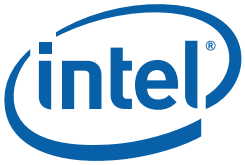

インテル® スレッディング・ビルディング・ブロック

入門ガイド

インテル® スレッディング・ビルディング・ブロックは、スレッドを使用する C++ コード向けのランタイムベースの並列プログラミング・モデルです。テンプレート・ベースのランタイム・ライブラリーで構成され、マルチコア・プロセッサーの性能を最大限に引き出します。インテル® スレッディング・ビルディング・ブロックを使用すると、以下のようにスケーラブルなアプリケーションを記述できます。

- スレッドの代わりにタスクを指定する
- データ並列プログラミングを強調する
- 同時収集および並列アルゴリズムを活用する

本ガイドは、インテル® スレッディング・ビルディング・ブロックを使用して並列アプリケーションを記述、コンパイル、リンク、および実行する例を示します。例では、ライブラリーの主な特徴と、アプリケーションを正しくビルドしリンクする方法について説明します。このガイドに従って一通り作業を行ったら、独自のコードに対してインテル® スレッディング・ビルディング・ブロックを活用する準備は完了です。

目次

著作権/法律に基づく表示

改訂履歴

文書番号	リビジョン番号	説明	改訂日付
314904-001	001	新しいテンプレートを適用。	2006 年 8 月

1 デフォルトのフォルダーパス

最初に、インテル® スレッディング・ビルディング・ブロックがシステムに正しくインストールされていることを確認してください。正しくインストールされていない場合、INSTALL.txt の説明に従ってインストールを行ってください。

本ガイドで使用されている bin、doc、および examples フォルダーのデフォルトの場所を以下に示します。

プラットフォーム	デフォルトのフォルダー
Linux*	/opt/intel/tbb/1.0/[bin|doc|examples]
Mac OS*	/Library/Frameworks/Intel_TBB.framework/Versions/1.0/[bin|doc|examples]
Windows* (インテル® IA-32 プロセッサー)	C:\Program Files\Intel\TBB\1.0\[doc|examples]
Windows* (インテル® 64 命令セット・アーキテクチャー (ISA))	C:\Program Files (x86)\Intel\TBB\1.0\[doc|examples]

2 Michael Voss 環境変数の設定 (Linux* および Mac OS* のみ)

インテル® スレッディング・ビルディング・ブロックを使用する前に、以下のように必要なライブラリーとインクルード・ファイルの場所を含む環境変数を設定する必要があります。

1. 設定スクリプトが含まれているフォルダーに移動します。スクリプトは、`bin` フォルダーにあります。
2. 使用しているオペレーティング・システム用の適切なスクリプトを実行します。Linux* および Mac OS* システムでは、bin フォルダーから source コマンドで `tbbvars.[c]sh` スクリプトを実行します。スクリプトは、`LD_LIBRARY_PATH`、`DYLD_LIBRARY_PATH` (Mac OS)、および `CPATH` 変数のパスを変更します。変更は、現在のシェルにのみ影響します。

3 parallel_for を使用するアプリケーションの作成

このセクションでは、文字列一致サンプルプログラムで parallel_for テンプレートを使用する基本的な例を示します。文字列の各位置について、プログラムは文字列の他の部分と一致する最大部分文字列の長さと位置を表示します。

例として、文字列 “babba” を考えてみます。位置 0 から開始する “ba” は、文字列中で他の部分 (位置 3) と一致する最大の部分文字列です。

このプログラムの完成版は、インテル® スレッディング・ビルディング・ブロックの examples/GettingStarted フォルダーにあります。または、指示に従ってコードを追加し、アプリケーションを完成させることもできます。このセクションでは、各ステップで追加するコードを **青**で示します。前のステップから存在するコードは黒で示します。行番号は最終的に完成した例の行番号を表しています。

コード例を作成するには

1. 新しい空のアプリケーションを作成します。
   main 関数に、task_scheduler_init オブジェクトを追加して、タスク・スケジューラーを初期化します (37 行目)。
   - インテル® スレッディング・ビルディング・ブロックのアルゴリズムを使用するスレッドはすべて、task_scheduler_init オブジェクトで初期化する必要があります。この例では、task_scheduler_init オブジェクトのデフォルト・コンストラクターは、スレッドがタスク実行に関係していることをタスク・スケジューラーに通知します。また、デストラクターは、スレッドがスケジューラーを必要としなくなったことをスケジューラーに通知します。
   - task_scheduler_init クラスの定義は、task_scheduler_init.h ヘッダーファイルからインクルードされます (04 行目)。
   - using ステートメントは、見つかったライブラリーのクラスと関数がすべて含

まれるネーム空間 `tbb` をインポートします (07 行目)。

```
04:   #include
       “tbb/task_scheduler_init.h”

07:   using namespace tbb;

36:   int main() {
37:     task_scheduler_init init;
47:     return 0;
48:   }
```

2. プログラムによって変換される例文字列 (38 - 40 行目) および最も一致した部分文字列と位置を保持する配列 (41 - 42 行目) を作成します。
   この例は、一連の ‘a’ および ‘b’ からなるフィボナッチ文字列を生成します。
3. 各位置で一致した最大部分文字列の長さと位置を出力するステートメントを追加します (45 - 46 行目)。

```
01:     #include <iostream>
02:     #include <string>
04:     #include "tbb/task_scheduler_init.h"

07:     using namespace tbb;
08:     using namespace std;
09:     static const size_t N = 23;

36:     int main() {
37:      task_scheduler_init init;

38:     string str[N] = { string("a"), string("b") };
39:     for (size_t i = 2; i < N; ++i) str[i] = str[i-1]+str[i-
        2];
40:     string &to_scan = str[N-1];

41:     size_t *max = new size_t[to_scan.size()];
42:     size_t *pos = new size_t[to_scan.size()];

43–44:  // will add code to populate max and pos here

45:      for (size_t i = 0; i < to_scan.size(); ++i)
```

```
46:       cout << " " << max[i] << "(" << pos[i] << ")" <<
          endl;
47:       return 0;
48:       }
```

4. parallel_for テンプレート関数への呼び出しを追加します (43 - 44 行目)。
   呼び出しの最初のパラメーターは、反復空間について記述する blocked_range オブジェクトです。
   blocked_range は、インテル® スレッディング・ビルディング・ブロック・ライブラリーによって提供されるテンプレート・クラスです。コンストラクターは、以下の 3 つのパラメーターを受け取ります。
   - 範囲の下限。
   - 範囲の上限。
   - <grainsize>。parallel_for は、範囲を約 *<grainsize>* 要素のサブ範囲に分けます。

   parallel_for 関数の 2 番目のパラメーターは、各サブ範囲に適用される関数オブジェクトです。

```
01:   #include <iostream>
02:   #include <string>

04:   #include “tbb/task_scheduler_init.h”
05:   #include “tbb/parallel_for.h”
06:   #include “tbb/blocked_range.h”

07:   using namespace tbb;
08:   using namespace std;
09:   static const size_t N = 23;

36:   int main() {
37:    task_scheduler_init init;

38:    string str[N] = { string(” a” ), string(” b” ) };
39:    for (size_t i = 2; i < N; ++i) str[i] = str[i-
      1]+str[i-2];
40:    string &to_scan = str[N-1];

41:    size_t *max = new size_t[to_scan.size()];
```

```
42:   size_t *pos = new size_t[to_scan.size()];

43:   parallel_for(blocked_range<size_t>(0,  to_scan.size(),
      100),
44:         SubStringFinder( to_scan, max, pos ) );

45:   for (size_t I = 0; I < to_scan.size(); ++i)
46:    cout << ” ” << max[i] << ” (” << pos[i] << ” )” <<
      endl;
47:   return 0;
48:  }
```

5. `parallel_for` ループのボディーを実装します (10 - 35 行目)。

   実行時に、テンプレート `parallel_for` は範囲を自動的にサブ範囲に分割して、各サブ範囲で `SubStringFinder` 関数オブジェクトを起動します。

6. クラス `SubStringFinder` を定義して (10 行目)、指定されたサブ範囲内で見つかった max および pos 配列要素を埋めます。

   16 行目の `r.begin()` はサブ範囲の最初、`r.end()` はサブ範囲の最後をそれぞれ返します。

```
01:    #include <iostream>
02:    #include <string>

03:    #include <algorithm>
04:    #include “tbb/task_scheduler_init.h”
05:    #include “tbb/parallel_for.h”
06:    #include “tbb/blocked_range.h”

07:    using namespace tbb;
08:    using namespace std;
09:    static const size_t N = 23;

10:    class SubStringFinder {
11:    const string str;
12:     size_t *max_array;
13:     size_t *pos_array;
14:    public:
15:    void operator() ( const blocked_range<size_t>& r )
       const {
16:      for ( size_t I = r.begin(); I != r.end(); ++I ) {
17:       size_t max_size = 0, max_pos = 0;
```

```
18:       for (size_t j = 0; j < str.size(); ++j)
19:        if (j != i) {
20:         size_t limit = str.size()-max(I,j);
21:         for (size_t k = 0; k < limit; ++k) {
22:          if (str[I + k] != str[j + k]) break;
23:          if (k > max_size) {
24:           max_size = k;
25:           max_pos = j;
26:          }
27:         }
28:       }
29:       max_array[i] = max_size;
30:       pos_array[i] = max_pos;
31:      }
32:     }
33:     SubStringFinder(string &s, size_t *m, size_t *p) :
34:       str(s), max_array(m), pos_array(p) { }
35:     };

36–48:  // The function main starting at line 36 goes here
```

4 アプリケーションのビルド

インテル® スレッディング・ビルディング・ブロックは、GCC* および Microsoft* コンパイラーと互換性がありますが、このセクションでは、インテル® C++ コンパイラーを使用していると仮定します。GCC または Microsoft* C++ コンパイラーを使用している場合は、環境に応じてコマンドを変更してください。

examples フォルダーのコードをビルドする

「3. `parallel_for` を使用するアプリケーションの開発」でコードを入力しなかった場合、`examples/GettingStarted` フォルダーにあるソースコードをビルドします。

Linux* または Mac OS* システム

1. `cd` コマンドを使用して `examples/GettingStarted/sub_string_finder` フォルダーに移動します。
2. “make” と入力して、例をビルドします。

Windows* システム

注: このセクションでは Visual Studio* .NET* 2003 (vc7.1\) を使用しています。Visual Studio* 2005 を使用する場合は、`vc7.1\` を `vc8\` に置き換えてください。

1. 以下のいずれかの方法を使用して、Visual Studio* で `examples\GettingStarted\sub_string_finder\vc7.1\sub_string_finder.sln` ファイルを開きます。
   - `sub_string_finder.sln` ファイルが含まれているフォルダーに移動して、ファイルをダブルクリックします。
   - [スタート] メニューから Visual Studio* を起動し、**[ファイル] - [開く] - [プロジェクト]** (Visual Studio* 2005 の場合は [プロジェクト/ソリューション]) を選択して、`sub_string_finder.sln` ファイルを開きます。
2. <Ctrl-F5> を押して例をビルド、実行します。

手動で入力したコードをビルドする

「3. `parallel_for` を使用するアプリケーションの開発」でコードを手動で入力した場合、直接コンパイラーを起動してアプリケーションをビルドします。

Linux* または Mac OS* システム

コマンドラインで、次のコマンドを入力します。

```
icc sub_string_finder.cpp -ltbb
```

Windows* システム (コマンドプロンプトから)

インテル® C++ コンパイラーのビルド環境から、次のコマンドを入力します。

```
icl /MD sub_string_finder.cpp tbb.lib
```

5 アプリケーションの実行

ビルドしたアプリケーションを実行するには

1. 通常通りにアプリケーションを実行します。実行後、しばらくすると、長さと位置がペアになったリストが出力されます。
2. (オプション) この例と直列実行版の性能を比較するには、インテル® スレッディング・ビルディング・ブロックの examples/GettingStarted フォルダーにある例 `SubStringFinder` の拡張版 `sub_string_finder_extended.cpp` をビルドして実行してください。この拡張版は、インテル® スレッディング・ビルディング・ブロックのアルゴリズムを使用して並列化した場合と、同じ作業を直列で行った場合を比較し、どの程度スピードアップしたかを計算して表示します。

6 次のステップ

インテル® スレッディング・ビルディング・ブロック・ライブラリーを最大限に活用するには、以下のリソースも参照してください。

1. **チュートリアル**
   インテル® スレッディング・ビルディング・ブロックで使用されている主なクラス、アルゴリズム、および概念を紹介するドキュメントが、 `doc` フォルダーにあります。ファイル名は、`Tutorial.pdf` です。
2. **リファレンス**
   インテル® スレッディング・ビルディング・ブロックで提供されているすべての関数とインターフェイスの詳細が含まれるリファレンス・マニュアルが、`doc` フォルダーにあります。ファイル名は、`Reference.pdf` です。
3. **サンプル**
   インテル® スレッディング・ビルディング・ブロックのさまざまな機能を紹介するプログラム例のコレクションが `examples` フォルダーにあります。最初に、この**「入門ガイド」**で使用した例 `SubStringFinder` の拡張版を試してみると良いでしょう。この拡張版は、`examples/GettingStarted/sub_string_finder` サブフォルダーにあります。ファイル名は、`sub_string_finder_extended.cpp` です。実行すると、インテル® スレッディング・ビルディング・ブロックのアルゴリズムを使用して並列化した場合と、同じ作業を直列で行った場合を比較し、どの程度スピードアップしたかを計算して表示します。
4. **Doxygen**
   インテル® スレッディング・ビルディング・ブロックのインクルード・ファイルに含まれているコメントから自動的に生成されたドキュメントです。このドキュメントは、`doc` フォルダー内の `html` サブフォルダーにあります。`html` フォルダーのファイルは HTML 形式です。HTML をサポートしているブラウザーで表示してください。