インテル® C++ コンパイラー 11.0 Windows* 版 プロフェッショナル・エディション

ー 入門ガイド ー

エクセルソフト株式会社
www.xlsoft.com

－　目次　－

1．はじめに

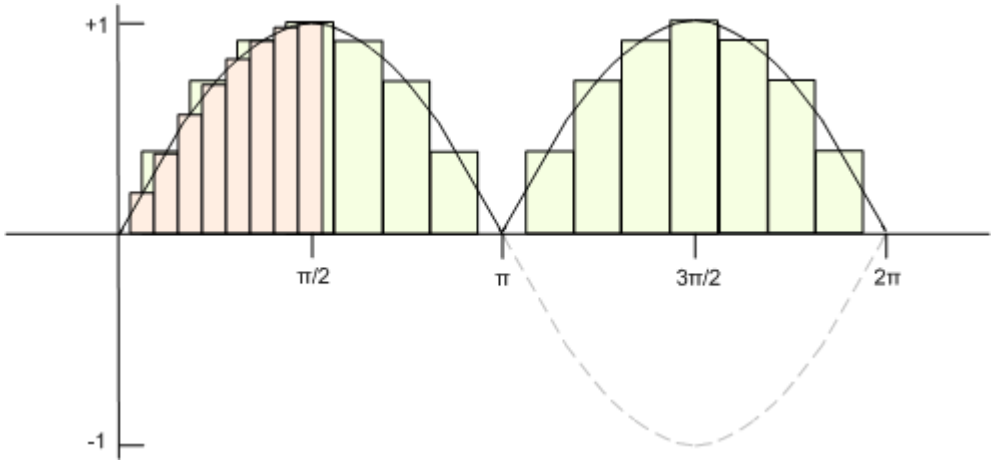
インテル® C++ コンパイラー 11.0 プロフェッショナル・エディションのインストールが完了したら、適切なインストール、設定、およびインテル® C++ コンパイラーの基本動作を確認するため、本ドキュメントで説明する動作検証を行ってください。この動作検証は、コマンドラインおよび Microsoft* Visual Studio* 統合開発環境（IDE）を使用して行います。なお、この検証では IA-32 対応アプリケーション用インテル(R) C++ コンパイラー、および以下に示すサンプルコードを使用します。

<install-dir>¥Compiler¥11.0¥xxx¥cpp¥Samples¥C++¥optimize¥int_sin.c

Note： <Install-Dir> は、インテル® C++ コンパイラーのインストール・パスです。デフォルトでは、"C:¥Program Files¥Intel" です。また、"xxx" はマイナーバージョンを示しています。

1－1．検証用サンプルコード

検証用サンプルコードは、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。次の図は、2^4 内点と 2^5 内点の最初の 8 片で何が計算されているかを示しています。



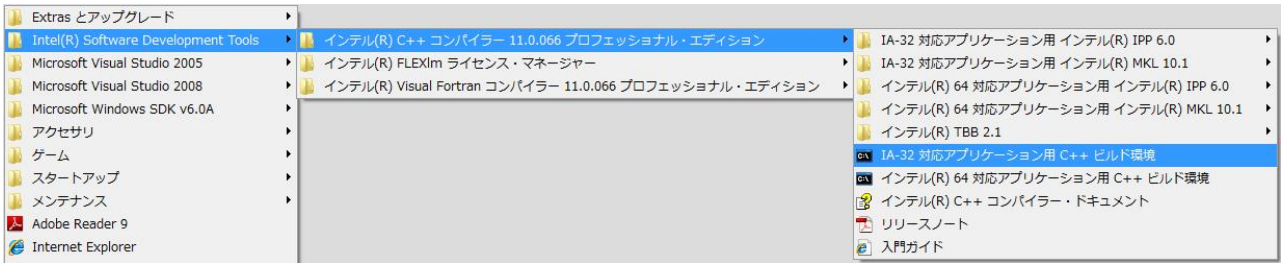
検証用サンプルコードをコンパイルして実行し、出力が既知の正しい値である 4 に収斂するかどうかをチェックすることで、コンパイラーが適切にインストールされたかどうかを確認できます。また、このサンプルコードには、マイクロソフトの標準関数ライブラリーに加えて、インテルの数値演算ライブラリーを使用しているため、インテル・ライブラリーが正しくインストールされているかどうかも確認することができます。サンプルコード内の時間関数はプログラム実行の開始から終了までを測定したアプリケーション・クロックの数を返します。

2. コマンドラインからの操作方法

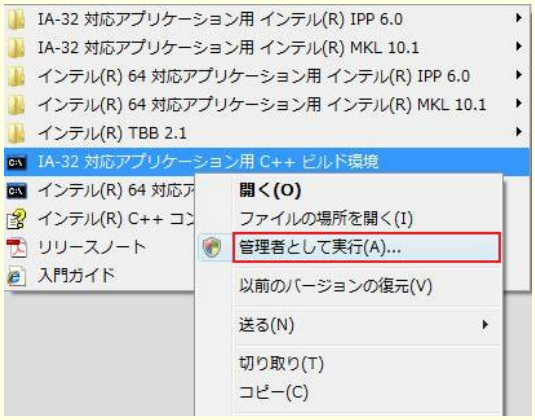
インテル® C++ コンパイラーは、icl コマンドを使用してコマンドラインから実行します。作業の大部分をコマンドラインからではなく、Microsoft Visual Studio IDE を使用して行っている場合でも、このセクションをスキップせず、動作検証を行うことをお勧めします。
以下の手順に従ってコマンドラインにおける動作検証を行ってください。

1. Windows [スタート] メニューから [プログラム] – [Intel(R) Software Development Tools] – [インテル(R) C++ コンパイラー 11.0.xxx プロフェッショナル・エディション] - [IA-32 対応アプリケーション用 C++ ビルド環境] を選択して、インテル® C++ コンパイラー専用コマンドウィンドウを開きます。このウィンドウでは、ビルドに必要な環境変数（PATH、LIB、INCLUDE）の設定が完了しています。これは、コマンドウィンドウ起動時に以下のバッチファイルが内部で実行されるからです。

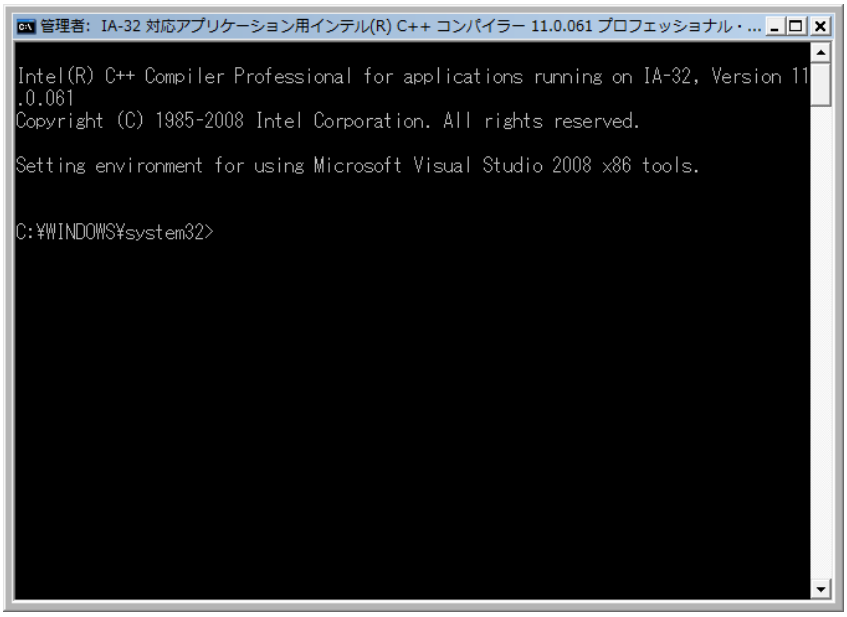
   “<install-dir>¥Compiler¥11.0¥xxx¥cpp¥bin¥iclvars.bat ia32”



ご注意：Microsoft Windows Vista* を使用している場合は、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。

```
prompt> 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等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 コマンドのビルド過程でコールされます。

```
prompt> 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。インテル® C++ コンパイラーのサポートするビルド環境を確認の上、インテル® コンパイラーの再インストールを行ってください。

> Note： icl コマンドは、内部でインテル® C++ コンパイラー（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 はビルド工程をコントロールしているので一般的にインテル® コンパイラー・ドライバーと呼ばれています。

4. カレント・ディレクトリーを int_sin.c 検証用サンプルコードが存在するフォルダーまで移動します。

```
prompt> cd <install-dir>¥Compiler¥11.0¥xxx¥cpp¥Samples¥C++¥optimize
```

２－１．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、サンプルコードをコンパイルしてください。

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

> Note：インテル® C++ コンパイラーは、特にオプションが指定されなかった場合、デフォルトでいくつかの最適化オプションが有効になります。そのため、最適化なしでコンパイルする場合は、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。

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

２－２．実行/プログラムの検証

実行プログラムは、サンプルコード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てください。

```
prompt> int_sin.exe
```

各計算で消費される実行時間 (プロセッサー・クロック・サイクルの数) は、内点の数が増えると、計算された整数値 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 = 1.095100e+004

２－３．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。ここでは、次のようにインテル® コンパイラーのデフォルトの最適化オプションを使用してコンパイラーを実行してください。

```
prompt> icl int_sin.c
```

デフォルトの最適化オプションには以下のような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 の命令を搭載したプロセッサーに特化した最適化オプション

２－４．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。

```
prompt> int_sin.exe
```

最適化を行わなかった場合と、アプリケーション・クロックの数を比較します。実際の時間の差は使用するアーキテクチャーに依存します。

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

    Application Clocks     = 1.918000e+003
```


Note：インテル® C++ コンパイラーは、デフォルトで最適化オプション /arch:SSE2 を使用しているため、作成される実行形式ファイルは、SSE2 命令を搭載したプロセッサーでのみ実行可能です。それ以外のプロセッサー、例えば、インテル® Pentium III などの SSE2 命令を持たないプロセッサーで動作可能な実行形式ファイルを作成する場合は、/Od オプションにてデフォルト最適化オプションを無効にするか、または /arch:IA32 オプションを指定して以下のようにコンパイルしてください。

```
prompt> icl /arch:IA32 int_sin.c
```

２－５．コンパイル（最適化オプションあり）－ その２

ここでは、さらに有効な最適化オプションを２つ試してみます。
１つ目は、/O2 よりもさらに強力な /O3 オプションです。
２つ目は、SSE の命令を搭載するインテル® プロセッサーに特化したベクトル化と呼ばれる /QxHost オプションです。このオプションは、プログラム内のループ処理を対象に最適化を行いますが、すべてのループ処理がベクトル化されるとは限りません。本オプションは、各ループ処理を診断しベクトル化可能であると判断されたループに対してのみ最適化が実行されます。


Note：インテル® コンパイラーの提供するベクトル化オプションは、以下のように SSE のバージョンごとに分かれています。

/QxSSE2、/QxSSE3、 /QxSSSE3、 /QxSSE4.1、 /QxSSE4.2

/QxHost オプションは、コンパイルが実行されるシステムのプロセッサーが所有する最新の SSE 命令に対応した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。例えば、コンパイル作業を行う開発システムが SSSE3 を搭載するインテル® Core™2 Duo プロセッサーなどの場合は、/QxHost オプションは /QxSSSE3 オプションに置き換えられます。なお、作成される実行形式ファイルは、SSSE3 に特化したバイナリーコードであるため、実行環境は少なくとも SSSE3 命令を搭載したプロセッサーである必要があります。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。これら /Qx 系オプションに対し、インテル® コンパイラーは、/Qax 系というオプションが用意されています。このオプションを使用した場合は、作成される実行形式ファイルは、SSE 命令に特化したコードに加え、汎用コードも追加されるので実行環境を特定しません。このオプションも SSE のバージョンごとに以下のように分かれています。

/QaxSSE2、/QaxSSE3、 /QaxSSSE3、 /QaxSSE4.1、 /QaxSSE4.2

/arch オプションも SSE 命令を使用したベクトル化オプションで、SSE の各バージョン単位でオプションが存在しますが、/Qx、/Qax オプションはよりインテル® プロセッサーに最適なバイナリーを生成するように設計されています。

ここでは、ベクトル化オプションとして /QxHost を使用してコンパイルを行いますが、特定のベクトル化オプションを直接指定しても構いません。ベクトル化オプションに関する詳細は、インストールされるコンパイラー・マニュアル、または以下の『インテル® コンパイラー最適化クイック・リファレンス・ガイド 』を参照してください。

http://jp.xlsoft.com/documents/intel/compiler/qr_guide_jp.pdf

では、/O3 オプションと、/QxHost オプションを指定して以下のようにコンパイルしてください。

```
prompt> icl /O3 /QxHost int_sin.c
```

Note：インテル® コンパイラーはデフォルトのコンパイルオプションを有しますが、オプションが指定された場合は、その指定されたオプションと同種類のデフォルトオプションが置き換えられます。この例の場合、デフォルトオプション /O2 が /O3 に置き換えられ、/arch:SSE2 が /QxHost オプションに置き換えられます。

Note：インテル® コンパイラーは、ベクトル化の診断情報を表示するオプションがあります。以下のように /Qvec-report オプションに診断レベルの番号を指定してコンパイルを実行すると、プログラム内の各ループし対して診断情報を表示することができます。

```
prompt> icl /O3 /QxHost /Qvec-report:3 int_sin.c
```

２－６．実行/パフォーマンスの比較 － その２

実際にベクトル化された int_sin.exe プログラムを実行し、結果を比較してください。

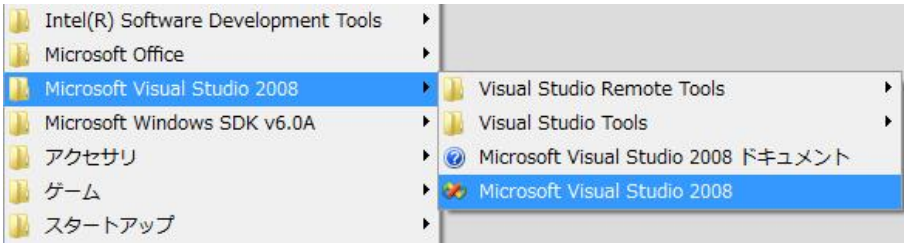
```
prompt> int_sin.exe

            :
            :
---------------------------------------
     8388608        |    4.000000e+000    |
---------------------------------------
    16777216        |    4.000000e+000    |
---------------------------------------
    33554432        |    4.000000e+000    |
---------------------------------------
    67108864        |    4.000000e+000    |

    Application Clocks     = 1.794000e+003
```

Note：実行結果は、搭載されるプロセッサーの SSE バージョンによりますので、場合によってはデフォルトオプションの結果と変わらない場合もあります。

3． Microsoft Visual Studio IDE からの操作方法

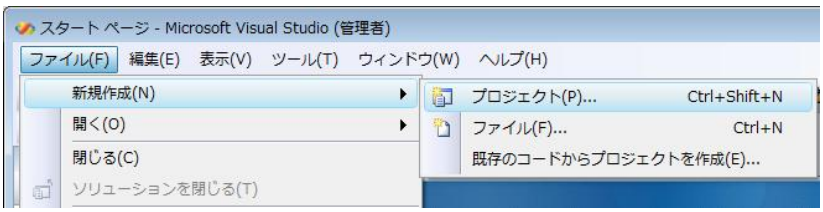
インテル® C++ コンパイラーを Microsoft Visual Studio 環境で使用する手順を説明します。Microsoft Visual Studio 環境を使用する場合は、まずプロジェクトを作成し、ビルド環境を設定する必要があります。ここでは、Microsoft Visual Studio 環境として、Visual Studio 2008（以下、VS2008）を使用して説明します。なお、サンプルコードはコマンドライン同様、“int_sin.c” を使用します。
それでは、以下の手順にしたがって Microsoft Visual Studio 環境における動作検証を行ってください。

1. まず、Windows [スタート] メニューから VS2008 を起動します。

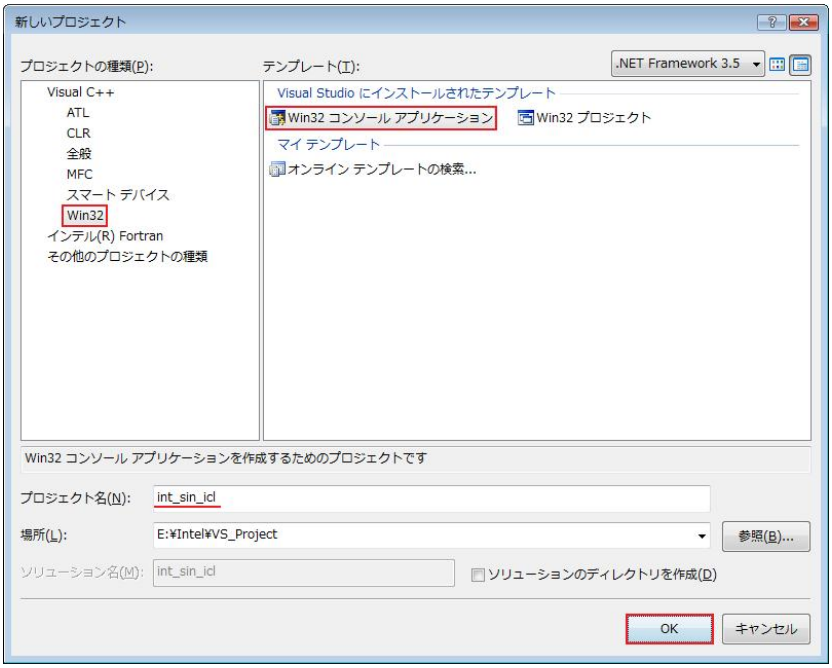


ご注意： Microsoft Windows Vista を使用している場合は、下図のようにショートカットを右クリックして表示されるメニューから [管理者として実行] を選択してください。

2. 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。

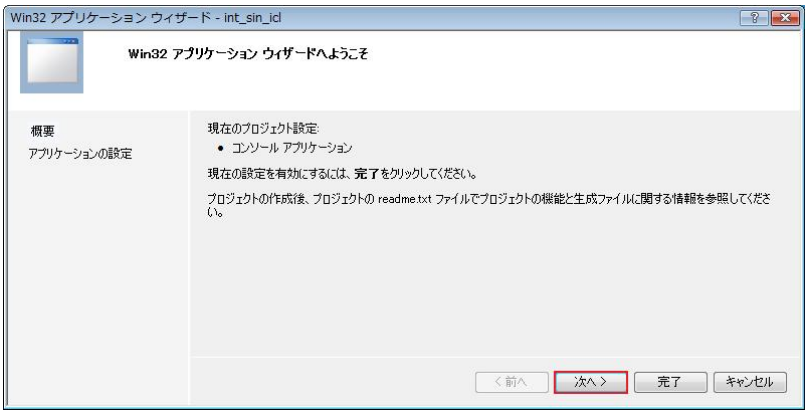


図：新しいプロジェクト



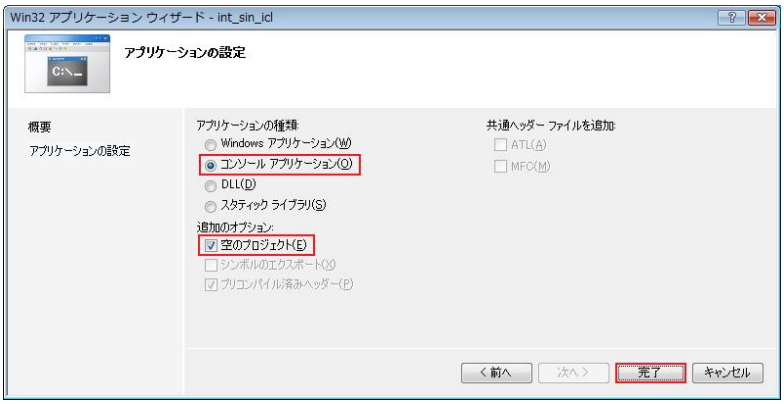
3. [Win32 アプリケーション ウィザード] が表示されるので、[次へ] をクリックします。

図：Win32 アプリケーション ウィザード

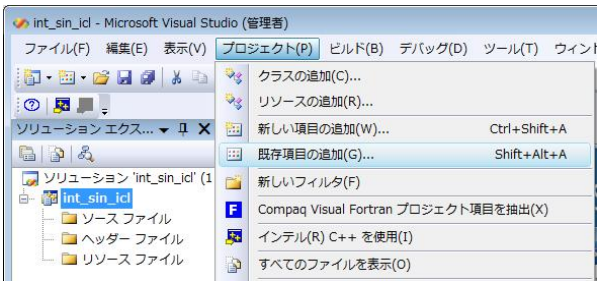


4. [アプリケーションの設定] ダイアログで、下図のように [コンソール アプリケーション] および [空のプロジェクト] が選択されていることを確認し、[完了] をクリックして "int_sin_icl" プロジェクトの作成を完了します。

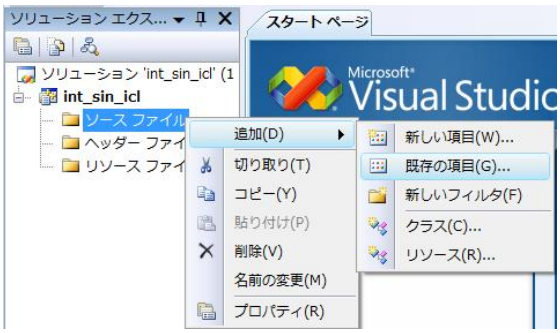
図：アプリケーションの設定



5. 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。



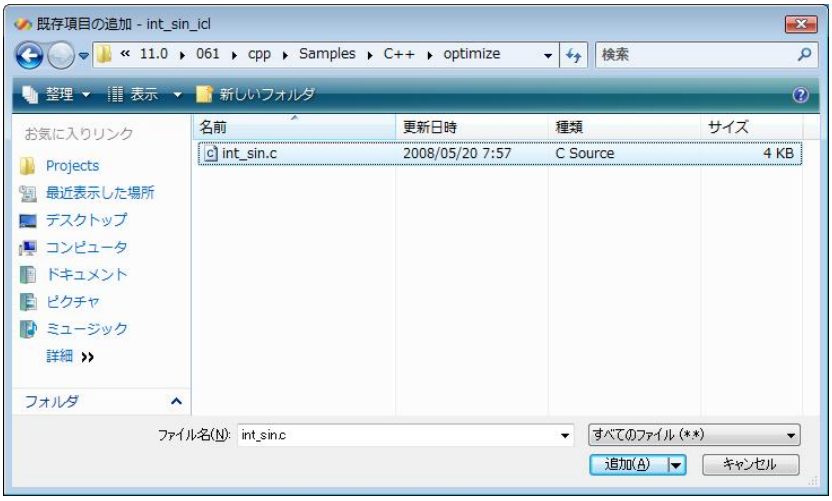
または、



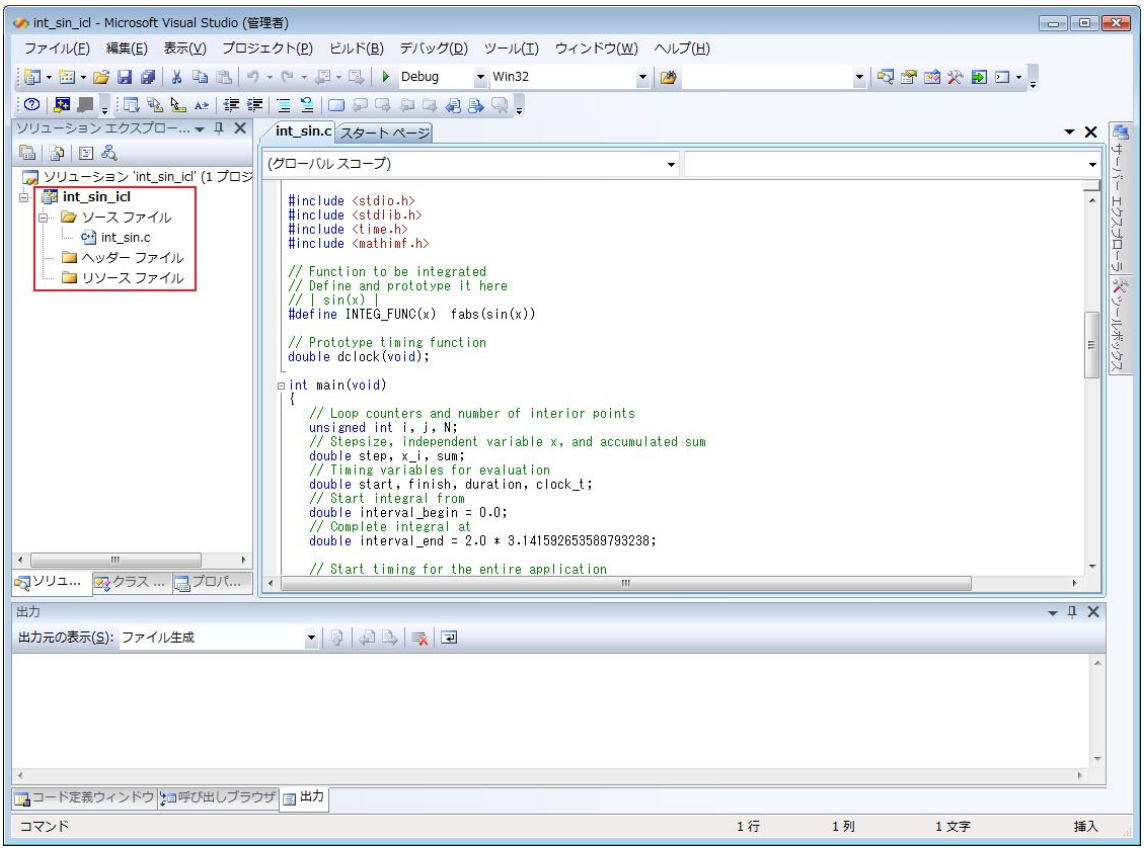
表示される [既存項目の追加] ダイアログで以下のサンプルコードを選択して [追加] ボタンをクリックします。

<install-dir>¥Compiler¥11.0¥xxx¥cpp¥Samples¥C++¥optimize¥int_sin.c

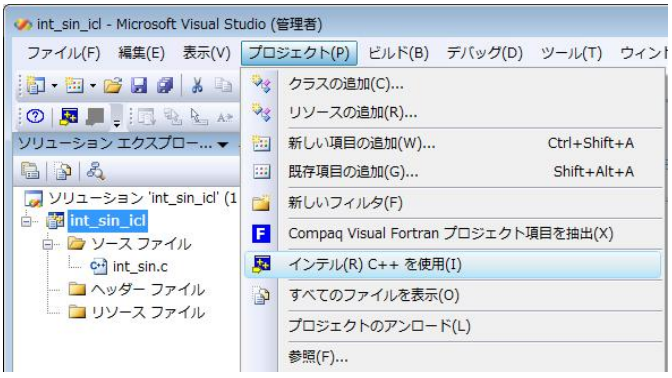
図：既存項目の追加



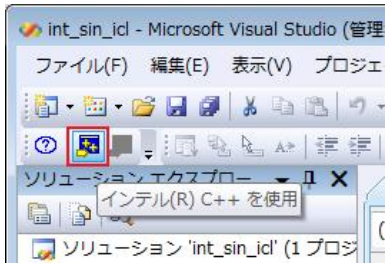
6. 新しいプロジェクト int_sin_icl の “ソースファイル” に、サンプルコード “int_sin.c” が追加されたことを確認します。



7. 次に、プロジェクトの変換を行います。 [プロジェクト] メニューから [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++ プロジェクトへ変換] ボタンをクリックします。



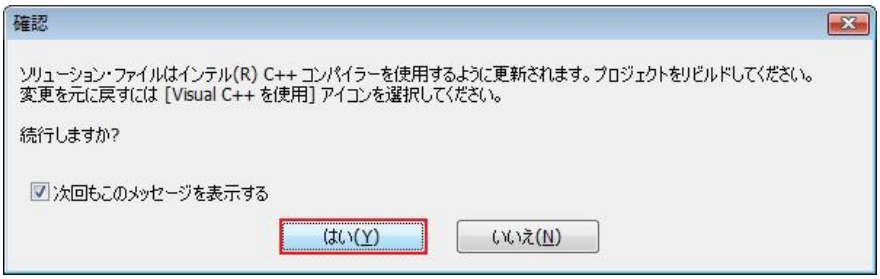
または、



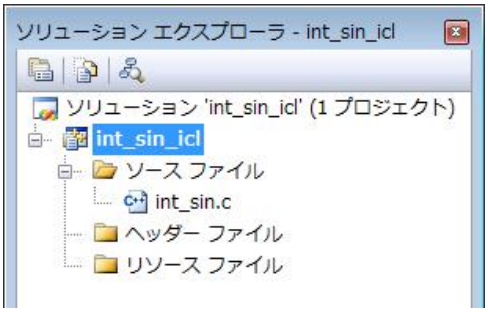
Note： [ソリューション エクスプローラ] からプロジェクトを右クリックして表示されるメニューから、[インテル(R) C++ を使用] を選択することもできます。

8. 表示される [確認] ダイアログで [はい] をクリックします。 変換が成功すると、Microsoft Visual C++* プロジェクトがインテル® C++ プロジェクトに変換され、新しいインテル® C++ プロジェクト・ファイル (.icproj) が作成されます。また、[ソリューション エクスプローラ] にインテル® C++ プロジェクトのロゴが表示されます。

図：確認

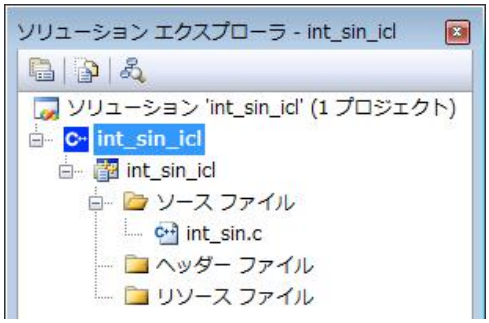


図：Microsoft Visual C++ プロジェクト

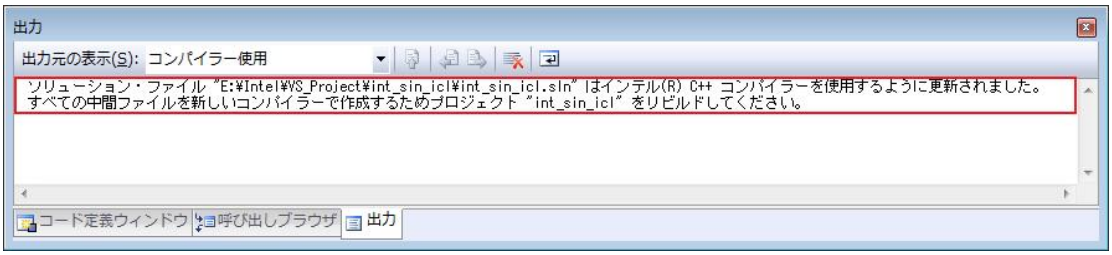


プロジェクトの変換

図：インテル® C++ プロジェクト



プロジェクトの変換の結果は、[出力] 画面にも表示されます。

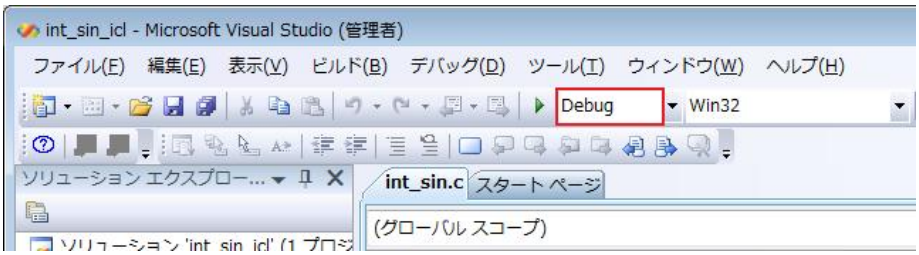


Note：インテル® C++ コンパイラーでは、.NET プロジェクトのようなマネージドコードを生成するプロジェクトをサポートしていないため、これらのプロジェクトは変換できません。

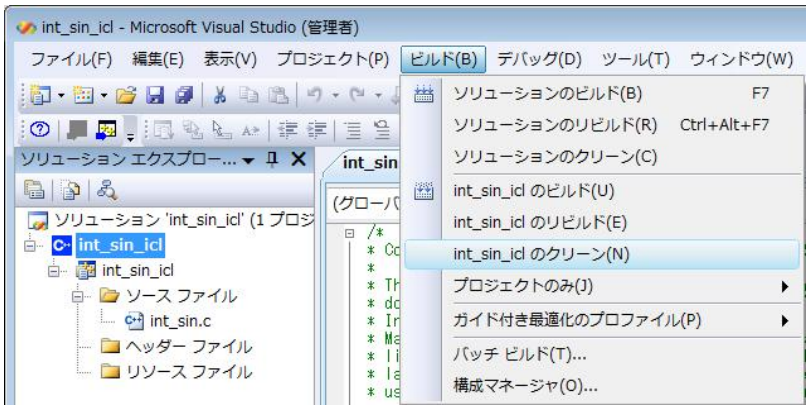
３－１．ビルド（最適化オプションなし）

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。

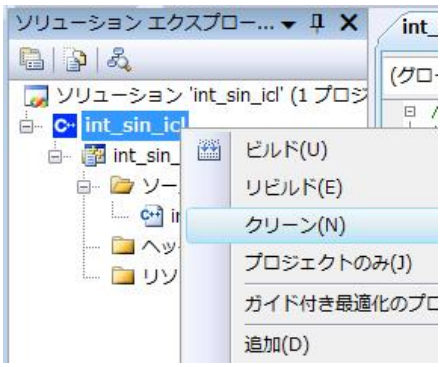
1. まず、プロジェクトの構成が、”Debug” 構成であることを確認してください。



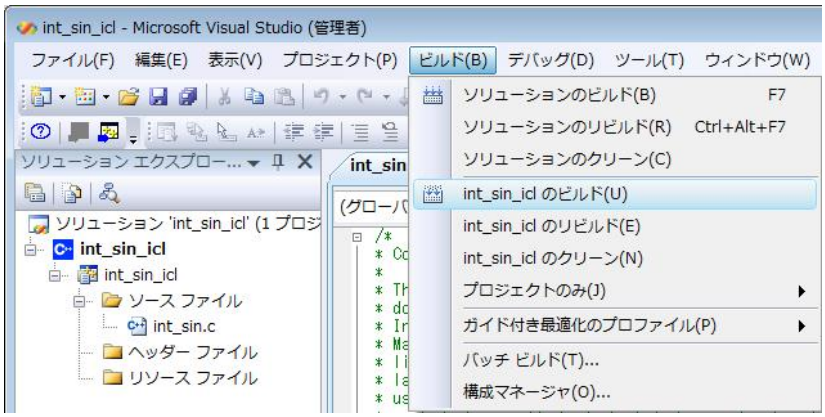
2. 作成したプロジェクトをビルドする前に、プロジェクトの内容を初期化します。VS2008 のメニューから、[ビルド] - [int_sin_icl のクリーン] を選択するか、または [ソリューション エクスプローラ] からプロジェクトを右クリックして表示されるメニューから、[クリーン] を選択します。



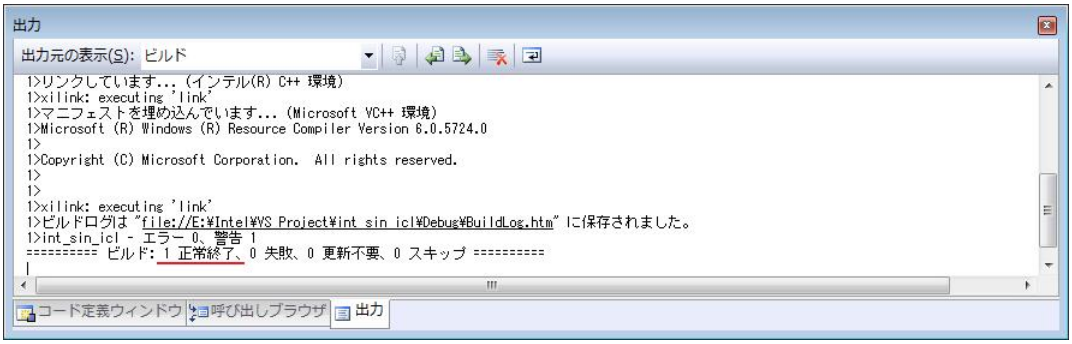
または



3. 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。



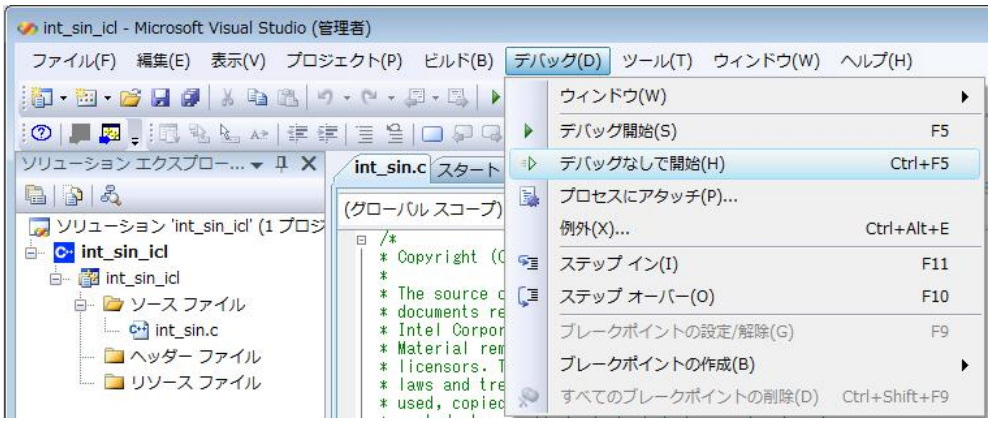
または

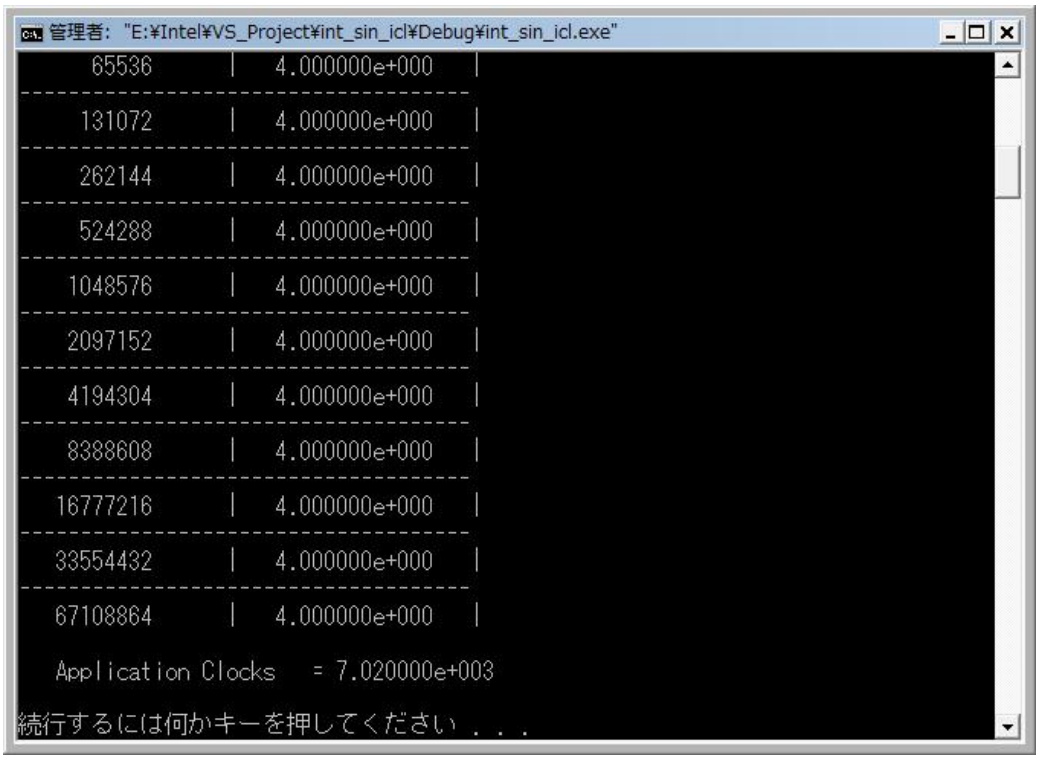


Note : プロジェクトには通常、Debug 構成と Release 構成という 2 種類のプロジェクト構成（ビルド設定環境）が用意されています。一般的に開発中のプロジェクトは Debug 構成で作業を行い、開発が完了した製品を Release 構成でビルドします。デフォルトのプロジェクト構成は Debug 構成で、プロジェクトは最適化なしで、シンボリック・デバッグ情報付きでビルドされます。これはコマンドラインから、 icl /Od /Zi int_sin.c と入力した場合とほぼ同じです。

３－２．実行/プログラムの検証

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。





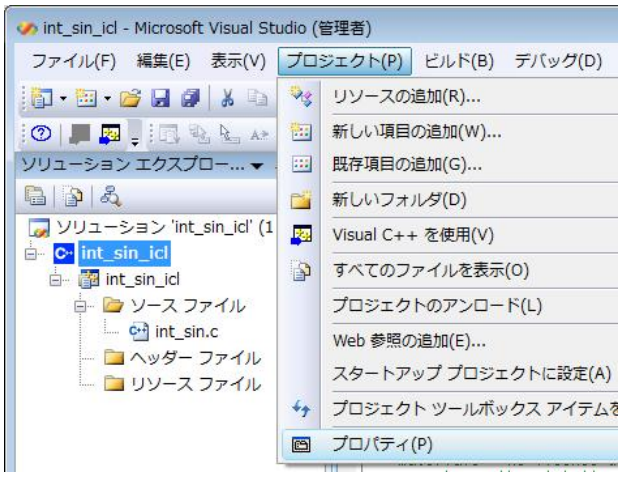
2. プログラム実行に使用された CPU 時間をメモします。

ご注意：プログラムの実行が正常に行われない場合は、Windows のシステム環境設定に問題があります。一度本製品をアンインストールして、以下の『インテル® C++ コンパイラー 11.0 インストール・ガイド』を参考に、再インストールをお試しください。
http://jp.xlsoft.com/documents/intel/cwin/ICC_J_Install.pdf

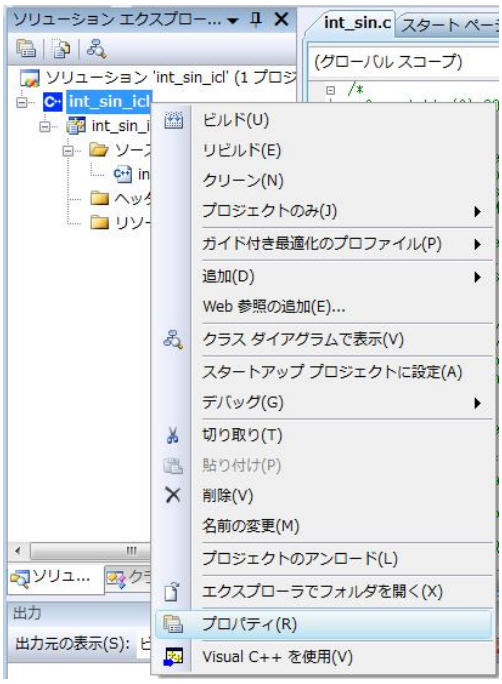
３－３．ビルド（最適化オプションあり）

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. まず、[プロパティ ページ] ダイアログを表示します。VS2008 のメニューから [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）を右クリックして表示されるメニューから、[プロパティ] を選択します。

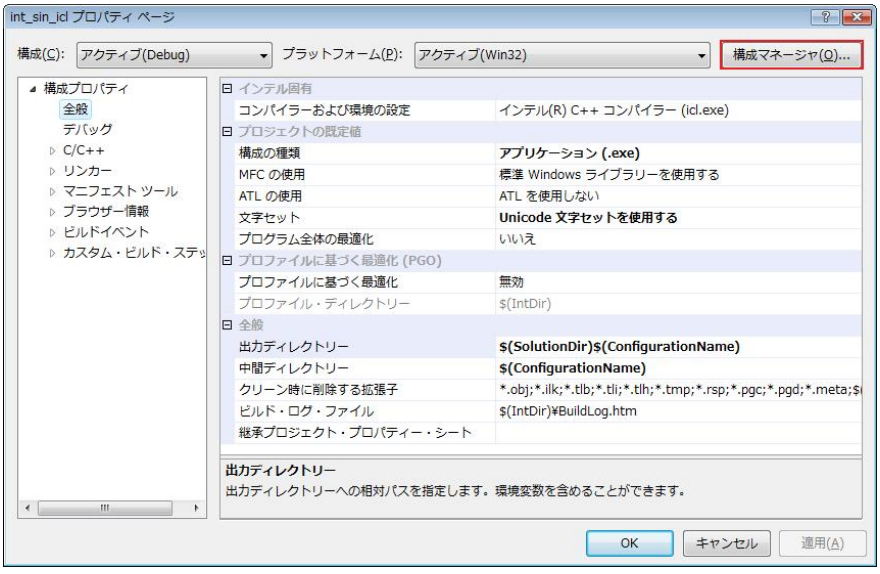


または、



表示される [プロパティ ページ] ダイアログ で、[構成マネージャ...] ボタンをクリックして、 [構成マネージャ] ダイアログを表示します。

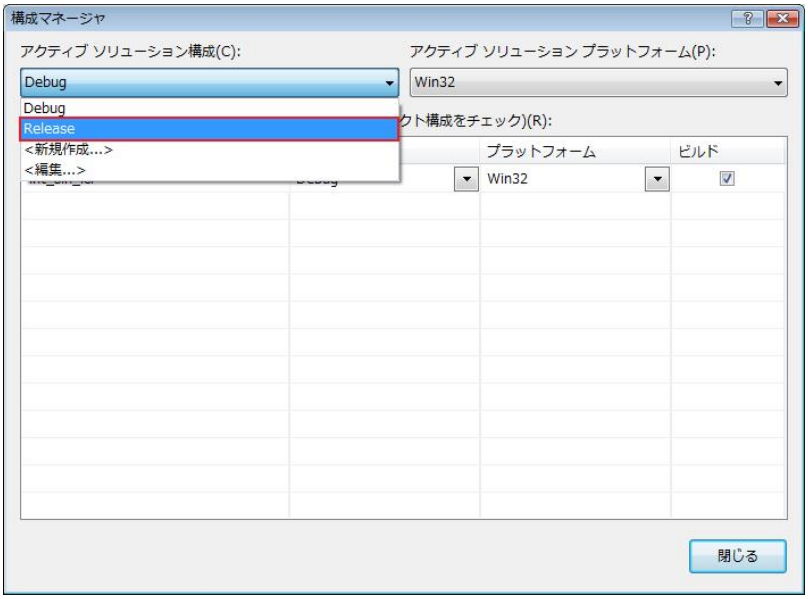
図：プロパティ ページ



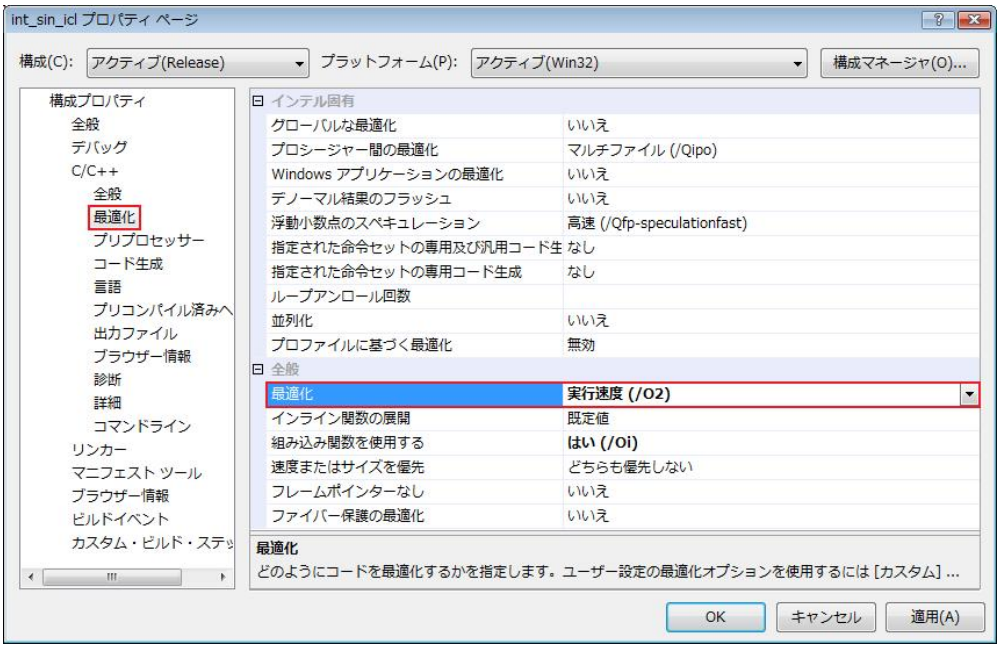
Note : この [プロパティ ページ] ダイアログは、コンパイルオプションの設定やインクルード・ファイル、ライブラリー・ファイルの指定など、プロジェクトのビルドに関するさまざまな設定を行う非常に重要なダイアログです。また、この [プロパティ ページ] は、Debug／Release それぞれのプロジェクト構成において個別のプロパティページを持っています。

2. [構成マネージャ] ダイアログで、[アクティブ ソリューション構成] を “Debug” から “Release” に変更し、 [閉じる] ボタンをクリックします。

図：構成マネージャ



3. [プロパティ ページ] ダイアログに戻り、[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。

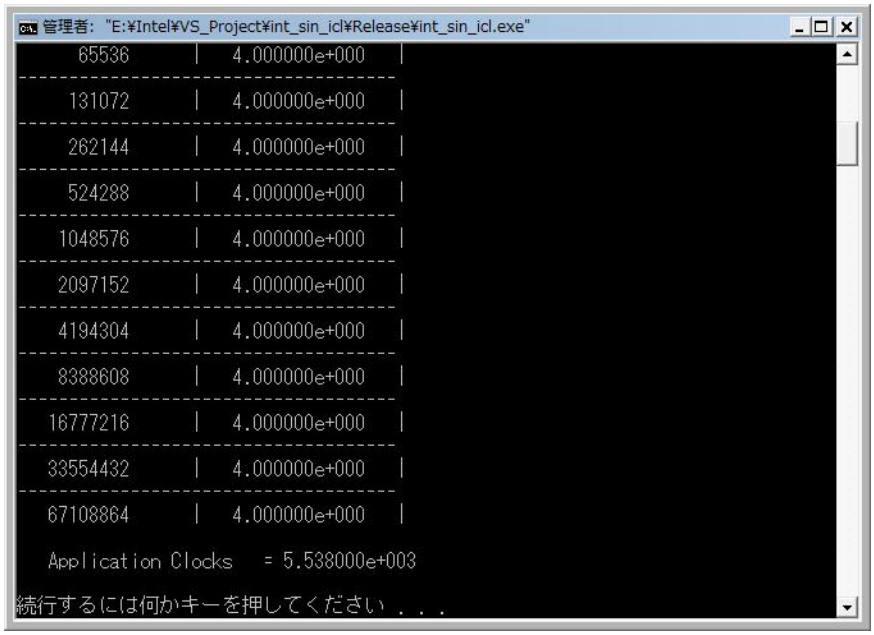


4. VS2008 のメニューから [ビルド] - [int_sin_icl のクリーン] を選択し、続いて [ビルド] - [int_sin_icl のビルド] を選択して、“Release” 構成で int_sin_icl プロジェクトをビルドします。

> Note : プロジェクト構成を Debug 構成から Release 構成に変更すると、最適化が有効になります。Release 構成において、デフォルトの最適化オプションは “実行速度”、つまり “/O2” が設定されています。なお、Debug 構成では “無効（/Od）” が設定されています。

３－４．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次の出力が表示されます。



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。

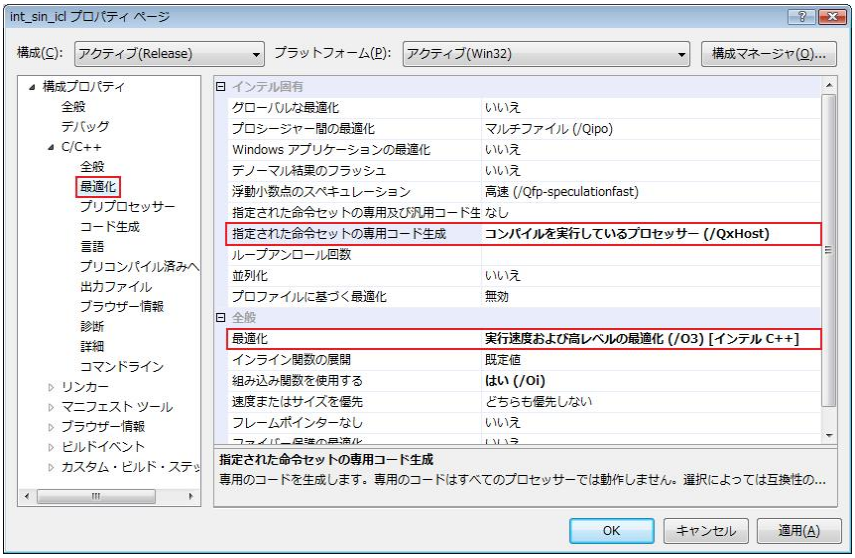
> Note : この例における (最適化なしから最適化ありにした場合の) 実行時間の大幅な向上はすべてのプログラムにあてはまるものではありませんが、通常は、適切な最適化を行うことで、インテル® プロセッサー上で実行するプログラムの実行時間を向上できます。

３－５．ビルド（最適化オプションあり）－ その２

コマンドライン同様、ここでもさらに最適化を行ってみましょう。使用する最適化オプションは、/O3 およびベクトル化オプションの /QxHost です。

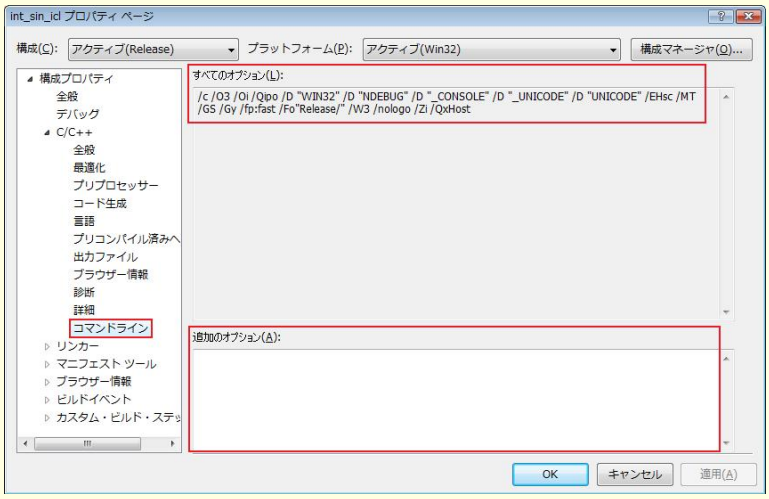
1. プロジェクトの [プロパティ ページ] ダイアログを開き、[構成プロパティ] - [C/C++] - [最適化] を選択して、下図のように [最適化] の値を “/O3”、 [指定された命令セットの専用コード生成] を ”/QxHost” に設定します。また、[構成プロパティ] - [C/C++] - [コード生成] を選択して、下図のように [ランタイム・ライブラリー] の値を “/MD” から “/MT” に変更してください。この変更により、リンクするランタイム・ライブラリーが動的ライブラリー（.dll）から静的ライブラリー（.lib）に変わります。本検証で使用しているサンプルコード（int_sin.c）はループ内でインテル数値演算ライブラリーを使用しており、このループをベクトル化するためにこの変更を行います。

図：プロパティ ページ



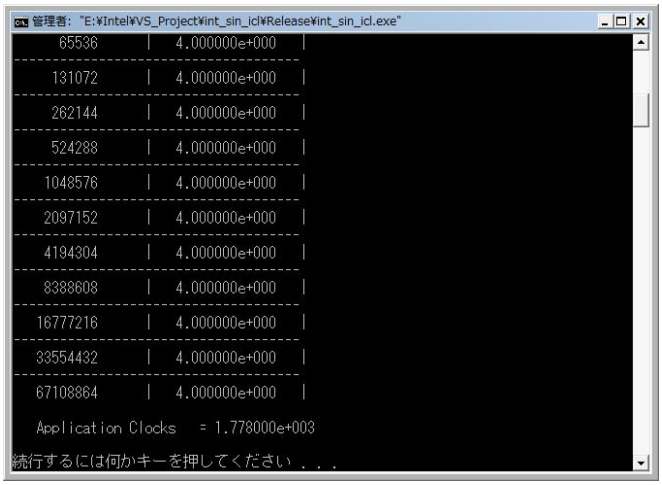
2. VS2008 のメニューから [ビルド] - [int_sin_icl のクリーン] を選択し、続いて [ビルド] - [int_sin_icl のビルド] を選択して、int_sin_icl プロジェクトをビルドします。

Note : 設定したすべてのコンパイルオプションを確認する場合は、[プロパティ ページ] ダイアログから、[構成プロパティ] - [C/C++] - [コマンドライン] を選択してください。今回の例では、"/O3" および "/QxHost" オプションが設定されていることが確認できます。なお、[追加のオプション] には手動でコンパイルオプションを追加することができます。ベクトル化の診断情報を表示する場合は、この [追加オプション] に /Qvec-report:3 などと直接指定してください。



3－6．実行/パフォーマンスの比較 ― その2

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. 最適化を行った場合の CPU 時間をメモして、結果を比較します。

Note : 最適化オプションの概要に関しては以下の『インテル® コンパイラー最適化クイック・リファレンス・ガイド』を参照してください。
http://jp.xlsoft.com/documents/intel/compiler/qr_guide_jp.pdf

４．既存ソースのコンパイル

ここでは、既存のソースファイルをコンパイルする際のいくつかのポイントをご紹介します。

４－１．複数のソースコードをコンパイルする場合

コマンドラインから、複数のソースコードをコンパイルする場合は、以下の例のようにすべてのソースコードをコマンドラインに指定してコンパイルしてください。以下の例でコンパイルが完了すると、main.exe という名前の実行形式ファイルが作成されます。

```
prompt> icl main.cpp sub1.cpp sub2.cpp
```

また、Visual Studio IDE から、複数のソースコードをコンパイルする場合は、下図のようにすべてのソースコードを [ソリューション エクスプローラ] の “ソースファイル” に追加してください。

４－２．特定のヘッダー／ライブラリー・ファイルを使用する場合

特定のヘッダーおよびライブラリー・ファイルをコマンドラインから使用する場合は、一般的に環境変数を使用します。設定する環境変数は、以下の３種類です。

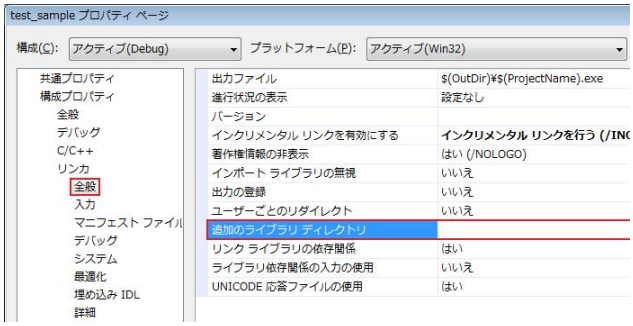
- INCLUDE：ヘッダーファイルの検索パス
- LIB：　　静的ライブラリー・ファイル（.lib）の検索パス
- Path：　　動的ライブラリー・ファイル（.dll）の検索パス

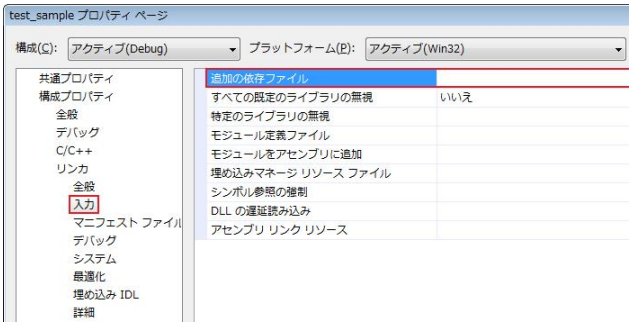
なお、これらの環境変数は、コマンドライン・ウィンドウ内で SET コマンドにて設定する一時的な環境変数、または Windows のシステム環境変数が利用できます。システム環境変数の設定方法は「５－３．環境変数について」を参照してください。

また、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] - [全般] を選択して、[追加の依存ファイル] に設定してください。

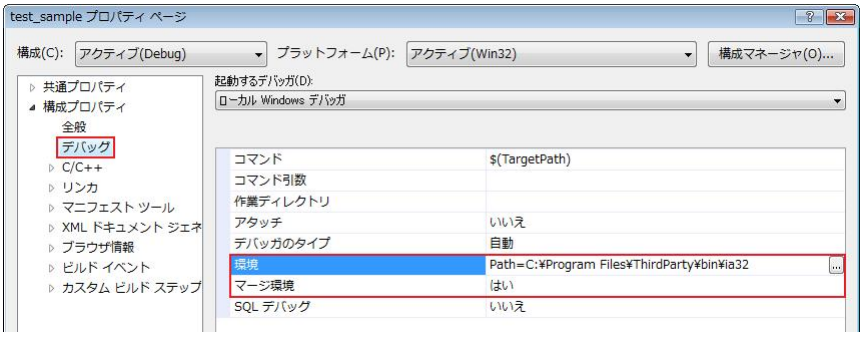




また、アプリケーションのデバッグや実行において、DLL ファイルが参照される場合があります。この DLL への検索パスを解決するには、システム環境変数の Path を使用するか、またはプロジェクトの [プロパティ ページ] ダイアログで [構成プロパティ] - [デバッグ] から [環境] を使用することが出来ます。たとえば以下のように記述して Path 環境変数への指定を行うことができます。

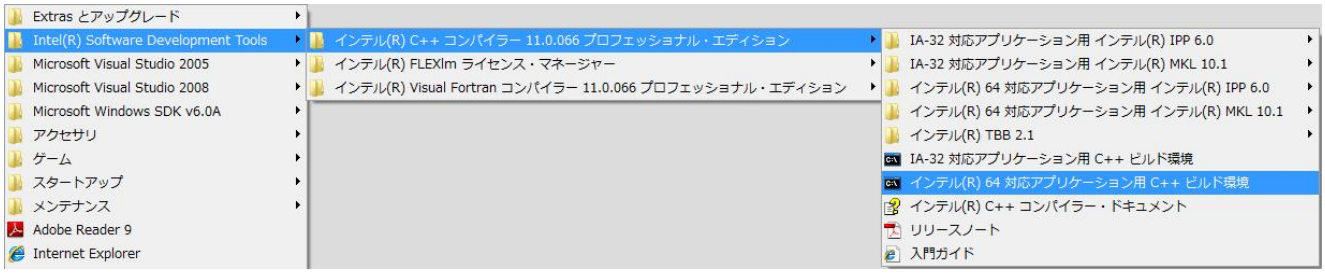
Path=C:¥Program Files¥ThirdParty¥bin¥ia32

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。



4－3．64 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(R) Software Development Tools] - [インテル(R) C++ コンパイラー 11.0.xxx プロフェッショナル・エディション] - [インテル(R) 64 対応アプリケーション用 C++ ビルド環境] を選択してください。このコマンドウィンドウでは、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（PATH、LIB、INCLUDE）の設定が完了しています。



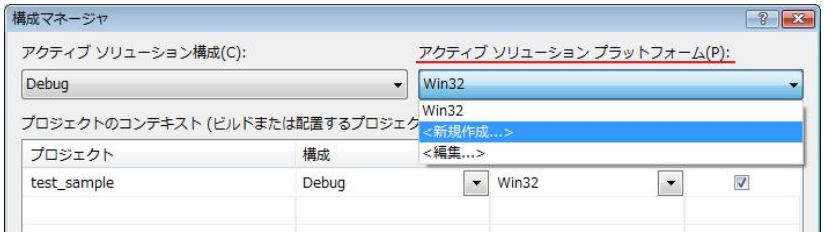
表示されるコマンドプロンプトに、以下のように icl コマンドを実行し、インテル® 64 対応アプリケーション用のインテル® コンパイラーが動作していることを確認してください。

```
prompt> icl
```

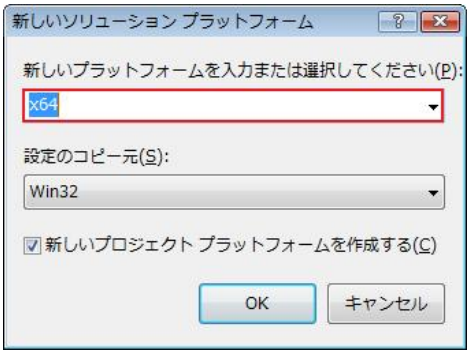
Visual Studio IDE からインテル® C++ コンパイラー を使用して 64 ビット対応アプリケーションを作成する場合は、プロジェクトを作成してから以下の操作を実行してください。

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 対応アプリケーション用インテル® 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。

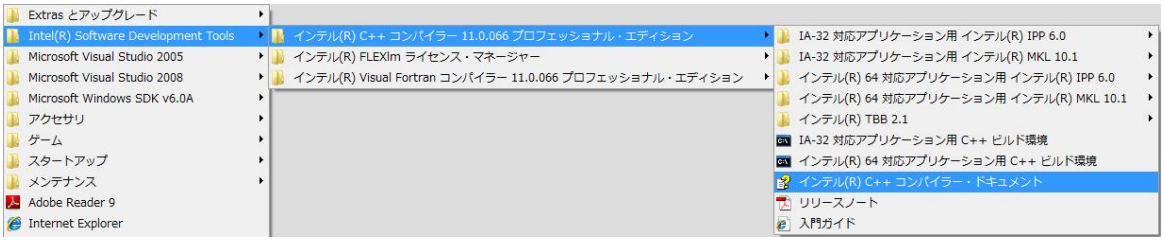
ご注意： 64 ビット対応アプリケーションを作成する場合は、インテル® 64 対応アプリケーション用インテル® コンパイラーのインストールはもちろん、リンク環境である Microsoft Visual Studio IDE 側でも 64 ビット用のライブラリー、その他必要なツールがインストールされている必要があります。Visual Studio 2005 および 2008 Standard Edition はデフォルト・インストールで X64 用ツールがインストールされますが、プロフェッショナル・エディション以上ではインストールする際にカスタム・インストールを選択し（またはアップデートで “機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

5． 追加情報

追加情報として、インテル® C++ コンパイラー関連事項をいくつか説明します。

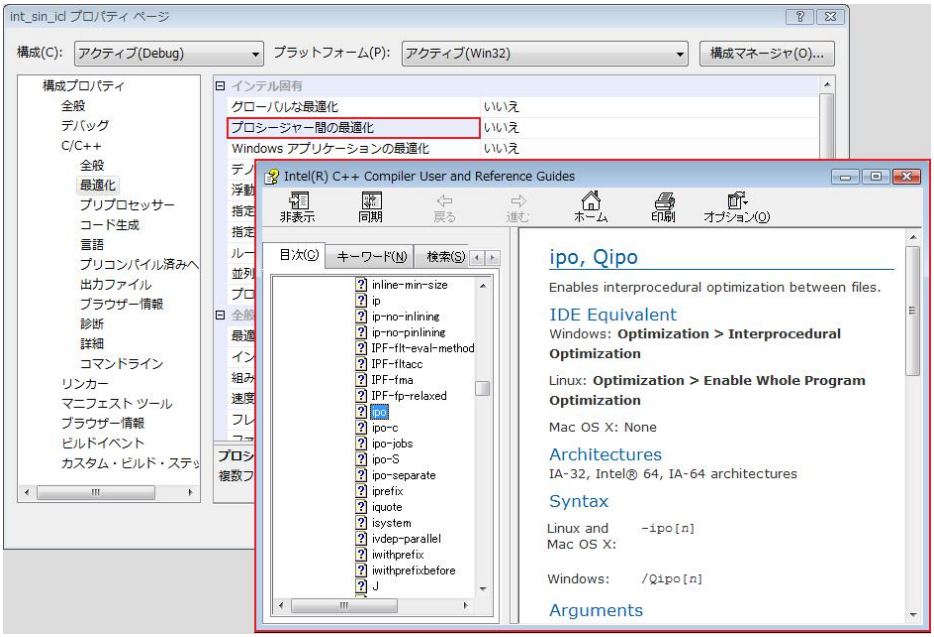
5－1． ドキュメントの参照方法

インストールされるドキュメントを参照するには、以下のように Windows [スタート] メニューからアクセスしてください。本製品に付属するライブラリーへのマニュアルも参照可能です。



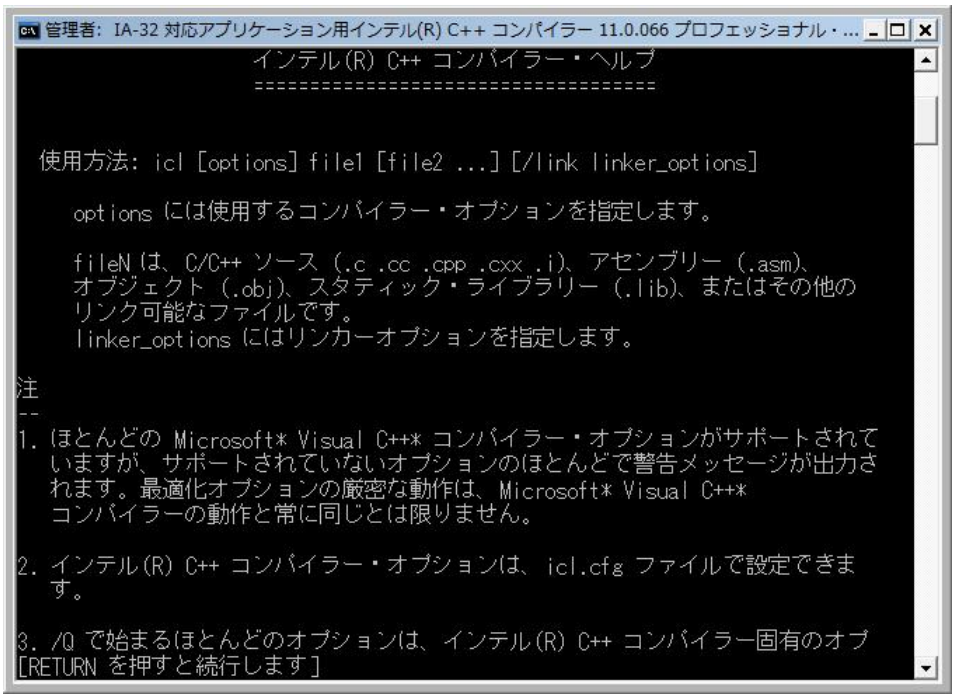
Visual Studio IDE からは、『インテル(R) C++ コンパイラー・ヘルプ』へのアクセスが可能です。

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。

prompt> icl /help



５－２．サンプルコード

本ドキュメントでは、サンプルコード（int_sin.c）を使用しましたが、インテル® C++ コンパイラーには、この他にも多くのサンプルが用意されています。以下にその概要をご紹介します。

- 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
- 組み込み関数を使用したサンプル
- ベクトル化オプション用サンプル

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。

<Install-Dir>¥Compiler¥11.0¥xxx¥cpp¥Samples¥C++¥ja_JP¥samples.htm

５－３．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。

< install-dir >¥Compiler¥11.0¥xxx¥cpp¥bin¥iclvars.bat 引数

このバッチファイルを実行するためには、引数が必要となります。

この引数の値は、開発環境に応じて以下のように指定する必要があります。

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

ia32_ia64 ・・・ IA-32 ホストシステムで IA-64 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

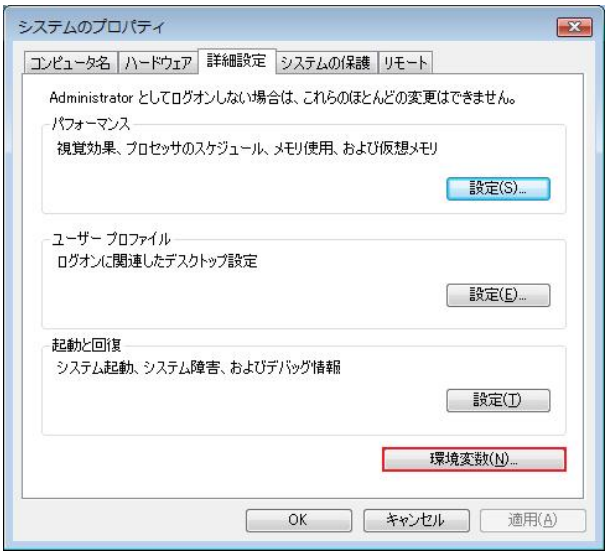
ia64 ・・・ IA-64 ホストシステムで IA-64 用アプリケーションをコンパイルする場合

インテル® コンパイラーでは、「インテル® コンパイラー専用コマンドウィンドウ」を起動した際に、適切な引数を使用して自動で環境変数設定バッチファイルが実行されます。たとえば、本ドキュメントの「２．コマンドラインからの操作方法」で起動した [IA-32 対応アプリケーション用 C++ ビルド環境] 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ています。

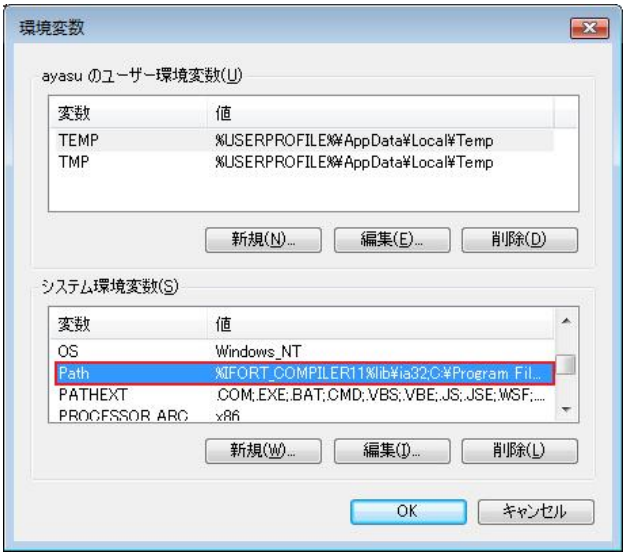
< install-dir >¥Compiler¥11.0¥xxx¥cpp¥bin¥iclvars.bat ia32

なお、Windows のデフォルトのコマンドプロンプトを使用する場合は、この環境変数設定バッチファイルを実行してビルド環境変数を手動で構築する必要があります。
また、コマンドラインにおけるランタイム環境変数については、「インテル® コンパイラー専用コマンドウィンドウ」を使用する場合は、すでに Path の設定が完了しているので特に追加設定する必要はありませんが、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、ビルド環境変数設定と同様に手動で環境変数設定バッチファイルを実行して Path を設定するか、またはシステム環境変数を使用して Path を指定する必要があります。
このシステム環境変数は、Windows Vista の場合、[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。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、続いて表示される [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。

図：システムのプロパティ



図：システム環境変数



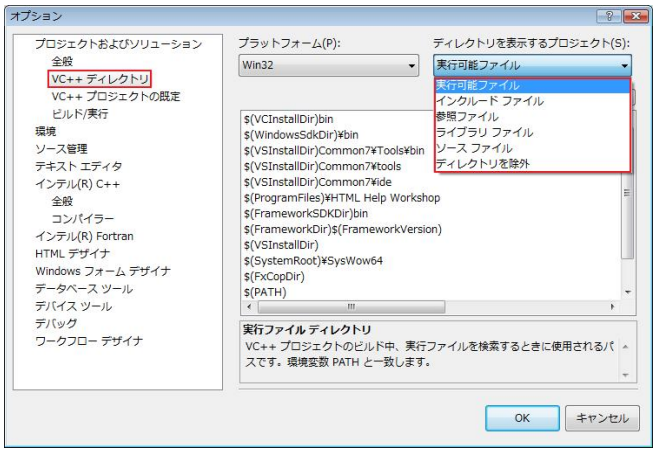
通常、このシステム環境変数の Path には、インテル® コンパイラーをインストールした時点で適切な値が設定されています。例えば、IA-32 システムにインストールした場合は以下のようにインテル® コンパイラーの提供する DLL ライブラリーへのパスが設定されています。

< install-dir >¥Compiler¥11.0¥xxx¥cpp¥lib¥ia32

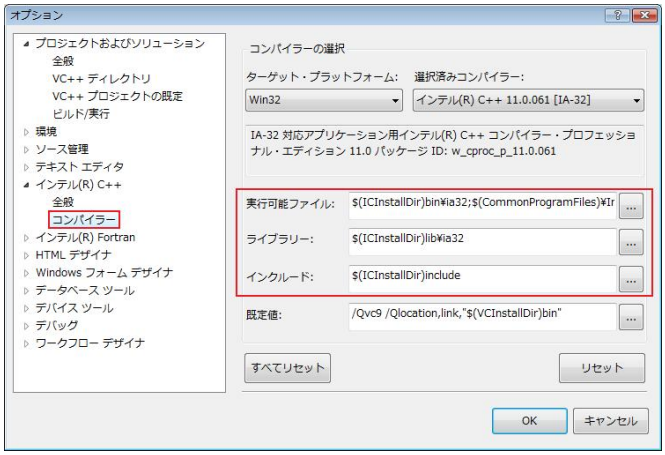
＜Visual Studio 開発環境における作業＞

Visual Studio 開発環境におけるビルド環境変数は、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。

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



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、特定のビルド環境変数を設定する場合は、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® コンパイラーでは、インテル® コンパイラー用のビルド環境変数のほかに、マイクロソフト・コンパイラー用のビルド環境変数も参照してコンパイルを行います。なお、これらビルド環境変数の変更は、Visual Studio 開発環境レベルの変更となるため、通常は、「４－２．特定のヘッダー／ライブラリー・ファイルを使用する場合」で説明したプロジェクト・レベルでの変更をお勧めします。

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します。Visual Studio のメニューから、[デバッグ] - [デバッグなしで開始] を選択した際にインテル® コンパイラーが提供するライブラリーの参照エラーが表示される場合は、システム環境変数の設定値を確認してください。インテル® コンパイラーのデフォルト・インストールを行った場合は、このシステム環境変数は適切な値が自動で設定されています。

５－４．マルチスレッドによる並列化について

インテル® コンパイラーはマルチコア、マルチプロセッサーに対応したマルチスレッドによる並列化をサポートしています。通常、シングルスレッド処理をマルチスレッド化することによりパフォーマンスの向上が見込まれます。インテル® コンパイラーのマルチスレッド化の対象箇所はプログラム内のループ処理であり、このループ処理をマルチスレッド化することにより複数のコアまたはプロセッサーに処理を分配して効率良く処理を行うことができます。この効果はプロセッサーのコア数、またはプロセッサー数に伴って大きくなります。

インテル® コンパイラーでは、マルチスレッド化の方法として以下の２つの手法をサポートしています。

- インテル® コンパイラーの自動並列化機能を使用する
- OpenMP 標準規格を使用する

① インテル® コンパイラーの自動並列化機能を使用する場合

インテル® コンパイラーには、自動でマルチスレッド化を行う自動並列化機能があります。この機能は、コンパイル時にプログラム内のループ処理に対してチェックが行われ、プログラムロジック的に安全にマルチスレッド変換が可能で、しかもマルチスレッド化による効率性が見込まれると診断されたループ処理に対してマルチスレッド変換が実行されます。この自動並列化機能を使用するには、コンパイル時に /Qparallel オプションを付加します。コマンドラインでは、以下のように指定することができます。

```
prompt> icl /Qparallel main.c
```

また、自動並列化機能には、見込まれる効率性のしきい値を調整することができます。このしきい値は 0 から 100 の範囲で設定が可能で、デフォルトでは 100 の値が設定されており、確実な効率性の向上が見込まれるループに対してのみマルチスレッド変換が適用されます。例えば、効率性の診断を無視して安全に変換可能なループに対してマルチスレッド変換を適用させたい場合は、次のようにしきい値に 0 を指定します。

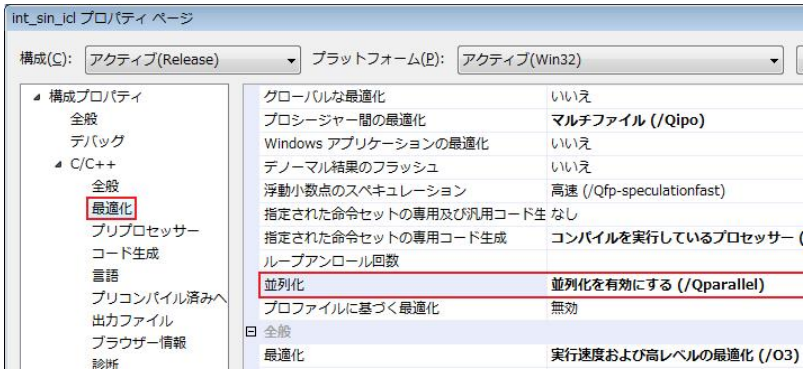
```
prompt> icl /Qparallel /Qpar-threshold:0 main.c
```

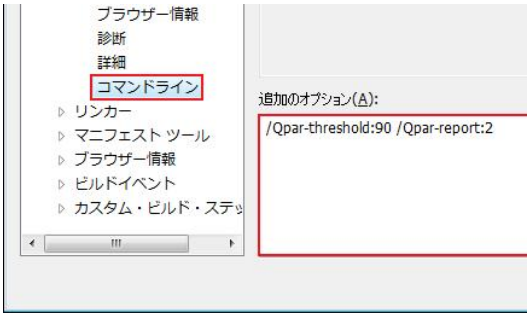
なお、自動並列化機能による診断結果を表示することも可能です。診断結果表示には /Qpar-report オプションに診断レベルを付けて、例えば以下のように指定します。

```
prompt> icl /Qparallel /Qpar-threshold:0 /Qpar-report2 main.c
```

Note：自動並列化機能は、しきい値を下げても安全にマルチスレッド化できないと診断されたループに対しては実行することができません。マルチスレッド化に関する詳細は、コンパイラー・ドキュメントを参照してください。

自動並列化機能を Visual Studio IDE から使用する場合は、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] - [C/C++] - [最適化] を選択して、下図のように [並列化] に “並列化を有効にする” を選択し、また [C/C++] – [コマンドライン] の [追加のオプション] に /Qpar-threshold:90 /Qpar-report:2 などとオプションを直接指定してください。





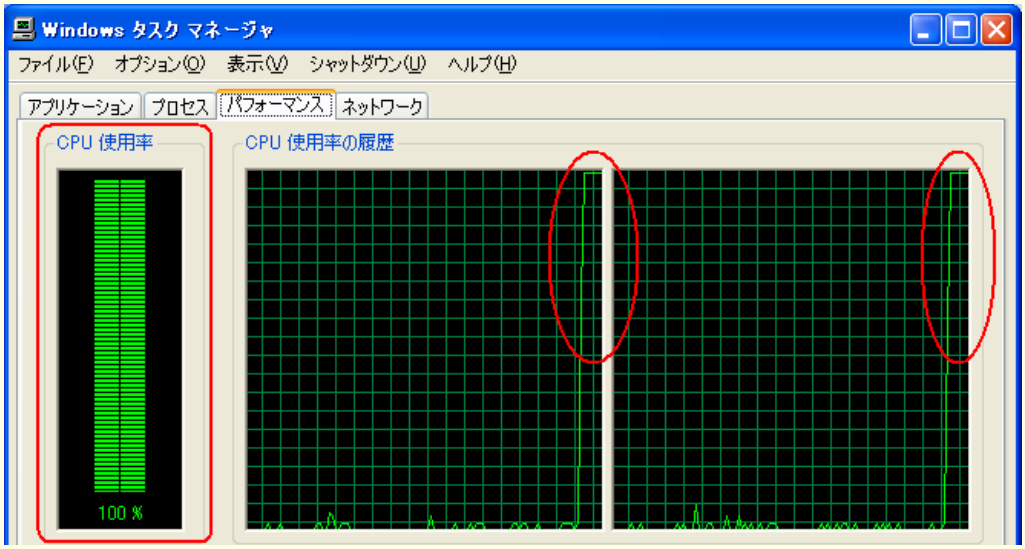
Note：サンプルコード int_sin.c を以下のように自動並列化することが出来ます。

prompt> icl /Qparallel /Qpar-threshold:0 int_sin.c

また、マルチスレッドにすることによりパフォーマンスが向上し、システム環境によっては実行時間が短すぎて十分に観察することが出来ない場合があります。このサンプルコードの場合、以下の行のループ回数を大きくすることにより計算量を増やすことが出来ます。

for (j=2;j<27;j++) → for (j=2;j<29;j++)

マルチスレッドによる CPU 使用状況は、「Windows タスク マネージャ」などで確認することができます。「パフォーマンス」タブで CPU の使用率が 100% になっていることを確認してください。



② OpenMP 標準規格を使用する場合

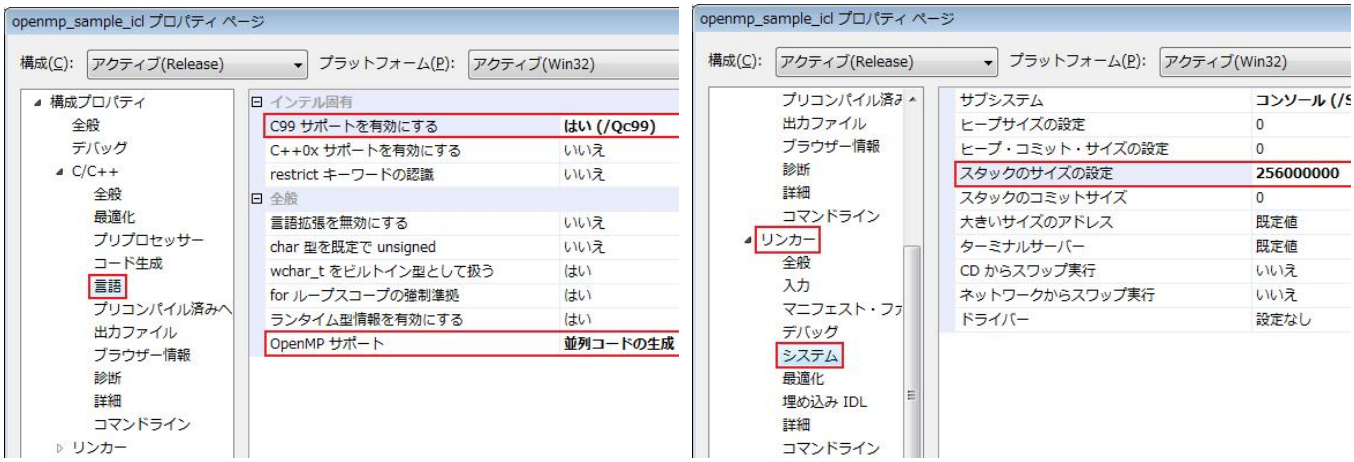
インテル® コンパイラーは、OpenMP によるマルチスレッドもサポートします。OpenMP では、マルチスレッド化したいループに対して、OpenMP 記述子（#pragma omp）を追加します。OpenMP のプログラム例は、インストールされる以下のサンプルを参考にしてください。

< install-dir >¥Compiler¥11.0¥xxx¥cpp¥Samples¥C++¥openmp_samples¥openmp_sample.c

本サンプルにおけるコマンドラインでのコンパイルは、以下のように実行します。

```
prompt> icl /Qstd=c99 /Qopenmp /F256000000 openmp_sample.c
```

Visual Studio IDE から使用する場合は、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] - [C/C++] - [言語] を選択して、下図のように [C99 サポートを有効にする] を “はい”、 [OpenMP サポート] を “並列コードの生成” に設定し、同様に [構成プロパティ] - [リンカー] - [システム] から [スタックサイズの設定] の値に “256000000” を指定してください。



サンプルコード int_sin.c も OpenMP を使用してマルチスレッド化することが出来ます。次のように、for ループ文の直前に #pragma omp 記述子を追加するだけでマルチスレッド化することができます。

```
#pragma omp parallel for private (x_i) reduction (+:sum)
    for (i=1;i<N;i++)
    {
        x_i = i * step;
        sum += INTEG_FUNC(x_i) * step;
    }
```

コマンドラインでのコンパイルは以下のように /Qopenmp を追加するだけです。

```
prompt> icl /Qopenmp int_sin.c
```

Note：OpenMP 標準規格に関する詳細は、以下のドキュメントをご参照ください。
http://jp.xlsoft.com/documents/intel/compiler/527J-001.pdf

6. 最後に

インテル® C++ コンパイラーの動作検証が完了したら、次はさまざまな最適化オプションを試してアプリケーションのパフォーマンス向上を図りましょう。インテル® C++ コンパイラーには、本ドキュメントで使用した最適化オプションのほかにも、多くのオプションが用意されています。最適化オプションの詳細はコンパイラー・ドキュメント、または以下の『インテル® コンパイラー最適化クイック・リファレンス・ガイド』を参照してください。
http://jp.xlsoft.com/documents/intel/compiler/qr_guide_jp.pdf

その他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

2008 年 12 月 1 日