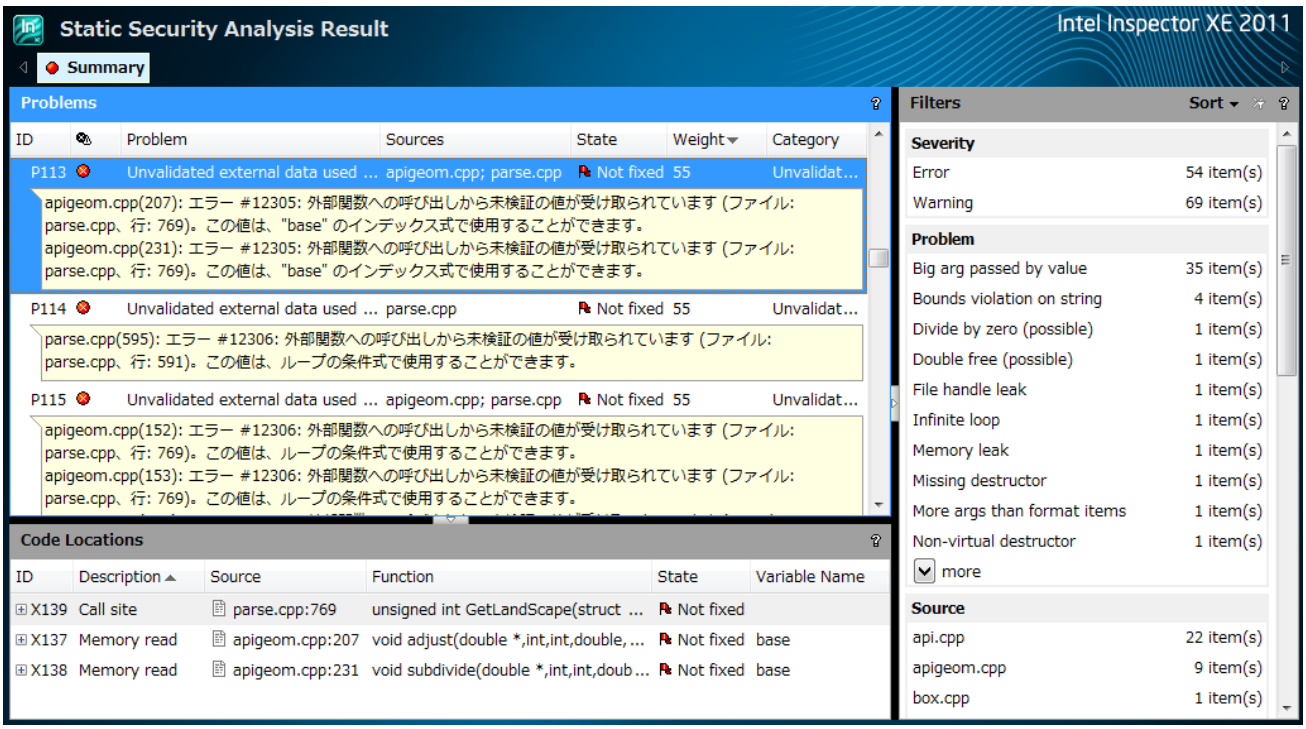
[構成プロパティ] ー [C/C++] ー [診断 [インテル(R) C++]] → [スタティック・セキュリティー解析のレベル]

[構成プロパティ] ー [C/C++] ー [診断 [インテル(R) C++]] → [インクルード・ファイルの解析]

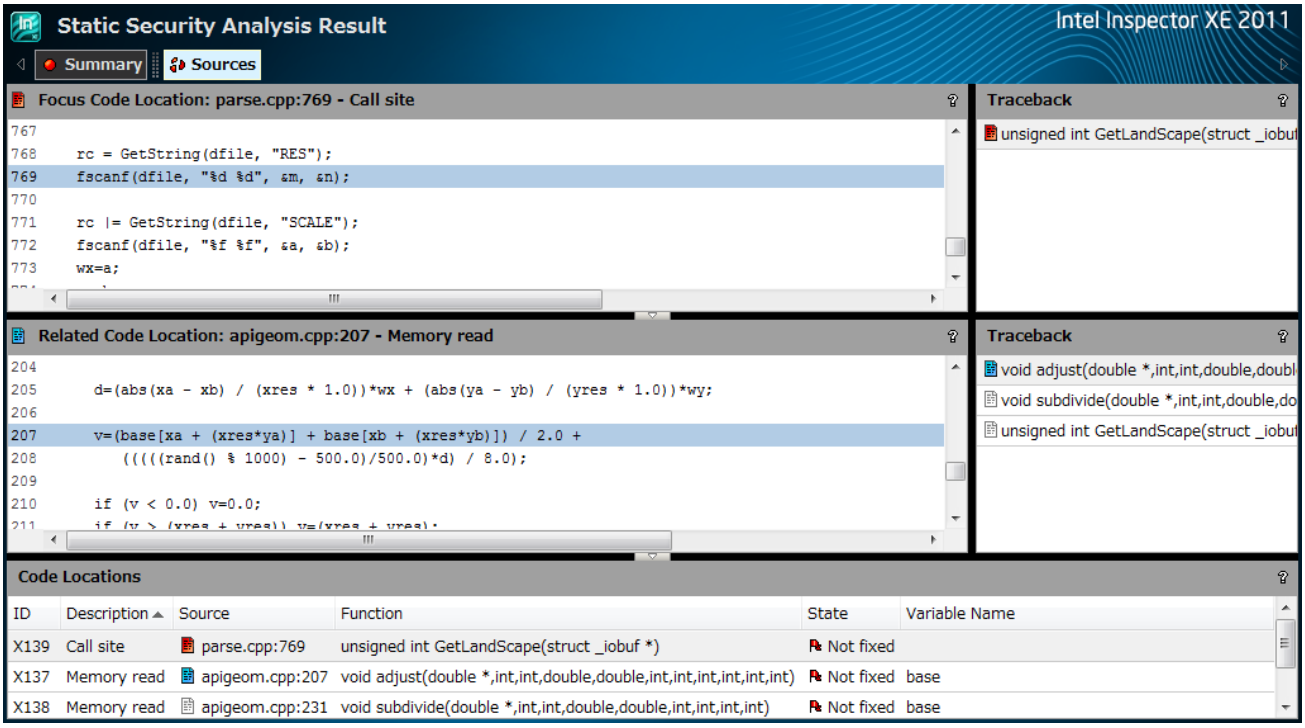


SSA の分析が完了すると、プロジェクトのルートディレクトリに My Inspector XE Results-<product name> のディレクトリがデフォルトで作成され、このディレクトリに分析結果データが格納されます。IDE から本機能を使用している場合は、インテル® Inspector XE の表示ビューが自動で起動するか、またはこのディレクトリに生成される *.inspxe ファイルをダブルクリックして表示ビューを起動することができます。

以下の画面は、SSA 機能による結果の例です。検出された問題の種類や内容、ソースコード内の行番号や関数情報などが表示されています。また右のペインでは、サマリー表示やフィルターの機能が利用できます。



また、問題の項目をダブルクリックしてソースコードを表示し問題の場所を容易に特定することができます。



５－８．関数／ループ・プロファイラー

アプリケーション内の時間のかかる処理（Hotspot）を確認することを目的に、本機能を使用して関数およびループ処理の実行回数や処理時間などの情報を取得することができます。
本機能の使用手順はプロファイルに基づく最適化（PGO）に似ており、まず実装コンパイルを行い、アプリケーションを実行させることでプロファイルデータを出力させます。なお、本機能は、Time Stamp Counter (TSC) を利用しているためシングルスレッドプログラムに対してのみ利用することができます。

＜使用手順＞

① 実装コンパイル

- 以下のプロファイルレベルのいずれかを指定してアプリケーションをコンパイルします。
  - 関数レベルのプロファイル

    オプション：/Qprofile-functions
  - 関数およびループレベルのプロファイル

    オプション：/Qprofile-loops:<arg>（arg = inner | outer | all（プロファイルするループの指定））
  - ループ本体レベルのプロファイル

    オプション： /Qprofile-loops-report:[n]（n = 1 または 2（実装制御レベル））

    （※ 本レベルを使用するには、/Qprofile-loops: オプションが必須となります）

コンパイル例：

```
> icl /Qprofile-loops:all /Qprofile-loops-report:2 /Feoutfile.exe main.cpp sub1.cpp sub2.cpp
```

② 実行

- コンパイルしたアプリケーションを実行し、出力ファイルを生成します。出力ファイルはカレントフォルダーに出力されます。

（出力ファイル）

・loop_prof_funcs_<timestamp>.dump（タブ区切りテキスト形式）←関数情報ファイル

・loop_prof_loops_<timestamp>.dump（タブ区切りテキスト形式）←ループ情報ファイル

・loop_prof_<timestamp>.xml（XML 形式）←関数およびループ情報ファイル

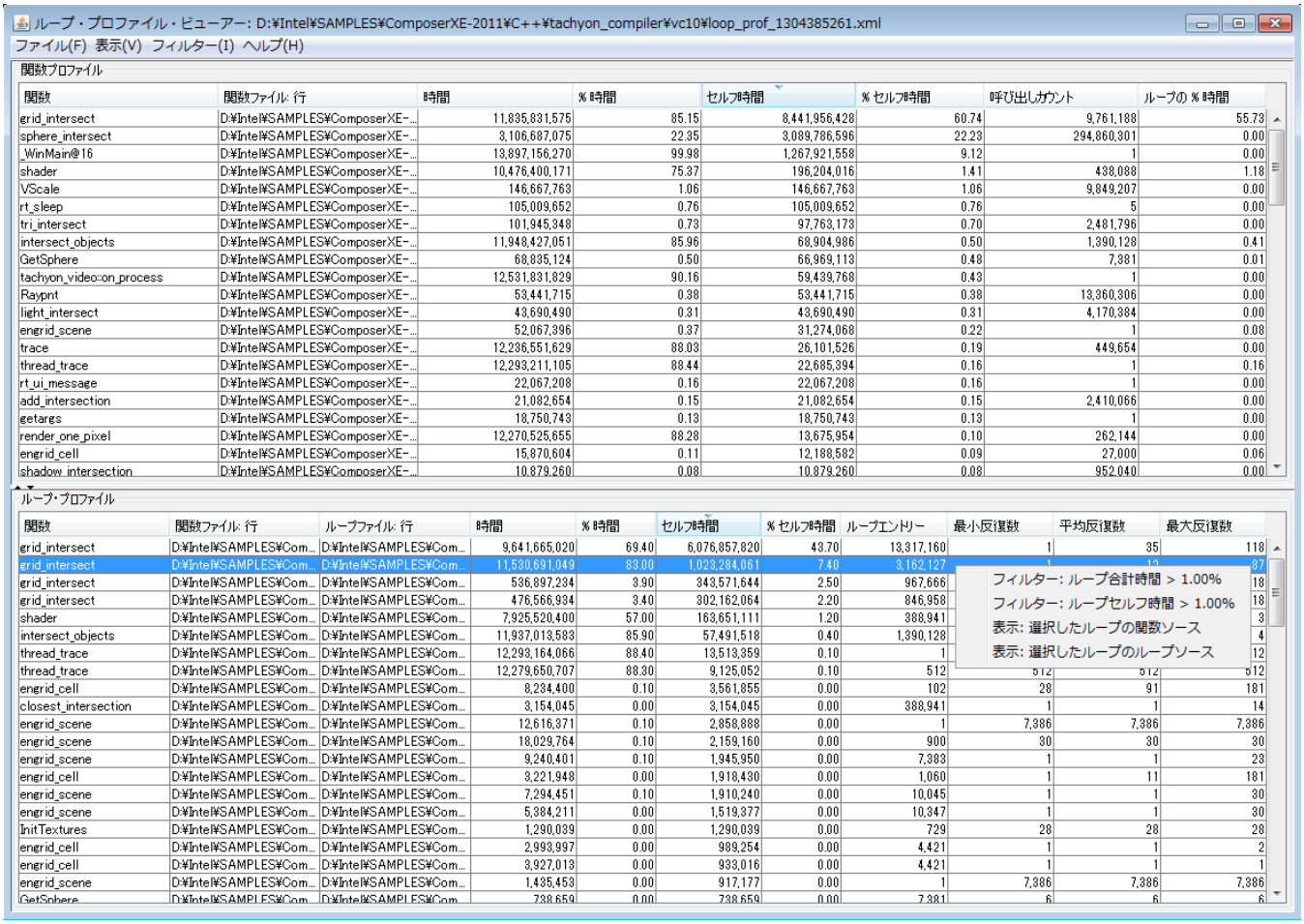
③ 結果表示

- XML 形式ファイルの表示は、以下のデータ表示ツールを実行（ダブルクリック）して表示されるウィンドウのファイルメニューから XML 形式ファイルを指定して開くことができます。

  データ表示ツール：C:¥Program Files¥Intel¥ComposerXE-2011¥bin¥loopprofileviewer.jar

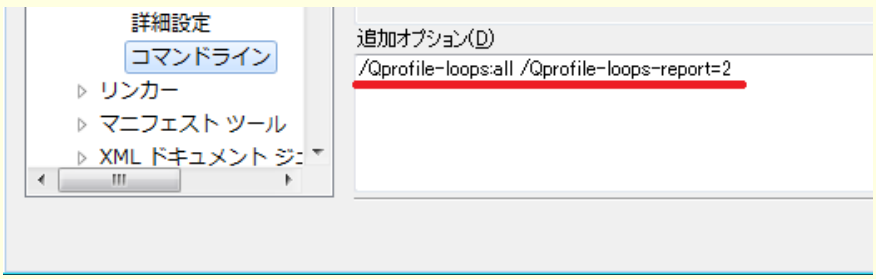
  ※ このデータ表示ツールを実行するには、システムに Java ランタイム環境が必要です。

本機能の表示結果例を以下に示します。



表示結果は、「関数プロファイル」と「ループ・プロファイル」とで上下に分かれて表示されます。プロファイル内容には実行回数や実行時間などが表示されており、アプリケーション内で処理時間のかかっている関数やループの情報を確認することができます。また、ある結果レコードを選択して右クリックするとポップアップメニューが表示され、表示内容のフィルター機能や選択したレコードの関数／ループのソースコードを表示させることもできます。表示内容の各項目に関する詳細は、製品マニュアルを参照してください。

Note : 本機能を Visual Studio から使用する場合、プロファイルレベルのオプションがプロジェクトのプロパティー ページに用意されていないので、[C/C++] - [コマンドライン] の [追加のオプション] に直接指定してください。

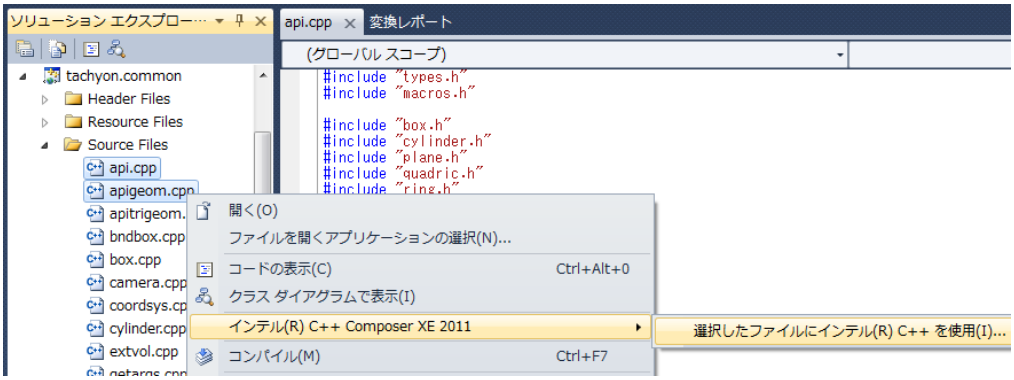


6． 関連情報

ここでは、インテル® C++ コンパイラーでのビルドに関して、その他関連情報をいくつかご紹介します。

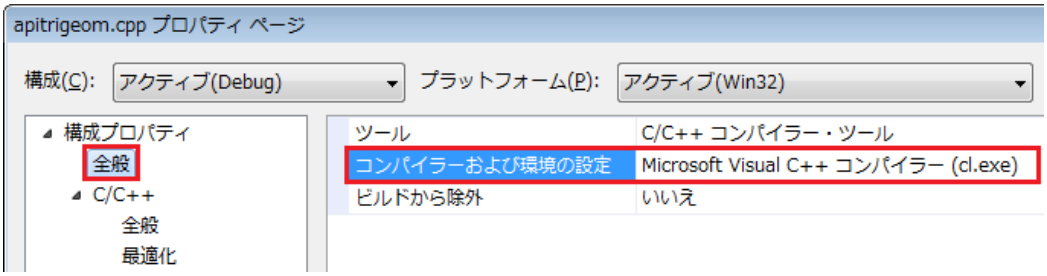
6－1． ソースファイル単位でのインテル® C++ コンパイラーの使用方法

特定のソースファイルに対してインテル® C++ コンパイラーを指定する場合は、以下の図のように対象のファイルを選択（Ctrl キー＋左クリックで複数選択可能）して、右クリックで表示されるメニューから [インテル(R) C++ Composer XE 2011] - [選択したファイルにインテル(R) C++ を使用] を選択します。

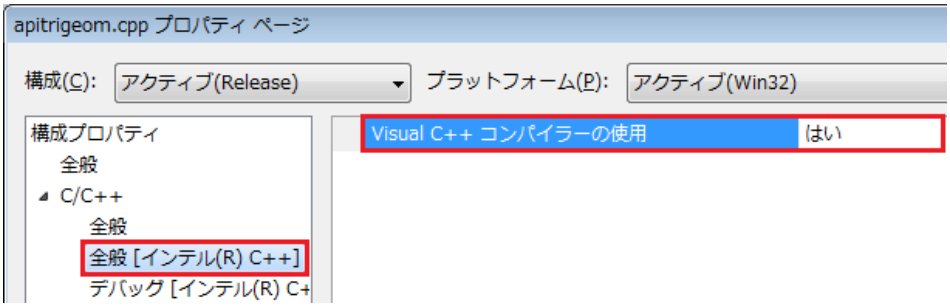


ただし、既にインテル® C++ コンパイラーを使用するように設定されている場合はこのメニューは表示されません。ソースファイルを右クリックして [プロパティ] を選択し、表示される [プロパティ ページ] ダイアログから現在選択されているコンパイラーが確認できます。

（Visual Studio 2005/2008）



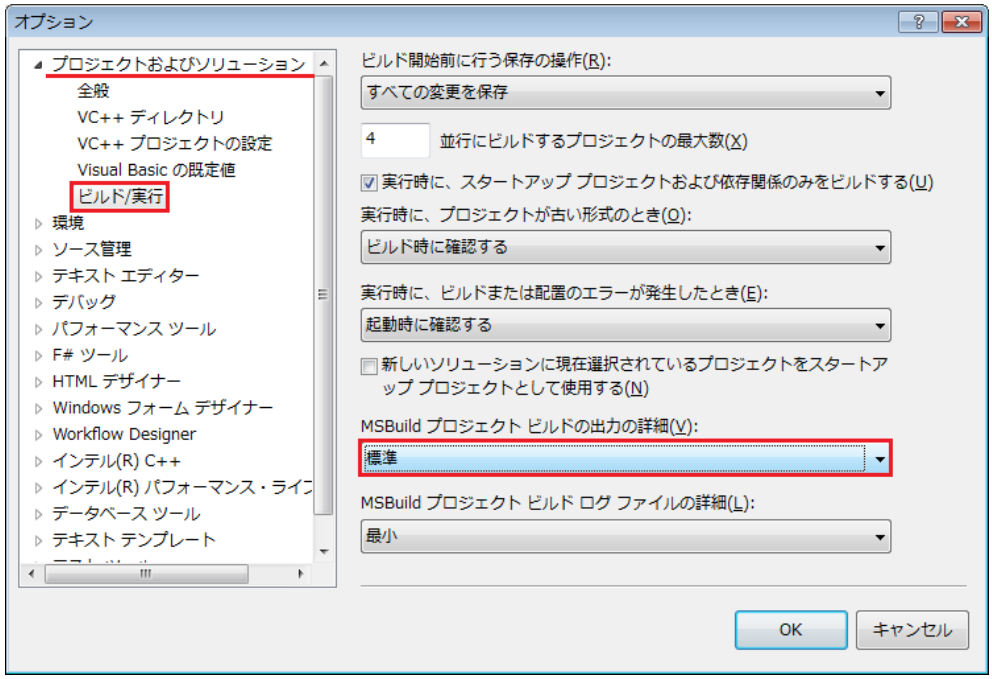
（Visual Studio 2010）

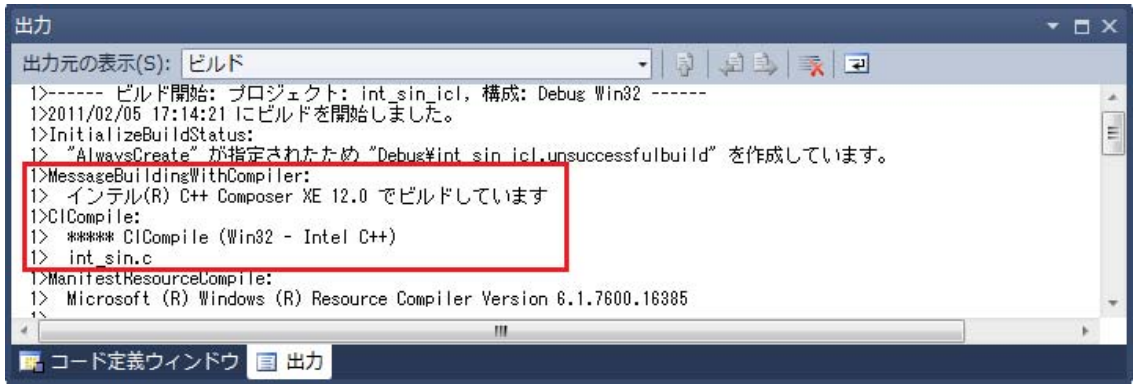


Note：この設定を行うには、プロジェクト自体がインテル® C++ コンパイラーによって管理されている必要があります。Microsoft Visual C++ のプロジェクトに対してこの操作を行うと、プロジェクト自体がインテル® C++ コンパイラーを使用するように切り替えられ、選択したソースファイルのみインテル® C++ コンパイラー使用の対象となり、それ以外のファイルは Microsoft Visual C++ コンパイラーのままという状態となります。

6－2．VS2010 出力ウィンドウでインテル® C++ コンパイラーの確認方法

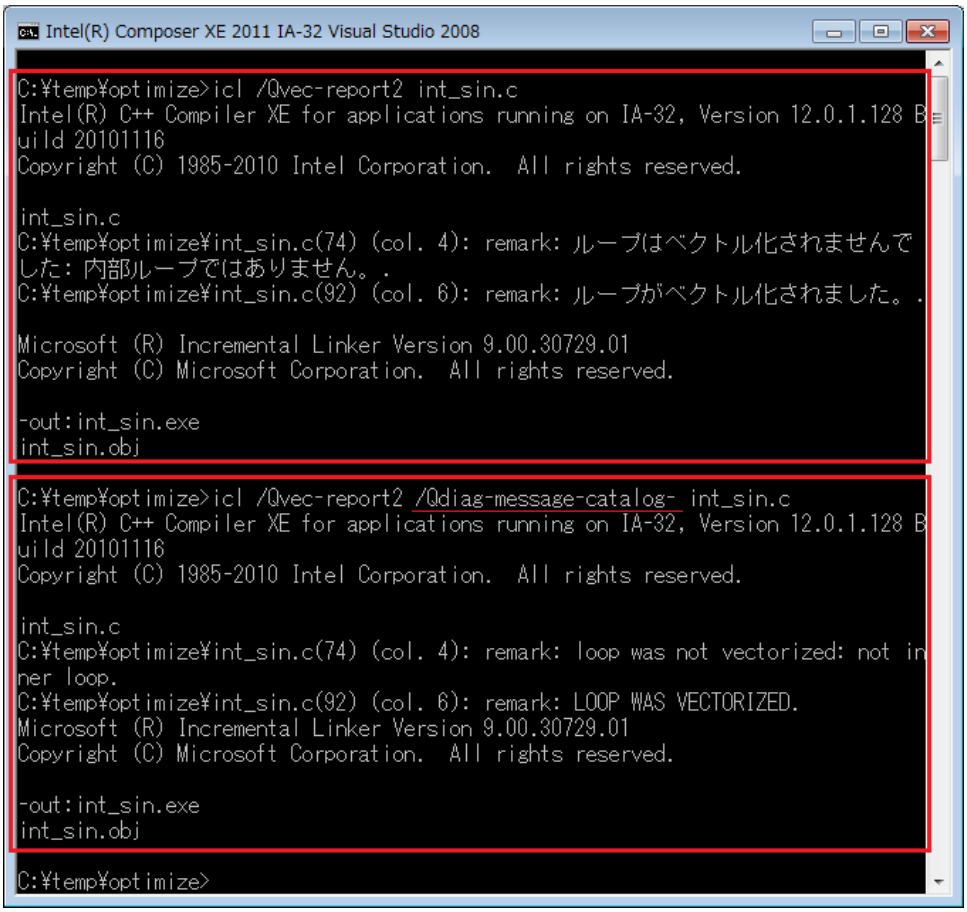
本内容は、特にインテル® C++ コンパイラーの使用方法とは直接関係ありませんが本章でご紹介します。Visual Studio 2010 の出力ウィンドウにはデフォルトでコンパイラーの情報が表示されないためインテル® C++ コンパイラーでビルドが行われているのか確認できません。そこで [ツール] - [オプション] から [オプション] ダイアログを表示して、[プロジェクトおよびソリューション] - [ビルド/実行] から“MSBuild プロジェクト ビルドの出力の詳細”の値を、デフォルトの“最小”から“標準”に変更することによりコンパイラー情報が表示されるようになります。





６－３．コンパイラーメッセージを英語で表示する方法

インテル® C++ Composer XE 2011 日本語版には、日本語と英語の両方のコンパイラーメッセージファイルが含まれており、日本語環境では日本語メッセージがデフォルト出力されています。このコンパイラーメッセージを英語で表示する場合は、“/Qdiag-message-catalog-”というオプションを追加します。



６－４．再配布可能ライブラリーについて

インテル® C++ コンパイラーには複数のライブラリー（静的／動的）が含まれています。アプリケーションを作成する際に“/MD”や“/MDd”オプションでライブラリーを動的リンクした場合は実行環境でインテル® C++ コンパイラーが提供するライブラリーも必要になる場合があります。これらの再配布可能なライブラリーはエンド・ユーザー・ソフトウェア使用許諾契約書に定める契約条件に従う必要があります。インテル® C++ コンパイラーに関する再配布可能なライブラリー情報は、以下のファイルに記載されています。

C:¥Program Files¥Intel¥ComposerXE-2011¥Documentation¥ja_JP¥credist.txt

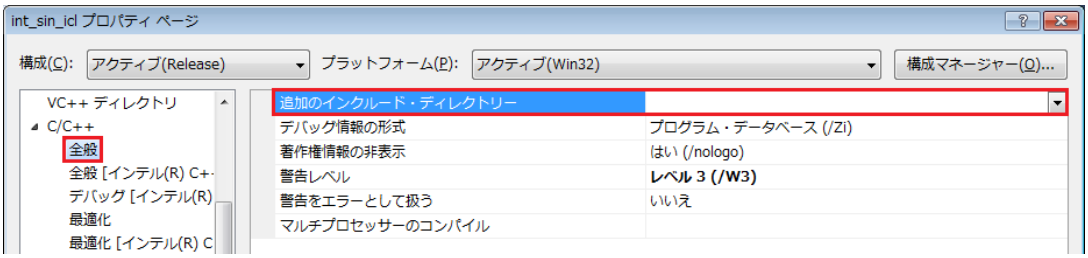
また、インテル® C++ コンパイラーの再配布ライブラリーはインストールパッケージとしても提供されています。詳細は以下のインテル社のウェブサイトを参照してください。

http://software.intel.com/en-us/articles/redistributable-libraries-for-the-intel-c-and-visual-fortran-composer-xe-for-windows/

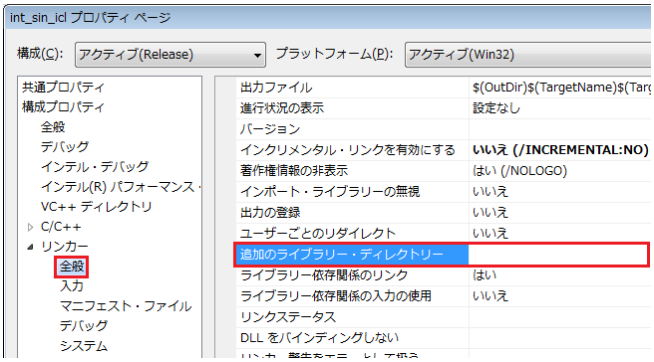
6－5．特定のライブラリーを使用する場合の設定

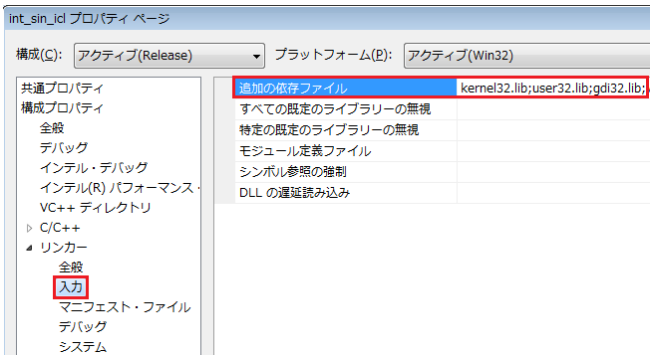
特定のライブラリーVisual Studio IDE から使用する場合は、対象のファイルおよびパスの設定が必要です。これらの設定は プロジェクトの [プロパティ ページ] ダイアログで行います。

まず、ヘッダーファイル・パスの指定は、 [構成プロパティ] - [C/C++] - [全般] を選択して、[追加のインクルード・ディレクトリー] に設定してください。



ライブラリー・ファイル・パスの指定は、 [構成プロパティ] - [リンカー] - [全般] を選択して、[追加のライブラリー・ディレクトリ] に設定してください。また、ライブラリー・ファイルの指定は、 [構成プロパティ] - [リンカー] - [入力] を選択して、[追加の依存ファイル] に設定してください。

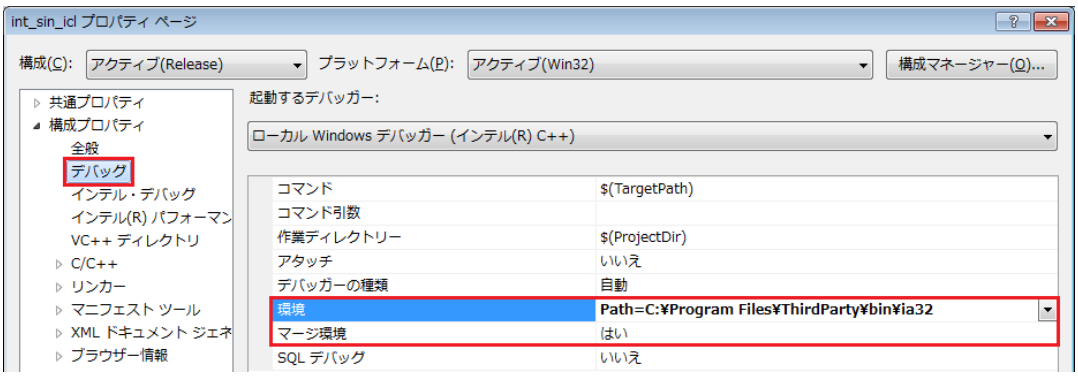




また、アプリケーションのデバッグや実行において、DLL ファイルが参照される場合があります。この DLL への検索パスを解決するには、システム環境変数の Path を使用するか、またはプロジェクトの [プロパティ ページ] ダイアログで [構成プロパティ] - [デバッグ] から [環境] を使用することが出来ます。たとえば以下のように記述して Path 環境変数への指定を行うことができます。

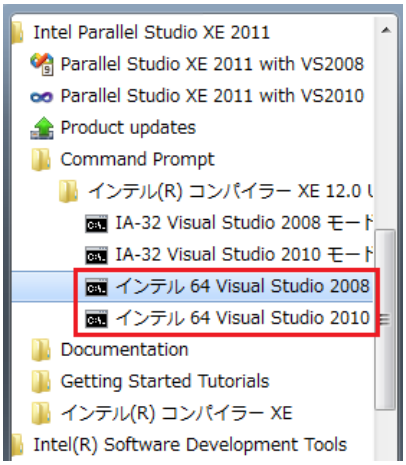
Path=C:¥Program Files¥ThirdParty¥bin¥ia32

なお、同ページ内の [マージ環境] が “はい” に選択されていることを確認してください。



6－6．64 ビット（インテル® 64）対応アプリケーションの作成

コマンドラインからインテル® C++ コンパイラーを使用して 64 ビット対応アプリケーションを作成する場合は、Windows [スタート] メニューから [プログラム] - [Intel Parallel Studio XE 2011] - [Command Prompt] - [インテル(R) コンパイラー XE 12.0 Update 1] - [インテル 64 Visual Studio 2008 モード] などのインテル 64 アーキテクチャー用コマンドプロンプトを選択してください。このコマンドプロンプトでは、インテル® 64 対応アプリケーションをビルドするに必要な環境変数の設定が完了しています。



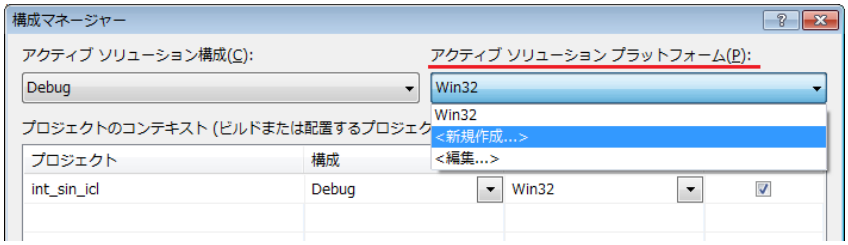
表示されるコマンドプロンプトに、以下のように icl コマンドを実行し、インテル® 64 対応アプリケーション用のインテル® コンパイラーが動作していることを確認してください。

```
> icl
```

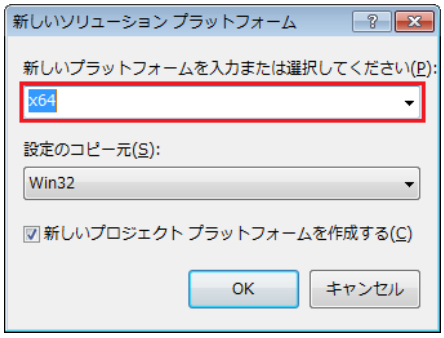
Visual Studio IDE からインテル® C++ コンパイラー を使用して 64 ビット対応アプリケーションを作成する場合は、プロジェクトを作成してから以下の操作を実行してください。

まず、[構成マネージャ] で、[アクティブ ソリューションプラットフォーム] から “<新規作成...>” を選択して [新しいソリューション プラットフォーム] 画面を開きます。同画面で、下図のように新しいプラットフォームに “x64” を選択します。この操作で、ビルド環境が 64 ビット（インテル® 64）に変更されます。Visual Studio IDE のメイン画面に戻り、ビルドを実行します。

図：構成マネージャ

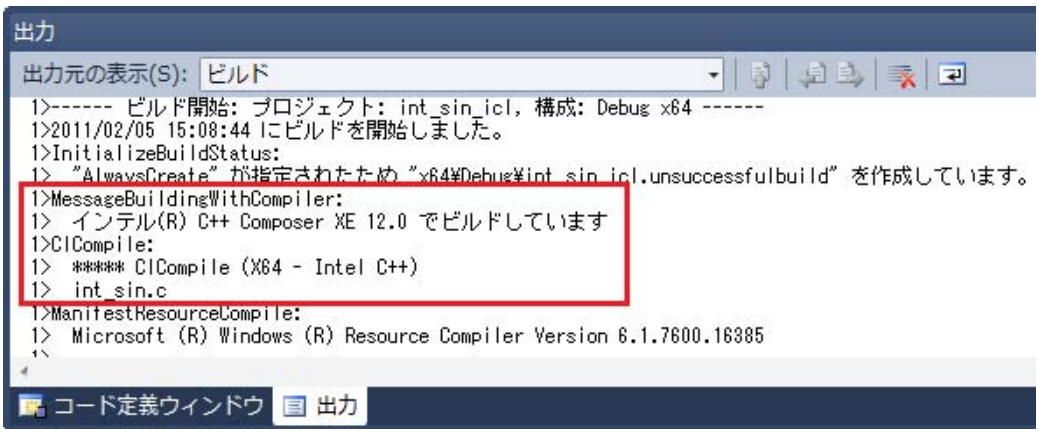


図：新しいソリューション プラットフォーム



ビルドの結果は、[出力] 画面にて確認することができます。

インテル® 64 用インテル® C++ コンパイラー が使用されていることを確認してください。



ご注意：64 ビット対応アプリケーションを作成する場合は、インテル® 64 対応アプリケーション用インテル® コンパイラーのインストールはもちろん、リンク環境である Microsoft Visual Studio IDE 側でも 64 ビット用のライブラリー、その他必要なツールがインストールされている必要があります。Visual Studio 2005 および 2008 Standard Edition はデフォルト・インストールで X64 用ツールがインストールされますが、プロフェッショナル・エディション以上ではインストールする際にカスタム・インストールを選択し（またはアップデートで “機能の追加と削除” を選択し）、以下の [セットアップ オプションページ] 画面にて、”X64 コンパイラおよびツール” をチェックしてインストールする必要があります。

なお、Visual Studio 2010 では、どのバージョンでもデフォルトで x64 開発環境がインストールされます。

7． 追加情報

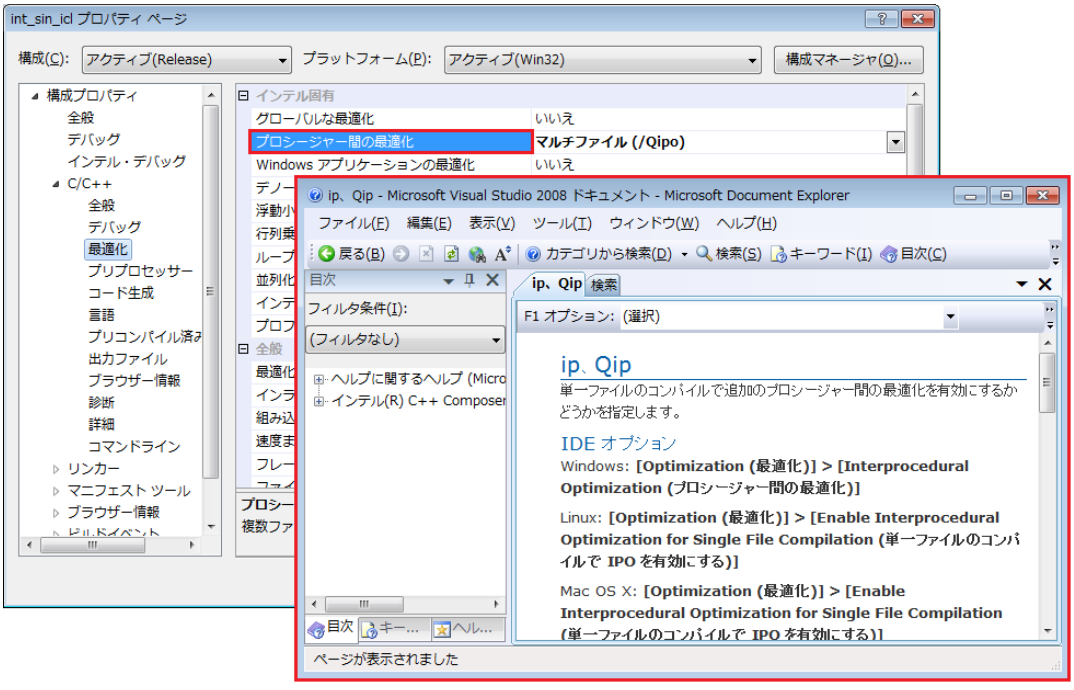
追加情報として、インテル® C++ コンパイラーに関する一般情報をいくつか説明します。

7－1． ドキュメントの参照方法

インストールされるドキュメントを参照するには、Windows [スタート] メニューからアクセスします。本製品に付属するライブラリーへのマニュアルやチュートリアルも参照可能です。

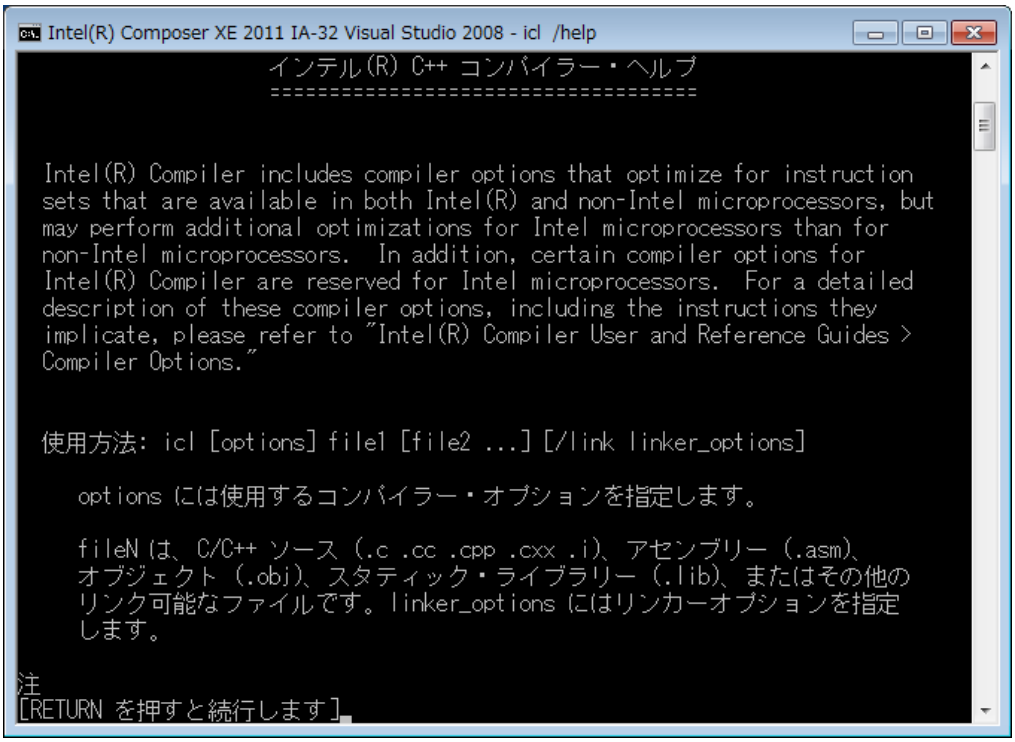
Visual Studio IDE からは [ヘルプ] メニューより『インテル(R) C++ Composer XE ヘルプ』への参照が可能です。

また、Visual Studio IDE から、”F1” キーによる状況依存ヘルプ機能が利用できます。これは、例えば以下のようにプロジェクトの [プロパティ ページ] で [プロシージャー間の最適化] をクリックしてフォーカスを置いた状態で、キーボードの “F1” キーを押下することにより、『インテル® C++ コンパイラー・ドキュメント』がポップアップされてプロシージャー間の最適化（IPO）に関するオプションの内容が表示されます。



コマンドラインからは、以下のコマンドにてコンパイラーオプションなどに関するヘルプが表示されます。

> icl /help



7－2．サンプルコード

本ドキュメントでは、サンプルプログラム（optimize）を使用しましたが、インテル® C++ コンパイラーには、この他にも以下のようなサンプルが用意されています。

- Intel Cilk Plus サンプル
- ガイド付き自動並列化用サンプル
- 組み込み関数を使用したサンプル
- PGO オプション用サンプル
- IPO オプション用サンプル
- OpenMP* を使用したサンプル
- ベクトル化オプション用サンプル
- 複数の並列化モデルを使用したサンプル

サンプルコードに関する詳細は、以下の “samples.htm” を参照してください。

C:¥Program Files¥Intel¥ComposerXE-2011¥Samples¥ja_JP¥C++¥samples.htm

7－3．環境変数について

インテル® コンパイラーの使用において基本的な環境変数は自動で設定されますが、環境変数について知ることはトラブルシューティングに役立ちます。ここではインテル® コンパイラーによる開発に必要な２つの環境変数、「ビルド環境変数」と「ラインタイム環境変数」について説明します。

① 「ビルド環境変数」・・・アプリケーションのビルド時に、開発環境やコンパイラー、リンカーによって必要とされる環境変数。ビルド環境変数には以下のものが含まれます。

- PATH： コンパイラーやリンカーなど各種実行ツールが存在するディレクトリー
- INCLUDE：ヘッダーファイルが存在するディレクトリー
- LIB： 静的ライブラリー・ファイル（.lib）が存在するディレクトリー

② 「ランタイム環境変数」・・・アプリケーションの実行時に、アプリケーションにより参照されるシステム環境変数。ランタイム環境変数には以下のものが含まれます。

- Path： 動的ライブラリー・ファイル（.dll）が存在するディレクトリー

これらの環境変数は、“コマンドラインにおける作業”と“Visual Studio 開発環境における作業”とで設定方法が異なります。

＜コマンドラインにおける作業＞

まず、コマンドラインにおける「ビルド環境変数」は、インテル® コンパイラーが提供する以下の環境変数設定バッチファイルを実行することで設定することができます。

C:¥Program Files¥Intel¥ComposerXE-2011¥bin¥compilervars.bat arch [vs]

このバッチファイルを実行するためには、引数を 2 つ指定します。

一つ目の“arch”は必須引数で、ターゲットのアーキテクチャーを指定します。

この引数の値は、以下のいずれかを指定します。

ia32 ・・・ IA-32 / Intel64 ホストシステムで IA-32 用アプリケーションをコンパイルする場合

ia32_intel64 ・・・ IA-32 ホストシステムで Intel64 用アプリケーションをクロスコンパイルする場合

intel64 ・・・ Intel64 ホストシステムで Intel64 用アプリケーションをコンパイルする場合

2 つ目の“vs”引数は、使用する Visual Studio のバージョンを指定します。この引数は必須ではなく、省略した場合は、統合に指定された Visual Studio のバージョンが採用されます。

この引数の指定可能な値は以下のいずれかです。

vs2005 ・・・ Visual Studio 2005 のエディションを使用する場合

vs2008 ・・・ Visual Studio 2008 のエディションを使用する場合

vs2010 ・・・ Visual Studio 2010 のエディションを使用する場合

通常インテル® コンパイラーでは、これらの環境変数設定バッチファイルは、インテル® コンパイラー専用コマンドプロンプトを起動した際に、適切な引数を使用して実行されます。なお、本ドキュメントのコマンドラインで起動したコマンドプロンプトでは、以下のように環境変数設定バッチファイルが実行されます。

C:¥Program Files¥Intel¥ComposerXE-2011¥bin¥compilervars.bat ia32 vs2008

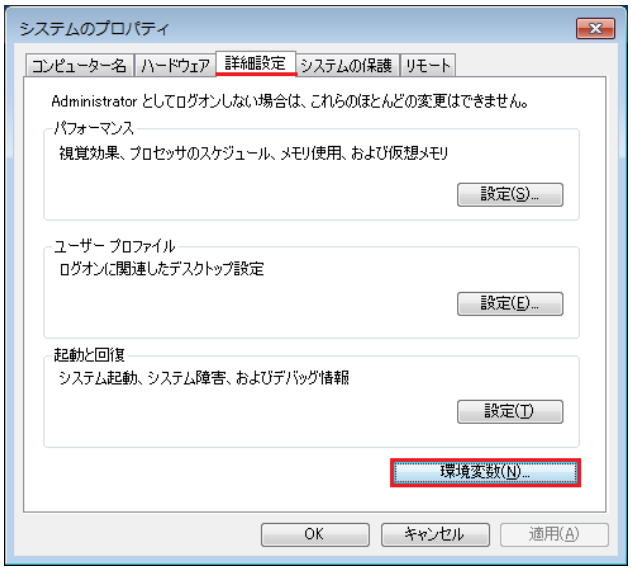
一方、コマンドラインにおける「ランタイム環境変数」は、上記で説明したビルド環境変数のバッチファイルが実行されたインテル® コンパイラー専用コマンドプロンプト環境では特に設定する必要はありませんが、それ以外の例えば Windows のデフォルトのコマンドプロンプトを使用してアプリケーションを実行する場合はシステム環境変数（PATH）を別途確認する必要があります。

このシステム環境変数は、[コンピュータ] を右クリックして表示されるメニューから [プロパティ] を選択して、続いて左メニューから [システムの詳細設定] を選択します（Windows XP では、[システムの詳細設定] の選択は不要）。そして、[システムのプロパティ] ダイアログの [詳細設定] タブから [環境変数] ボタンをクリックし、[環境変数] ダイアログのシステム環境変数の Path で確認することができます。

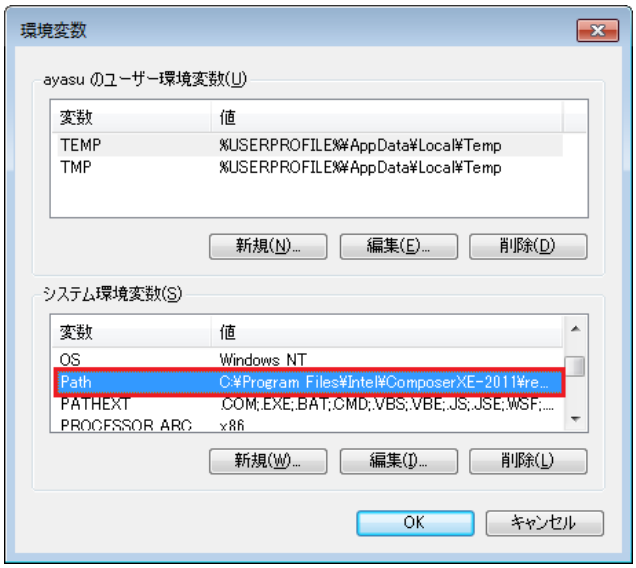
通常、この Path には、インテル® コンパイラーをインストールした時点で適切な値が設定されています。例えば、IA-32 システムにインストールした場合は以下のインテル® コンパイラーの提供するライブラリーへのパスがシステム環境変数に設定されています。

C:¥Program Files¥Intel¥ComposerXE-2011¥redist¥ia32¥compiler

図：システムのプロパティ



図：システム環境変数



＜Visual Studio 開発環境における作業＞

Visual Studio 開発環境における「ビルド環境変数」は、VS2005/2008 と VS2010 とで設定箇所が異なります。

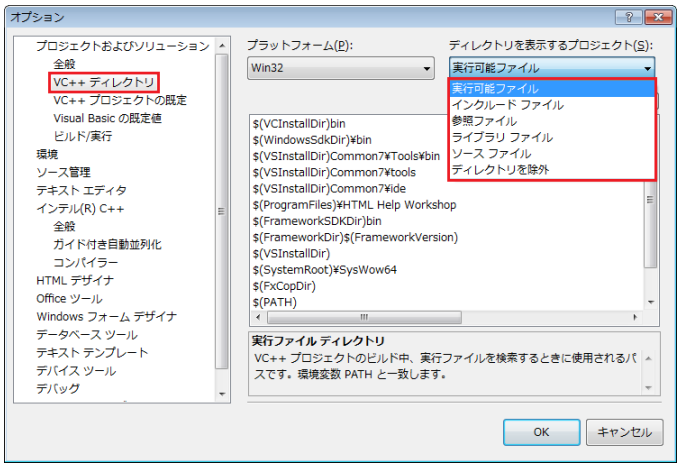
まず、VS2005/2008 では、Visual Studio のメニューより、[ツール] - [オプション] を選択して表示される [オプション] ダイアログで設定します。

この [オプション] ダイアログは、マイクロソフト・コンパイラー用の環境変数とインテル® コンパイラー用の環境変数設定画面が存在します。マイクロソフト・コンパイラー用の設定画面は、左のメニューから [プロジェクトおよびソリューション] - [VC++ディレクトリ] で表示され、またインテル® コンパイラー用の設定画面は、[インテル(R) C++] - [コンパイラー] にて表示されます。

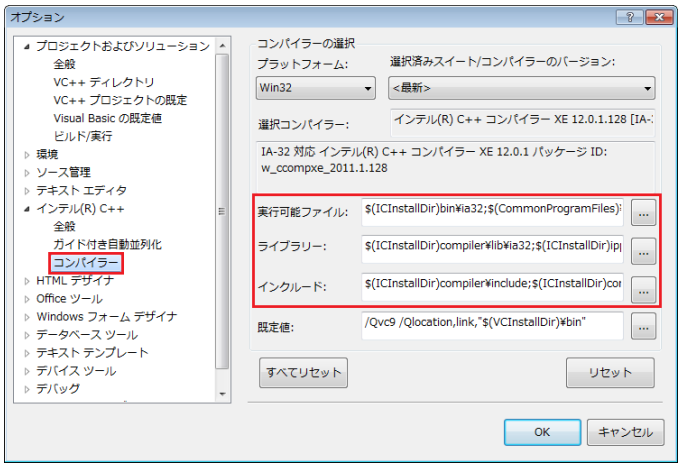
これらのビルド環境変数は、デフォルトで適切な値に設定されているため、特に変更する必要はありませんが、特定のビルド環境を設定する場合は、マイクロソフト・コンパイラー用の環境変数に追加設定してください。

インテル® コンパイラーは、インテル® コンパイラー用の環境変数設定値のほかに、マイクロソフト・コンパイラー用の環境変数値も参照しています。

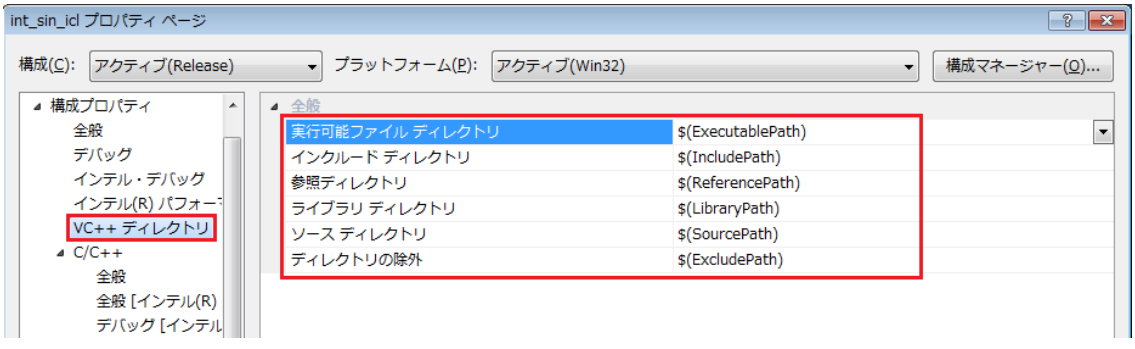
図：マイクロソフト・コンパイラー用の環境変数



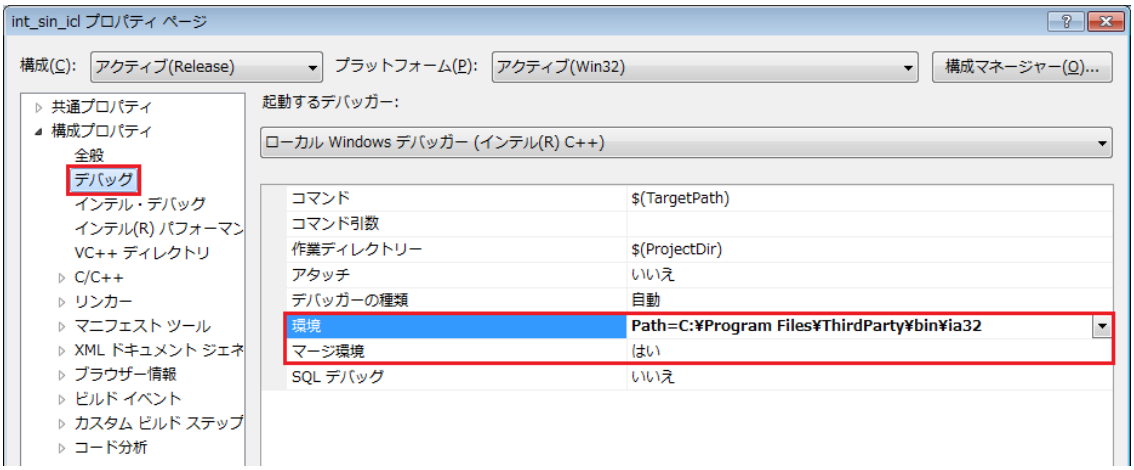
図：インテル® コンパイラー用の環境変数



VS2010 では、プロジェクトの [プロパティ ページ] を開いて [構成プロパティ] - [VC++ ディレクトリ] にて設定します。マイクロソフト・コンパイラーとインテル® コンパイラーとで共通の設定場所となります。



一方、Visual Studio 開発環境における「ランタイム環境変数」は、コマンドラインにおけるランタイム環境変数と同様 Windows のシステム環境変数として設定するか、または下図のようにプロジェクトの [プロパティページ] ダイアログで [構成プロパティ] - [デバッグ] から [環境] を使用することが出来ます。なお、同ページ内の [マージ環境] が “はい” に選択されていることを確認してください。



Visual Studio のメニューから、[デバッグ] - [デバッグなしで開始] を選択してアプリケーションを実行させた際に、ライブラリーの参照エラーが表示される場合は、これら「ランタイム環境変数」の設定値を確認してください。

なお、インテル® コンパイラーではインストールの際に、適切なランタイム設定値がシステム環境変数 (Path) に自動で設定されます。

8. 最後に

日本語環境に関する問題
http://www.xlsoft.com/jp/products/intel/tech/win_jp_limitation.html

評価版ダウンロード
http://www.xlsoft.com/jp/products/intel/download.html

「インテル® C++ Composer XE Windows 版」弊社メインサイト
http://www.xlsoft.com/jp/products/intel/compilers/ccw/index.html

本製品ご使用にあたって、ご不明な点がありましたら、下記お問い合わせ窓口より弊社サポートまでご連絡ください。
https://www.xlsoft.com/jp/services/xlsoft_form.html