「インテル® コンパイラー 入門」

～ 機能概要のご紹介 ～

エクセルソフト株式会社
安 晃生

Developers

Rock your code.

～ 内容 ～

1．インテル® コンパイラーの概要
2．インテル® コンパイラーの基本使用方法
3．最適化オプションについて
4．高速インテルライブラリーの利用
5．最後に

～ 内容 ～

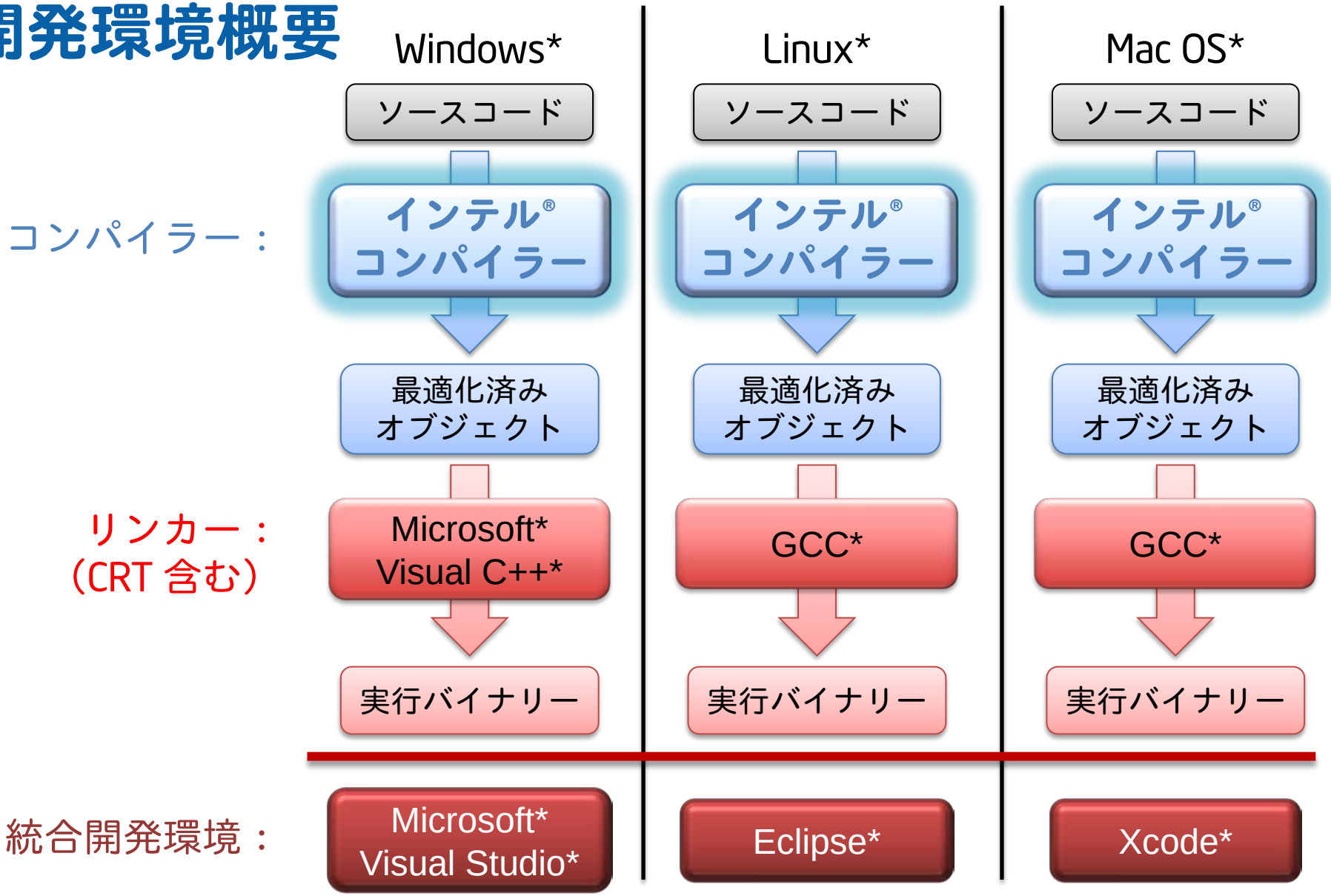
1．インテル® コンパイラーの概要
- インテル® コンパイラーとは・・・？
- インテル® コンパイラーの開発環境概要
- 製品紹介

2．インテル® コンパイラーの基本使用方法
3．最適化オプションについて
4．高速インテルライブラリーの利用
5．最後に

インテル® コンパイラーとは・・・？

- アプリケーションの**高速化**を実装するコンパイラー
- 優れた**最適化オプション**、**ライブラリー**を多数搭載
- **マルチコア**を有効利用するための**並列化モデル**を提供
- 最新の**インテル® プロセッサー**にいち早く対応
- Microsoft* Visual Studio* や GCC との**高い互換性**

開発環境概要



製品一覧

■ Windows* 版（日本語版あり）

- インテル® Composer XE 2011 Windows* 版 ← バンドル版
- インテル® C++ Composer XE 2011 Windows* 版
- インテル® Visual Fortran Composer XE 2011 Windows* 版

■ Linux* 版（日本語版あり）

- インテル® Composer XE 2011 Linux* 版 ← バンドル版
- インテル® C++ Composer XE 2011 Linux* 版
- インテル® Fortran Composer XE 2011 Linux* 版

■ Mac OS* 版

- インテル® C++ Composer XE 2011 Mac OS* 版
- インテル® Fortran Composer XE 2011 Mac OS* 版

製品コンポーネント一覧（Windows*）

インテル® Composer XE 2011 Windows* 版

インテル® C++ Composer XE 2011 Windows* 版

- インテル® C++ コンパイラー XE 12.0（IA-32 / インテル® 64）
- インテル® Parallel Debugger Extension
- インテル® スレッディング・ビルディング・ブロック (インテル® TBB) 3.0
- インテル® マス・カーネル・ライブラリー (インテル® MKL) 10.3
- インテル® インテグレーテッド・パフォーマンス・プリミティブ (インテル® IPP) 7.0

インテル® Visual Fortran Composer XE 2011 Windows* 版

- インテル® Fortran コンパイラー XE 12.0（IA-32 /インテル® 64）
- インテル® Parallel Debugger Extension
- インテル® マス・カーネル・ライブラリー (インテル® MKL) 10.3
- Microsoft* Visual Studio* 2008 Shell

製品コンポーネント一覧（Linux*）

インテル® Composer XE 2011 Linux* 版

インテル® C++ Composer XE 2011 Linux* 版

- インテル® C++ コンパイラー XE 12.0（IA-32 /インテル® 64）
- インテル® デバッガー 12.0
- インテル® スレッディング・ビルディング・ブロック (インテル® TBB) 3.0
- インテル® マス・カーネル・ライブラリー (インテル® MKL) 10.3
- インテル® インテグレーテッド・パフォーマンス・プリミティブ (インテル® IPP) 7.0

インテル® Fortran Composer XE 2011 Linux* 版

- インテル® Fortran コンパイラー XE 12.0（IA-32 /インテル® 64）
- インテル® デバッガー 12.0
- インテル® マス・カーネル・ライブラリー (インテル® MKL) 10.3

製品コンポーネント一覧（Mac OS*）

インテル® C++ Composer XE 2011 Mac OS* 版

- インテル® C++ コンパイラー XE 12.0（IA-32 /インテル® 64）
- インテル® デバッガー 12.0
- インテル® スレッディング・ビルディング・ブロック (インテル® TBB) 3.0
- インテル® マス・カーネル・ライブラリー (インテル® MKL) 10.3
- インテル® インテグレーテッド・パフォーマンス・プリミティブ (インテル® IPP) 7.0

インテル® Fortran Composer XE 2011 Mac OS* 版

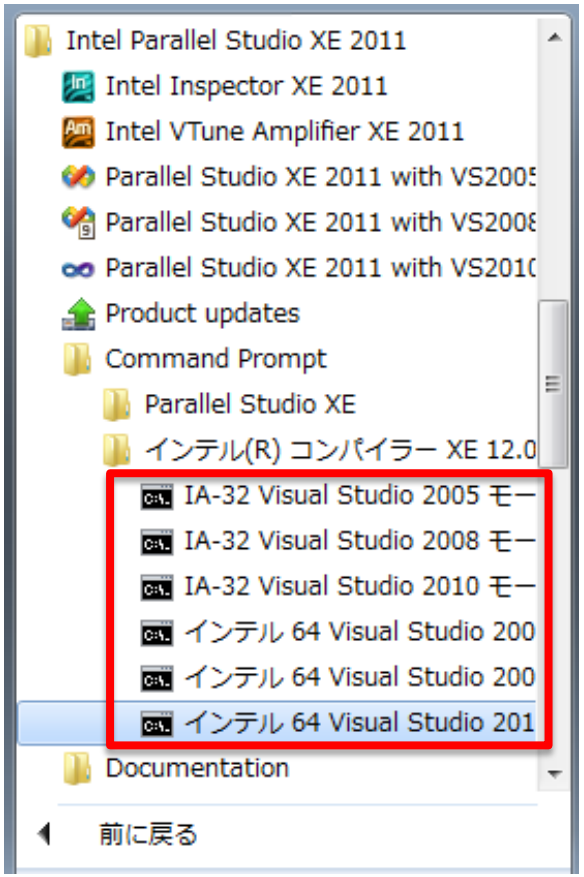
- インテル® Fortran コンパイラー XE 12.0（IA-32 /インテル® 64）
- インテル® デバッガー 12.0
- インテル® マス・カーネル・ライブラリー (インテル® MKL) 10.3

～ 内容 ～

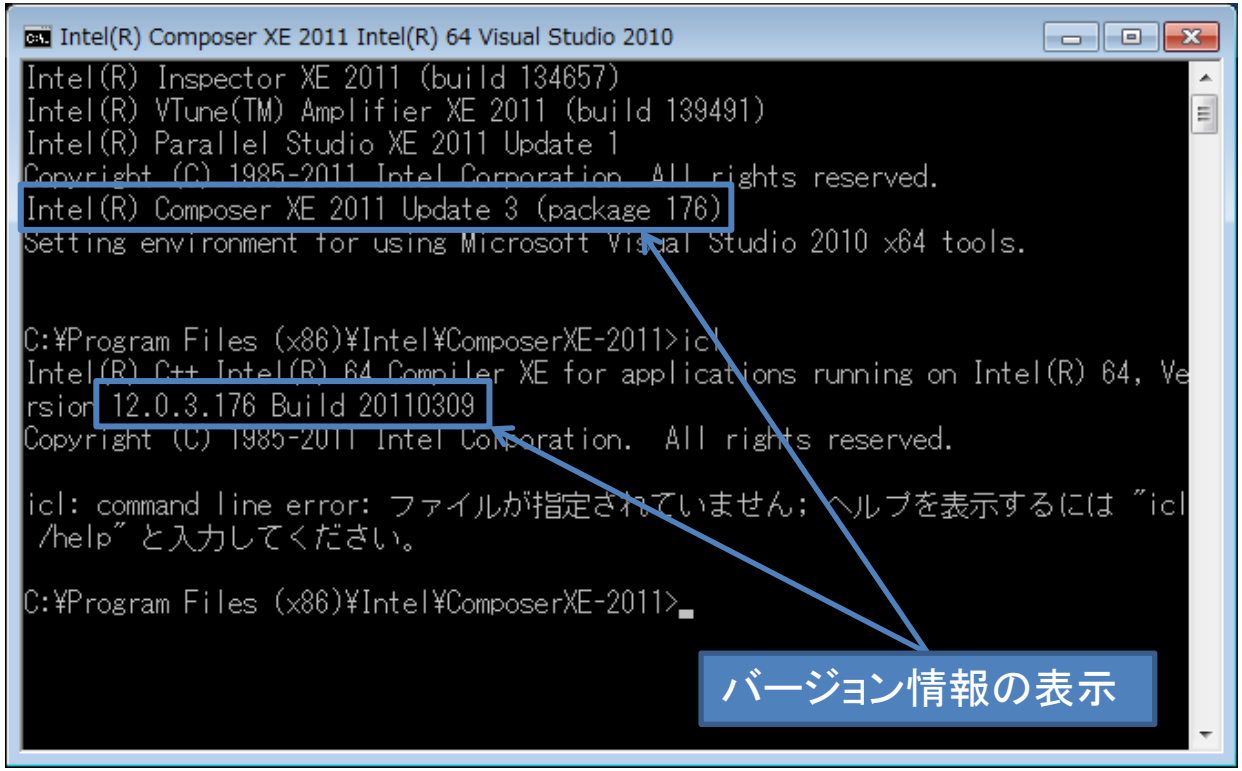
1．インテル® コンパイラーの概要
2．インテル® コンパイラーの基本使用方法
　- コマンドラインからのビルド
　- Visual Studio* からのビルド
3．最適化オプションについて
4．高速インテルライブラリーの利用
5．最後に

コマンドラインからのビルド（Windows*）

- Windows* スタートメニューに用意されたコマンドプロンプトを利用する。



- このコマンドプロンプトでは、ビルドに必要な環境変数 (Path / Include / Lib) が設定されている。



インテル® コンパイラー コマンド

インテル® コンパイラー（コンパイラー・ドライバー）の使用方法

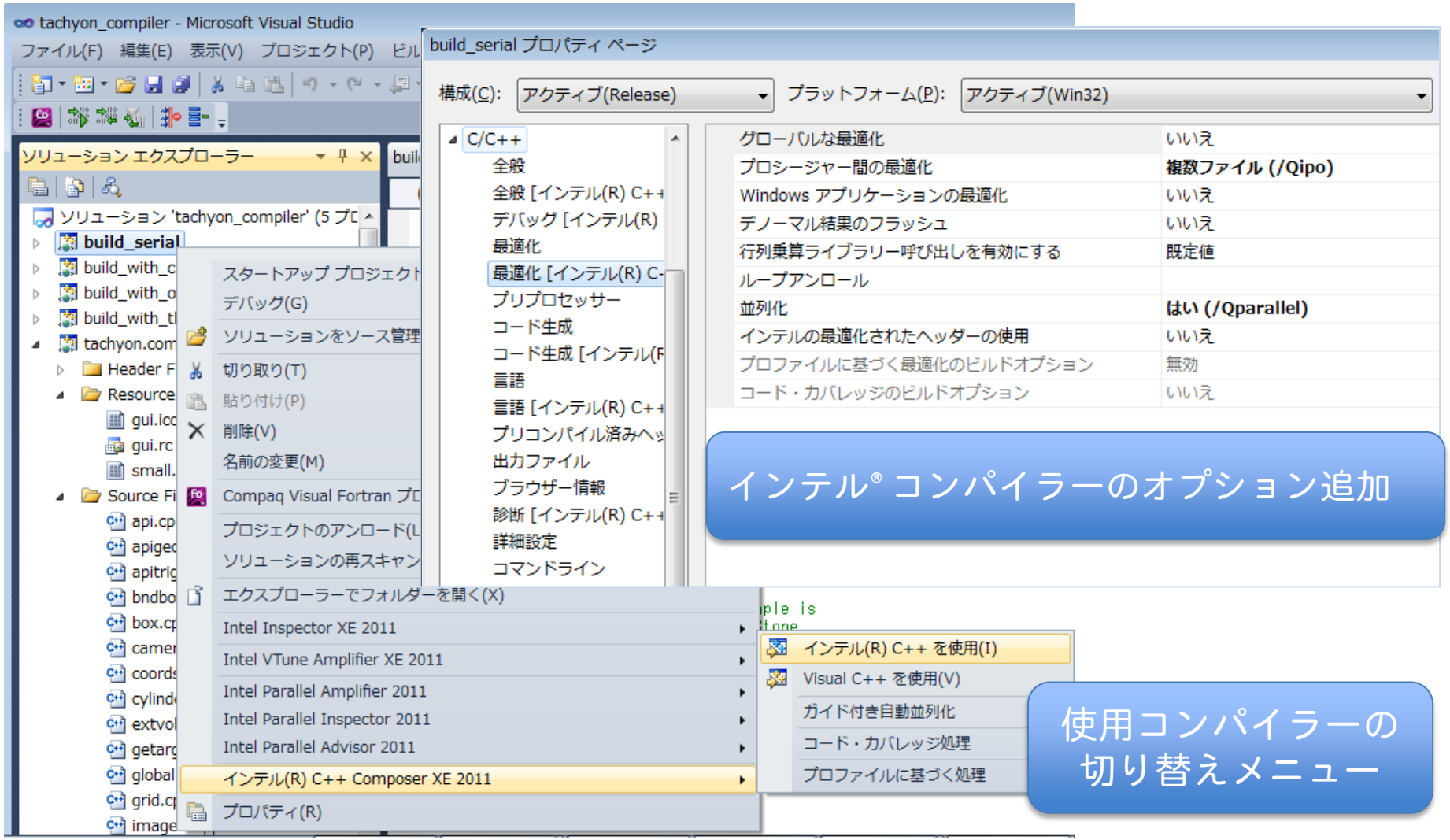
(Windows*)

```
C/C++ コンパイラー：
 > icl [options] file1 [file2...] [/link linker_options]

Fortran コンパイラー：
 > ifort [options] input_file(s)
```

(Linux*/MacOS*)

```
C/C++ コンパイラー：
 $ icc|icpc [options] file1 [file2...]

Fortran コンパイラー：
 $ ifort [options] input_file(s)
```

Visual Studio* からのビルド

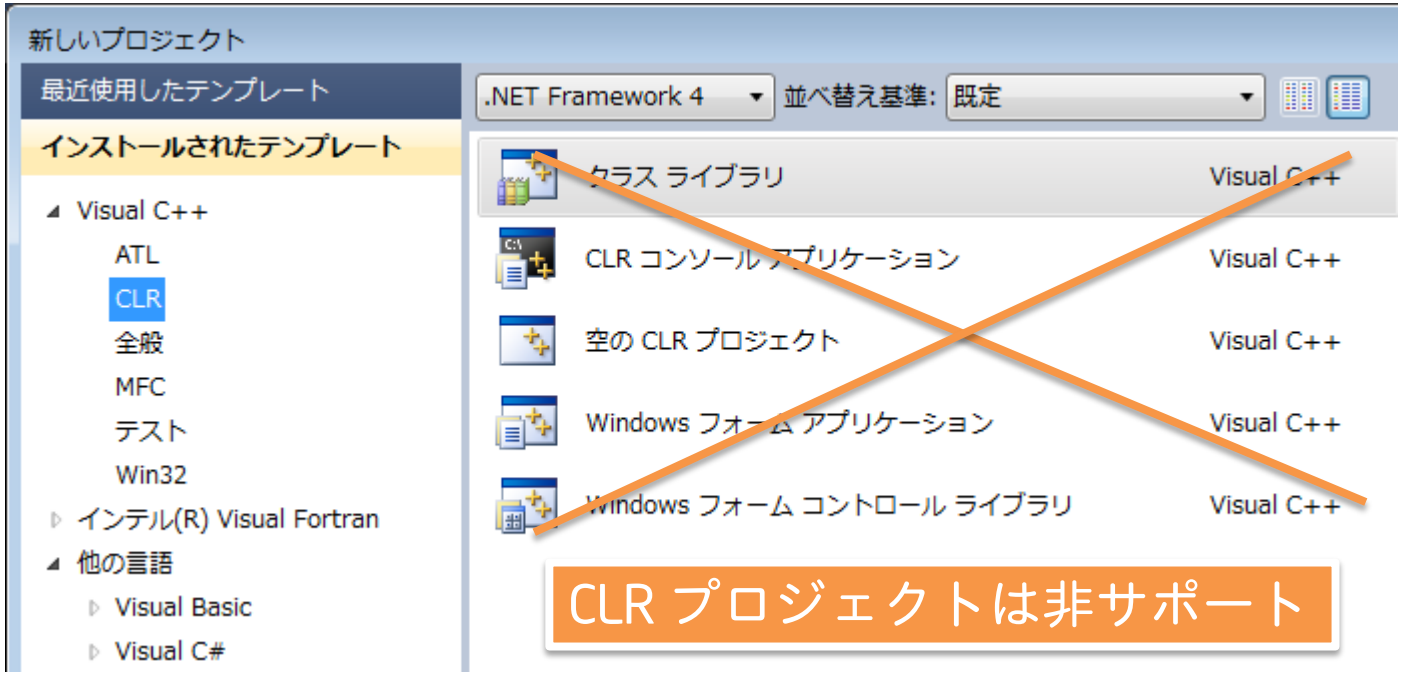


Developers

Rock your code.

使用可能なプロジェクトタイプ

インテル® C++ コンパイラー Windows* 版は、C# などの Microsoft* .NET アプリケーションのようなマネージドコードはサポートしません。
アンマネージドコード（ネイティブコード）のみのサポートとなります。



～ 内容 ～

1．インテル® コンパイラーの概要
2．インテル® コンパイラーの基本使用方法
3．最適化オプションについて
- ハイレベルな最適化（HLO）
- プロシージャ• 間の最適化（IPO）
- プロファイルに基づく最適化（PGO）
- 自動ベクトル化
- 自動並列化
- 浮動小数点演算モデル
4．高速インテルライブラリーの利用
5．最後に

代表的な最適化オプション一覧

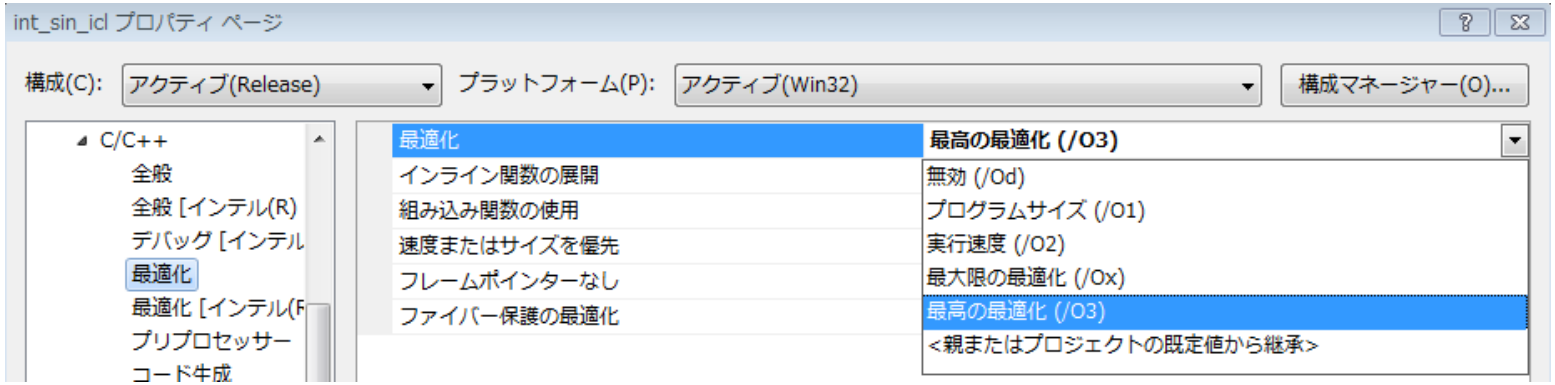
	Windows	Linux / Mac OS
ハイレベルな最適化（HLO）	/O3	-O3
プロシージャ• 間の最適化（IPO）	/Qipo	-ipo
プロファイルに基づく最適化（PGO）	/Qprof-gen /Qprof-use	-prof-gen -prof-use
自動ベクトル化	/arch:*code* /Qx*code* /Qax*code*	-m*code* -x*code* -ax*code*
自動並列化	/Qparallel	-parallel
ガイド付き自動並列化（GAP）	/Qguide[*n*]	-guide[*n*]
関数 / ループ・プロファイラー	/Qprofile-functions /Qprofile-loops:<arg>	-profile-functions -profile-loops=<arg>
スタティック・セキュリティー解析（SSA）	/Qdiag-enable:sc[*n*]	-diag-enable sc[*n*]
浮動小数点演算	/fp:*keyword*	-fp-model *keyword*

ハイレベルな最適化（HLO）

インテル® コンパイラーの
デフォルトは /O2（-O2）

オプション：/O3（Linux/MacOS：-O3）

- 自動ベクトル化オプションと併用することにより、/O2 よりさらに強力なデータ依存分析を行いベクトル化の可能性を高める。
- 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特に有効。
- 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るので注意が必要

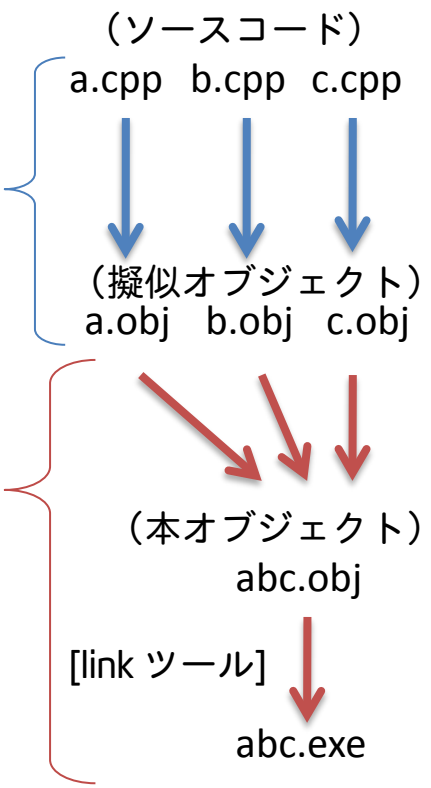


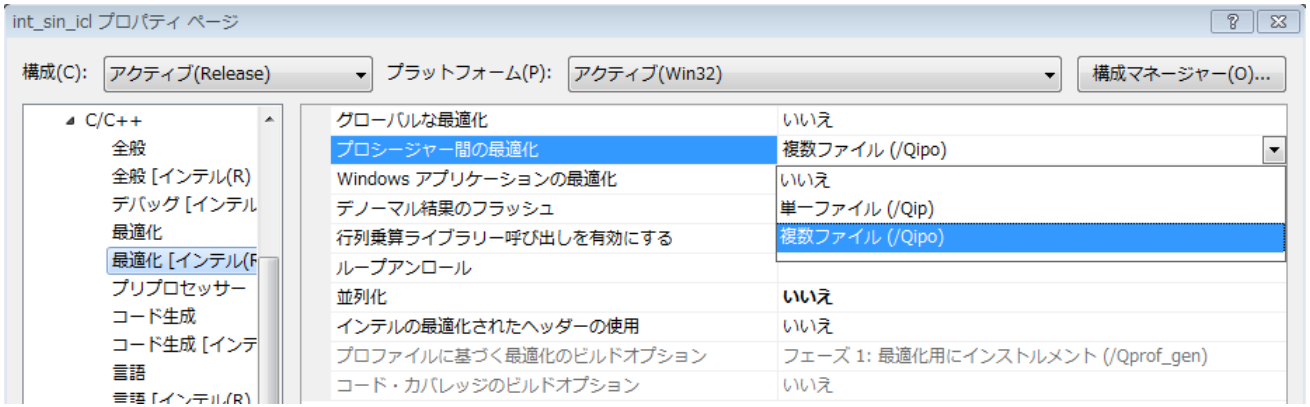
プロシージャ・間の最適化（IPO）

オプション：/Qipo（Linux/MacOS：-ipo）

- プログラム全体の構成を把握しソースファイル間における参照関係を考慮して効果的な最適化を実施する
- 「関数のインライン展開」「エイリアス解析」「定数伝播」「不要な処理の削除」「スタックフレームのアライメント」など最適化が可能
- 自動ベクトル化の可能性も高まる

IPO 動作概要(Win)





プロファイルに基づく最適化（PGO）

■ 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

[作業手順]　（※Linux/MacOS：-prof-gen、-prof-use）

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(Win)を指定してプログラムをビルド）

↓

フェーズ２：アプリケーションの実行
（動的プロファイル情報ファイル（.dyn）の生成）

↓

フェーズ３：フィードバック・コンパイル
（/Qprof-use オプション(Win)を使用して、動的プロファイル情報ファイル（.dyn）を反映したビルドを実施）

- PGO 機能により、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が得られる。
- フェーズ３にて、IPO 機能や他の最適化オプションも同時に指定するとさらに効果的

自動ベクトル化

ベクトル化とは・・・？

スカラー演算から SIMD(Single Instruction Multiple Data) 演算に変換して処理効率の良いコードを実装する技術です。

例）

```
for ( i=0; i<=MAX; i++ )
    c[i] = a[i] + b[i];
```

インテル® コンパイラーの自動ベクトル化機能を適用

[スカラー演算]
1 命令で 1 結果

[SIMD 演算]
1命令で複数結果（SSE/AVX 命令とXMM/YMM レジスター使用）

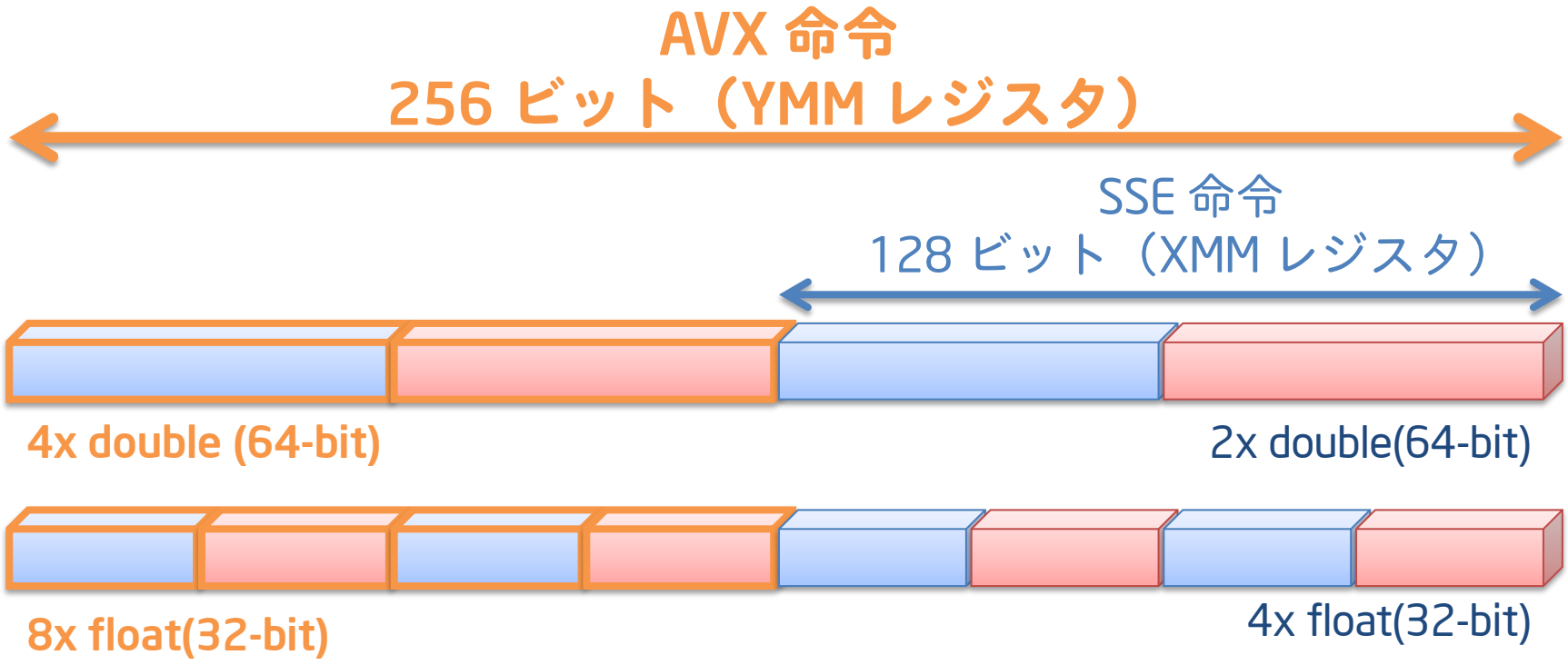
SSE 命令バージョン

1999	2000	2004	2006	2007	2008
SSE	SSE2	SSE3	SSSE3	SSE4.1	SSE4.2
70 instr Single-Precision Vectors Streaming operations	144 instr Double-precision Vectors 8/16/32 64/128-bit vector integer	13 instr Complex Data	32 instr Decode	47 instr Video Graphics building blocks Advanced vector instr	8 instr String/XML processing POP-Count CRC

- Intel® AES New Instruction – Intel® AES-NI（2009）
- Intel® Advanced Vector Extensions – Intel® AVX（2010/11）

自動ベクトル化（SSE vs. AVX）

AVX 命令は、第 2 世代 Core プロセッサー(Sandy Bridge)から搭載



自動ベクトル化（オプション一覧）

-O2 以上の最適化
オプションが必須

■ インテルおよびインテル互換プロセッサー用オプション

Windows　：/arch:*code*　例）/arch:IA32、/arch:SSE2、/arch:SSE4.1
Linux / MacOS　：-m*code*　例）-mia32、-msse2、-msse4.1
code：IA32、SSE、SSE2、SSE3、SSSE3、SSE4.1、SSE4.2、AVX

■ インテル® プロセッサー専用オプション

Windows　：/Qx*code*　例）/QxSSSE3、/QxSSE4.2、/QxAVX、/QxHost
Linux / MacOS　：-x*code*　例）-xSSSE3、-xSSE4.2、-xAVX、-xHost
code：SSE2、SSE3、SSSE3、SSE3_ATOM、SSE4.1、SSE4.2、AVX、Host

※CPU ランタイムチェックが実行され、サポート対象外 CPU 上では動作しない

■ インテル® プロセッサー専用コード＋汎用コードの生成オプション

Windows　：/Qax*code*　例）/QaxSSSE3、/QaxSSE4.2、/QaxAVX
Linux / MacOS　：-ax*code*　例）-axSSSE3、-axSSE4.2、-axAVX
code：SSE2、SSE3、SSSE3、SSE3_ATOM、SSE4.1、SSE4.2、AVX

※専用コードの生成が有益であると判断された場合、自動ディスパッチャーが追加され
　実行コードがランタイムで決定される
※デフォルト汎用コードは /arch:SSE2(Windows)、-msse2(Linux/MacOS)
※汎用コードは /arch または /Qx (Windows)、- m または -x (Linux/MacOS) で変更可能

自動ベクトル化（コンパイル例）

デフォルトオプション：/arch:SSE2（Windows）　-msse2（Linux、MacOS）
※O2 以上の最適化オプションが使用された場合は暗黙的にオンとなる。

```
(Windows)     > icl main.cpp
(Linux/MacOS) $ icc main.cpp
```

/O2(Windows)、-O2(Linux/MacOS) オプションがデフォルトなので、
/arch:SSE2(Windows)、-msse2(Linux/MacOS) のコードが生成される。

```
(Windows)     > icl  /QxSSE4.2  main.cpp
(Linux/MacOS) $ icc  -xSSE4.2  main.cpp
```

SSE4.2 以上の命令が搭載されたインテル® プロセッサーでのみ動作可能な
コードが生成される。

```
(Windows)     > icl  /QaxSSE4.2  main.cpp
(Linux/MacOS) $ icc  -axSSE4.2  main.cpp
```

コンパイラーが SSE4.2 コードの生成を有益と見なした場合、SSE4.2 コード
と汎用コードとして /arch:SSE2(Windows)、-msse2(Linux/MacOS) コードが生
成される。そうでない場合は、汎用コードのみを生成。

自動ベクトル化（生成コード例）

```
static double A[1000], B[1000],
               C[1000];
void add() {
  int i;
  for (i=0; i<1000; i++)
    if (A[i]>0)
      A[i] += B[i];
    else
      A[i] += C[i];
}
```

/QxSSE2

```
.B1.2::
  movaps     xmm2, A[rax*8]
  pxor       xmm0, xmm0
  cmpltpd    xmm0, xmm2
  movaps     xmm1, B[rax*8]
  andps      xmm1, xmm0
  andnps     xmm0, C[rax*8]
  orps       xmm1, xmm0
  addpd      xmm2, xmm1
  movaps     A[rax*8], xmm2
  add        rax, 2
  cmp        rax, 1000
  jb         .B1.2
```

SSE2

/QxAVX

```
.B1.2::
  vmovupd    ymm3, A[rax*8]
  vmovupd    ymm1, C[rax*8]
  vcmpgtpd   ymm2, ymm3, ymm0
  vblendvpd  ymm4, ymm1,B[rax*8], ymm2
  vaddpd     ymm5, ymm3, ymm4
  vmovupd    A[rax*8], ymm5
  add        rax, 4
  cmp        rax, 1000
  jb         .B1.2
```

AVX

/QxSSE4.1

```
.B1.2::
  movaps     xmm2, A[rax*8]
  xorps      xmm0, xmm0
  cmpltpd    xmm0, xmm2
  movaps     xmm1, C[rax*8]
  blendvpd   xmm1, B[rax*8], xmm0
  addpd      xmm2, xmm1
  movaps     A[rax*8], xmm2
  add        rax, 2
  cmp        rax, 1000
  jb         .B1.2
```

SSE4.1

自動ベクトル化（結果レポート）

オプション：

/Qvec-report[n]（Linux/MacOS：-vec-report[n]）n：0～5

Intel(R) Composer XE 2011 Intel(R) 64 Visual Studio 2010

```
C:¥temp>icl /Qvec-report2 /c /nologo vec_sample.cpp
vec_sample.cpp
C:¥temp¥vec_sample.cpp(39): (col. 3) remark: ループがベクトル化されました。.
C:¥temp¥vec_sample.cpp(39): (col. 3) remark: ループはスキップされました: 複数の
バージョンがあります。.
C:¥temp¥vec_sample.cpp(47): (col. 3) remark: ループはベクトル化されませんでした:
 ベクトル化は可能ですが非効率です。.
C:¥temp¥vec_sample.cpp(47): (col. 3) remark: ループはスキップされました: 複数の
バージョンがあります。.
C:¥temp¥vec_sample.cpp(54): (col. 3) remark: ループはベクトル化されませんでした:
 内部ループではありません。.
C:¥temp¥vec_sample.cpp(54): (col. 3) remark: ループはベクトル化されませんでした:
 ベクトル依存関係が存在しています。.

C:¥temp>
```

自動並列化

- コンパイラーがループ文に対して、マルチスレッドへの変換を試みます。
- マルチスレッドを安全に実装でき、また効率的だと判断された場合に適用される。
- 実装は、OpenMP* の並列化モデルが使用される。

オプション： /Qparallel　（Linux/MacOS：-parallel）

レポートオプション：
/Qpar-report[n]（Linux/MacOS：-par-report[n]）n：0～3
※ n は表示レベル、3 が最も詳細なレポートを表示。

しきい値設定オプション：（効率性の診断レベル）
/Qpar-threshold[n]（Linux/MacOS：-par-threshold[n]）n：0～100
※ n の数字を小さくすれば並列化の可能性が高くなるが、逆にパフォーマンスが落ちる可能性もある。
※ デフォルトのしきい値は最高レベルの100に設定され高い効率性を要求される。

自動並列化（実験）

- 円周率（π）の計算サンプルに自動並列化を適用

```
...
for (i=0; i<INTERVALS; ++i)
{
   x = Step * ((double)i-0.5);
   pi += 4.0 / (1.0 + x*x);
}
pi = Step * pi;
...
```

Intel(R) Composer XE 2011 Intel(R) 64 Visual Studio 2010

```
C:¥temp>icl pi.cpp /nologo
pi.cpp

C:¥temp>pi.exe
Time  =  5.170000[s] (PI: 3.141593)
```

← シングルスレッドの結果

```
C:¥temp>icl pi.cpp /nologo /Qparallel /Qpar-report2
pi.cpp
   procedure: main
C:¥temp¥pi.cpp(19): (col. 2) remark: ループが自動並列化されました。.

C:¥temp>pi.exe
Time  =  1.652000[s] (PI: 3.141593)
```

← マルチスレッドの結果

```
C:¥temp>
```

インテル® Core™ i7　4 コア、64 ビット 1.6GHz

浮動小数点演算モデル

インテル® コンパイラーでは、浮動小数点演算の最適化を以下の観点から実施することができます。

- **精度**
  正しい解にどれだけ近い値を算出するか
- **再現性**
  以下のような条件下で一貫した値を生成するか
  - 異なる実行
  - 異なるビルドオプション
  - 異なるコンパイラー
  - 異なるプラットフォーム
  - 異なるアーキテクチャー
- **パフォーマンス**
  計算・処理速度の向上

これらの要求は、互いにトレードオフ（二律背反の関係）です。

浮動小数点演算モデル（オプション）

/fp:*キーワード*（Linux/MacOS：-fp-model *キーワード*）
※キーワードは以下の通り

fast[=1|2]：	強力な最適化を適用（解の精度に影響を与える可能性あり） fast（fast=1）がデフォルトオプション 非正規化数の生成なし（※メイン関数内で Flush-To-Zero (FTZ) の有効化） fast=2 ではさらに強力な最適化を実施
precise：	解の精度に影響を与えない最適化のみを適用
source：	計算の中間結果をソースで定義された型の精度で丸め、解の精度に影響を与えない最適化のみを適用
double：	計算の中間結果を 53 ビット（double）精度で丸め、 解の精度に影響を与えない最適化のみを適用
extended：	計算の中間結果を 64 ビット（extended）精度で丸め、解の精度に影響を与えない最適化のみを適用
strict：	precise と except を適用し、縮約統合を無効にし、#pragma fenv_access を有効化
except：	浮動小数点例外セマンティックスを有効化

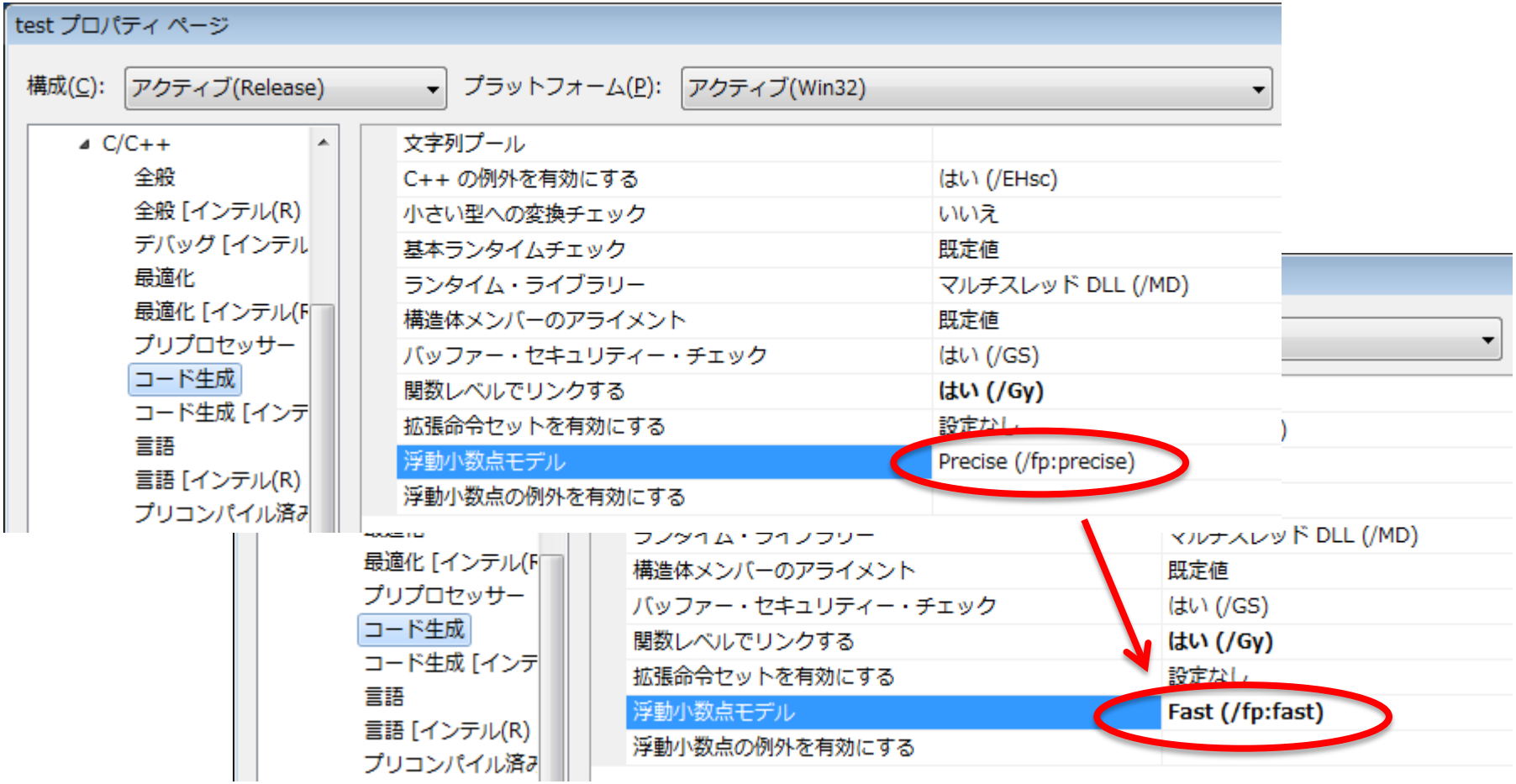
浮動小数点演算モデル（オプション）

以下の表は、浮動小数点演算に関する最適化項目とキーワードの関係を示します。

キーワード	値の安全性	式の評価	制御レジスターへのアクセス	縮約統合	厳密な FP 例外
precise source double extended	高	環境依存 source 依存 double extended	無	有	無
strict	高	環境依存	有	無	有
fast=1 (デフォルト)	中	不明	無	有	無
fast=2	低	不明	無	有	無
except except-	影響なし 影響なし	影響なし 影響なし	影響なし 影響なし	影響なし 影響なし	有 無

浮動小数点演算モデル（Visual Studio* 2010）

Visual Studio* 2010 では、/fp:precise がデフォルト設定されています。
パフォーマンスが必要な場合は、適宜 /fp:fast などに切り替えてください。



～ 内容 ～

1．インテル® コンパイラーの概要
2．インテル® コンパイラーの基本使用方法
3．最適化オプションについて
4．高速インテルライブラリーの利用
　- Intel® MKL（Math Kernel Library）の紹介
　- Intel® IPP（Integrated Performance Primitives ）の紹介
5．最後に

Intel® MKL（数値演算用ライブラリー）

- ◆BLAS（Basic Linear Algebra Subprograms）
  - ベクトル演算(レベル1)、ベクトル - 行列演算(レベル2)、行列 - 行列演算(レベル3)
- ◆スパース BLAS
  - Sparse BLAS レベル1、2、3（sparse ベクトル/行列）
- ◆LAPACK（ Linear Algebra PACKage ）
  - ソルバーおよび固有ソルバー
- ◆ScaLAPACK（Scalable Linear Algebra PACKage）
  - 計算、ドライバ、補助ルーチンの分散型メモリー版
- ◆DFT（Discrete Fourier Transform）
  - 混合基数、多次元変換
- ◆クラスター DFT
  - DFT 分散型メモリー版
- ◆スパースソルバー（PARDISO／DSS ／ISS）
  - 実数または複素数、対称、構造対称または非対称、正定値、不定値またはエルミートのスパース連立線形方程式
- ◆VML（Vector Math Library）
  - 高速な libm 関数
- ◆VSL（Vector Statistical Library）

Intel® IPP（マルチメディア用ライブラリー）

Intel® Integrated Performance Primitives 16 種類の関数ドメイン

画像およびビデオ

- 画像処理
- 色変換
- JPEG/JPEG2000
- ビデオ・コーディング
- コンピュータ・ビジョン
- レンダリング

信号処理

- 信号処理
- オーディオ・コーディング
- 音声コーディング
- 音声認識
- ベクトル演算

データ処理

- データ圧縮
- 暗号化
- ストリング処理
- 行列演算
- データ保全性

IPP サンプルコード：
http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-code-samples/

～ 内容 ～

技術情報サイト(iSUS) - http://www.isus.jp/

Developers

Rock your code.

参考資料

- インテル® Parallel Studio XE 製品ページ
  http://www.xlsoft.com/jp/products/intel/studio_xe/index.html

- 「インテル® C++ Composer XE 2011 Windows* 版 日本語 入門ガイド」
  http://jp.xlsoft.com/documents/intel/cwin/ICC_J_GSG.pdf

- 「インテル Visual Fortran Composer XE 2011 日本語版 入門ガイド」
  http://jp.xlsoft.com/documents/intel/fwin/IVF_J_GSG.pdf

- 書籍の紹介ページ
  http://www.xlsoft.com/jp/products/intel/tech/books.html

- サポート
  http://www.xlsoft.com/jp/products/intel/support/index.html

最適化に関する注意事項

インテル® コンパイラー、関連ライブラリーおよび関連開発ツールには、インテル製マイクロプロセッサーおよび互換マイクロプロセッサーで利用可能な命令セット（ SIMD 命令セットなど） 向けの最適化オプションが含まれているか、あるいはオプションを利用している可能性がありますが、両者では結果が異なります。また、インテル® コンパイラー用の特定のコンパイラー・オプション（ インテル® マイクロアーキテクチャーに非固有のオプションを含む） は、インテル製マイクロプロセッサー向けに予約されています。これらのコンパイラー・オプションと関連する命令セットおよび特定のマイクロプロセッサーの詳細は、『インテル® コンパイラー・ユーザー・リファレンス・ガイド』の「コンパイラー・オプション」を参照してください。インテル® コンパイラー製品のライブラリー・ルーチンの多くは、互換マイクロプロセッサーよりもインテル製マイクロプロセッサーでより高度に最適化されます。インテル® コンパイラー製品のライブラリー・ルーチンの多くは、互換マイクロプロセッサーよりもインテル製マイクロプロセッサーでより高度に最適化されます。インテル® コンパイラー製品のコンパイラーとライブラリーは、選択されたオプション、コード、およびその他の要因に基づいてインテル製マイクロプロセッサーおよび互換マイクロプロセッサー向けに最適化されますが、インテル製マイクロプロセッサーにおいてより優れたパフォーマンスが得られる傾向にあります。

インテル® コンパイラー、関連ライブラリーおよび関連開発ツールは、互換マイクロプロセッサー向けには、インテル製マイクロプロセッサー向けと同等レベルの最適化が行われない可能性があります。これには、インテル® ストリーミング SIMD 拡張命令 2（ インテル® SSE2）、インテル® ストリーミング SIMD 拡張命令 3（ インテル® SSE3）、ストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。

インテルでは、インテル製ではないマイクロプロセッサーに対して、最適化の提供、機能、効果を保証していません。本製品のマイクロプロセッサー固有の最適化は、インテル製マイクロプロセッサーでの使用を目的としています。インテルでは、インテル® コンパイラーおよびライブラリーがインテル製マイクロプロセッサーおよび互換マイクロプロセッサーにおいて、優れたパフォーマンスを引き出すのに役立つ選択肢であると信じておりますが、お客様の要件に最適なコンパイラーを選択いただくよう、他のコンパイラーの評価を行うことを推奨しています。インテルでは、あらゆるコンパイラーやライブラリーで優れたパフォーマンスが引き出され、お客様のビジネスの成功のお役に立ちたいと願っております。お気づきの点がございましたら、お知らせください。

改訂 #20110307