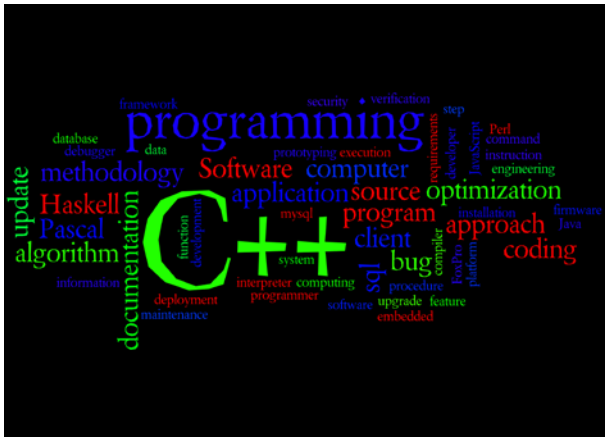
C++ コンパイラーの比較

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「Comparing C++ Compilers」の日本語参考訳です。



並列化およびベクトル化コードのパフォーマンスを最大限に引き出すには

並列化およびベクトル化されたコードを記述する場合、使用する C++ コンパイラーによって違いがあるのでしょうか? この記事では、Slashdot Media 寄稿編集者の Rick Leinecker 氏がコンパイラーの選択方法とその理由について述べます。

Visual Studio* 1.0 以前のコマンドライン・コンパイラーの時代から長年にわたって、私は Microsoft* 開発ツールを支持してきました。私の部下であるプログラマー達が Turbo C と Microsoft* C を比較したとき、さまざまな点で激しい論議になりましたが、私は常に Microsoft* C を支持してきました (そして、もちろん IT ディレクターであった私が議論に勝ちました)。

近年、私はインテル® C++ コンパイラーに大きな敬意を抱くようになりました。使い始めたのは、インテル® Parallel Studio をインストールしたときからですが、インテル® コンパイラーは Visual Studio* に簡単に統合できました。Visual Studio* で 1 つのサブメニューを選択するだけで、Microsoft* コンパイラーの代わりにインテル® コンパイラーを使用することができます。私のベンチマーク・テストでは、インテル® コンパイラーでコンパイルしたプログラムは、ほとんどの場合 Microsoft* コンパイラーでコンパイルしたものよりも高速で、多くの場合サイズも小さいです。これは、ソフトウェア開発者にとって願ったり叶ったりです。

ベクトル化の例

最初の例は、2 つのコンパイラーの自動ベクトル化機能を比較するために作成しました。自動ベクトル化は、プロセッサーのベクトル化 (SIMD とも呼ばれる) 命令を利用してパフォーマンスを向上する機能で、どちらのコンパイラーもこの機能を提供しています。以下に、コードを示します。

```
// 初期化
for (int j = 0; j < ROW; j++) {
    for (int i = 0; i < COL; i++) {
        Data[j][i] = i*j;
    }
}

// ベクトル化されるべき処理を実行
for (int j = 0; j < ROW; j++) {
    for (int i = 0; i < COL; i++) {
        sum += Data[i][j] + Data[j][i];
    }
}
```

ROW の値は 20000、COL の値は 20000 にそれぞれ定義して、入れ子のループが sum 変数に 400,000,000 回加算するようにしました。両コンパイラーの自動ベクトル化機能を検証するには、これで十分です。インテル® コンパイラーは、Microsoft* コンパイラーよりもやや優れていました。このプログラムを実行後、次のように **pragma** ディレクティブを使用することでパフォーマンスを向上できる可能性があることに気付きました。このディレクティブは、コンパイラーにヒントを与えることでより多くの自動ベクトル化を可能にします。

```
// ベクトル化されるべき処理を実行
for (int j = 0; j < ROW; j++) {
#pragma omp simd reduction(+:sum)
    for (int i = 0; i < COL; i++) {
        sum += Data[i][j] + Data[j][i];
    }
}
```

pragma なしの場合/ありの場合
Microsoft* コンパイラー: 6120ms/5772ms
インテル® コンパイラー: 5902ms/5382ms

訳者注: #pragma omp simd は OpenMP* 4.0 仕様から利用可能になりましたが、Visual C++* は OpenMP* 2.0 仕様をサポートしているため利用できません。コンパイラーがサポートする OpenMP* 仕様のバージョンを確認するには、こちらの記事をご覧ください。

並列化されたループ

次に、私は並列化されたループを比較してみました。**pragma** 修飾子を使用して並列化する際の標準的な方法である **OpenMP*** を使用しました。次のコードは、前述のコード例のループを並列化したものです。**pragma omp parallel for reduction(+:sum)** ディレクティブが追加されています。これだけでループが並列化されます。reduction 節は、sum 変数でデータ競合が発生しないようにします。内部的に各スレッドは変数の個別のコピーを持っているため、**parallel for** ループの最後でそれらを合計する必要があります。

```
// ベクトル化されるべき処理を実行
for (int j = 0; j < ROW; j++) {
#pragma omp parallel for reduction(+:sum)
    for (int i = 0; i < COL; i++) {
        sum += Data[i][j] + Data[j][i];
    }
}
```

ここでも、インテル® コンパイラーでコンパイルしたコードのほうが高速でした。以下に、それぞれの結果を示します。

Microsoft* コンパイラー: 5877ms

インテル® コンパイラー: 5438ms

まとめ

Microsoft* コンパイラーとインテル® コンパイラーは、どちらも優れた開発ツールです。パフォーマンスを最大限に引き出す必要がある場合、私はインテル® コンパイラーを使用します。そうでない場合、すべての条件が同じであれば、わざわざコンパイラーを切り替えることはせずに、デフォルトの Microsoft* コンパイラーを使用します。

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。