インテル® TBB ライブラリーと C++11 における正確な例外伝播 (exception_ptr)

この記事は、インテル® デベロッパー・ゾーンに公開されている「Exact exception propagation(exception_ptr) in the Intel® Threading Building Blocks (Intel® TBB) library and C++11」の日本語参考訳です。

C++03 と C++11 の例外およびマルチスレッディング

マルチスレッディングも、例外によるエラー処理も今日では珍しいものではありません。並列アルゴリズムの実行中に発生した例外がマスタースレッドに渡され、呼び出しから並列アルゴリズム関数にスローされることを、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) ライブラリーのユーザーが期待するのは理にかなっていると言えるでしょう。

例えば、次のコードは my_error **例外をスローします。**

```
struct my_error;
struct parallel_for_body {
    .....
    void operator()( const tbb::blocked_range<int>& range ) const {
        ...
        if (error){
            throw my_error();
        }
    }
};
```

この例外は並列アルゴリズムを呼び出すスレッドによって処理されることが期待されます。

```
try {
    tbb::parallel_for( tbb::blocked_range<int>( 0, SomeNumber ),
parallel_for_body() );
} catch(my_error& ve){
    // ハンドラーのコード
}
```

並列アルゴリズムの実行中、ユーザーによって記述されたコードは、内部ライブラリー・コードでラップされ、いくつかのワーカースレッドによって実行されます。上記の例外処理モデルでは、ライブラリーで以下の処理を行う必要があります。

1. **ワーカースレッドで例外をキャッチします。**
2. **キャッチした例外をマスタースレッドに渡し、マスタースレッドで再度スローします。**

C++03 **では、情報の一部が欠落することなく (スライシング)、またはすべての例外を事前に指定せずに、キャッチした例外をスレッド間で転送することは不可能です。**C++03 **にはスレッドの概念がないため、異なるスレッドで実行中の** 2 **つのユーザーコード領域間で例外を転送する場合、インテル®** TBB **ライブラリー**

のすべてのユーザー例外を tbb::captured_exception 例外に変更する必要があります。つまり、実際にスローされる例外ではなく、インテル® TBB 呼び出しからの tbb::captured_exception を処理します。

C++11 では、スレッド間の例外伝播が可能になりました[1]。これは、インテル® TBB のようなスレッド・ライブラリーにとって朗報です。

以前の記事で紹介した簡単な例[2] を使って、この "正確な" 例外伝播を説明します。

一言で言えば、どんな例外でも変更することなく伝播できます。

そのため、並列コードが `std::out_of_range` と `CMyDataCorruption` をスローする場合、次のようなコードを記述できます。

```
try {
       parallel_algorithm (/* … */);
} catch ( std::out_of_range& e ) {
       // ハンドラーのコード
} catch ( CMyDataCorruption& e ) {
       // ハンドラーのコード
}
```

この C++11 の機能を利用することで、インテル® TBB ユーザーは "自然な" 方法で例外を処理できるようになり、どのスレッドがスローした場合でも、コードでスローされたのと同じ例外の種類をキャッチできます。

拡張

バージョン 4.2 から、インテル® TBB は OS X*、Windows*、および Linux* で "正確な" 例外伝播をサポートしています。

正確な例外伝播を有効にする

正確な例外伝播を利用するには、いくつかの要件を満たす必要があります。

1. C++11 の `std::exception_ptr` をサポートするコンパイラーが必要です: GCC* 4.4 以上、Microsoft* Visual C++* 2010 以上、Clang* 2.9 以上、またはインテル® C++ コンパイラー 12 以上 (OS X* の場合はインテル® C++ コンパイラー 14 以上)。
2. インテル® TBB を使用するコードは、C++11 モードを有効にしてコンパイルする必要があります。
   - GCC*/Clang*/インテル® C++ コンパイラー (Linux*/OS X* の場合): -std=c++11 オプションを指定します。
   - Visual C++*/インテル® C++ コンパイラー (Windows* の場合): 利用可能な場合は自動で有効になります。
3. OS X* の場合: libc++ (Clang* C++ 標準ライブラリー) を使用する必要があります[3]。libc++ を使用する場合のみ、正確な例外伝播とその他のライブラリー依存の C++11 機能[4] を利用できます。
   - Clang* とインテル® C++ コンパイラーでは、-stdlib=libc++ を指定します。次に例を示します。

     ```
     clang++ -stdlib=libc++ -std=c++11 concurrent_code.cpp -ltbb
     ```

4. リンクする適切なインテル® TBB バイナリーが必要です。

適切なインテル® TBB バイナリーの選択

さまざまなプラットフォーム・ライブラリーへの移植を可能にするため、インテル® TBB は次の 2 つのユーザー例外伝播モードをサポートしています: `tbb::captured_exception` への変換 (C++03 モード) と正確な例外伝播 (C++11 モード)。

正確な例外伝播をサポートするには、適切なコンパイラーと標準ライブラリーを使用し、C++11 サポートを有効にしてインテル® TBB ライブラリーをビルドする必要があります。そのため、各プラットフォームに対し複数のバイナリーが提供されています。

Windows*

Windows* では、インテル® TBB 2.2 以降は、Microsoft* Visual Studio* 2010 以上で正確な例外伝播をサポートしています。

Linux*

下位互換性を保持するため、インテル® TBB 4.2 には 2 つのバイナリーが含まれています。

- GCC* 4.1.2 ランタイムとリンクされたバイナリー。`lib\{ia32,intel64}\gcc4.1` にあります。
- GCC* 4.4.4 ランタイムとリンクされたバイナリー。`lib\{ia32,intel64}\gcc4.4` にあります。

1 つ目のバイナリーは、GCC* 4.1.2 以上のランタイムとともに C++03 モード (すべての例外を `tbb::captured_exception` に変換) で使用します。2 つ目のバイナリーは、GCC* 4.4.4 以上のランタイムとともに C++03 または C++11 (正確な例外伝播) モードで使用します。

注: これらのランタイムは完全に互換ではありません。特に、GCC* 4.4.4 ランタイムとリンクしたアプリケーションを GCC* 4.1.2 ランタイムで実行することはできません。

OS X*

下位互換性を保持するため、インテル® TBB 4.2 には 2 つのバイナリーが含まれています。

- GCC* 4.1.2 ランタイムとリンクされたバイナリー。
- C++11 サポートを有効にして libc++ とリンクされたバイナリー (libc++ サブフォルダーにあります)。

1 つ目のバイナリーは、GCC* ランタイムとともに C++03 モードで使用します。2 つ目のバイナリーは、Clang* ランタイムとともに C++11 または C++03 モードで使用します。

注: これらのランタイムは完全に互換ではありません。あるライブラリーにリンクし、別のライブラリーを使用して実行することはできません。

環境設定

インテル® TBB に付属の tbbvars 環境設定スクリプトを使用することで、リンカーが適切なライブラリーにアクセスできるよう設定できます。

- OS X* では、インテル® TBB が有効な Clang* ランタイムとリンクするため libc++ 引数を指定する必要があります。

  ```
  tbbvars.sh libc++
  ```

- Linux* では、検出された gcc バージョンに基づいて自動的にライブラリーが選択されます。

  ```
  tbbvars.sh intel64
  ```

- Windows* では、使用する Microsoft* Visual Studio* のバージョンを指定する必要があります。

  ```
  tbbvars.bat intel64 vs2010
  ```

ソースからライブラリーをビルドする

インテル® TBB のオープンソース版を使用する場合は、ビルドプロセスに影響する 2 つの makefile 変数を使用します。

- cpp0x – インテル® TBB で c++11 サポートを有効/無効にします。
- stdlib – OS X* で Clang* とインテル® C++ コンパイラーの C++ 標準ライブラリーとして libc++ を有効にします。

インテル® C++ コンパイラーで正確な例外伝播が有効なライブラリーをビルドするには、次のコマンドを使用します。

- Windows*:

  ```
  make tbb compiler=icl
  ```

- Linux*:

  ```
  make tbb compiler=icc cpp0x=1
  ```

- OS X*:

  ```
  make tbb compiler=icc cpp0x=1 stdlib=libc++
  ```

まとめ

正確な例外伝播は、C++11 によってインテル® TBB ユーザーにもたらされる利点の 1 つです。並列アルゴリズムでも、シリアル関数と同様に例外を利用することができます。インテル® TBB 4.2 以上は、Windows*、Linux*、OS X* をサポートしており、例外処理方法として推奨されています。`captured_exception` クラスを使用する以前のアプローチは、下位互換性のため C++11 でもサポートされていますが、推奨されていません。

参考文献 (英語)

1. "Copying and rethrowing exceptions"
2. "Exception handling in TBB 2.2 - Getting ready for C++0x"
3. libc++

4. "Clang and standard libraries on Mac OS X"

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください