C、C++コンパイラのツール認定の効用

The Benefits of C and C++ Compiler Qualification

概要

組み込みアプリケーション開発で、コンパイルツールセットの正しい動作は、アプリケーションの機能安全にとって非常に重要です。そしてコンパイラが正しく利用できることの信頼を確立する二つの方法があります：テストによるコンパイラのツール認定（qualification）、あるいはアプリケーションのテストカバレッジをマシンコードレベルで評価することです。この資料では、コンパイラをツール認定することの方がはるかに効率的であることを紹介します。加えてコンパイラをツール認定することで、アプリケーション開発は独立した作業として切り離されるため、製品化までの時間と費用を軽減する効果があることも説明します。

自動車産業に於ける ISO26262 のような機能安全標準は、コンパイラのようなツールがセーフティクリティカルなアプリケーション開発に使用される場合にツール認定されなければならないことや、それをどのように実施するかについて記載しています。ツール認定は、ツールを使用するにあたって十分な確信をもてるようにするための、機能安全標準に記載されるプロセスです。

コンパイラは複雑なソフトウェア（2～5 百万規模のコードサイズ）であり、ソースコードからマシンコードへの変換で重要な役目を果たします。変換されたマシン語はセーフティクリティカルなアプリケーションやデバイスの一部となるため、生成されたコード上のエラーがセーフティクリティカルな事象を引き起こし得ます。

コード生成においてコンパイラがエラーを出さないという根本的な要求を満たすため、ISO26262 は重要な選択を行うことを要請しています。「ツール認定によってコンパイラが十分信頼できるか」、「生成されたマシンコードでどんなエラーも発見しうる十分に強力なアプリケーションテスト手順を作成するか」のどちらかが必要、ということです。本文書ではコンパイラのツール認定を選択するほうがより効果的であることを示します。

まず初めに、コンパイラがツール認定されていない場合に必要となる作業を見てみましょう。「*（コンパイラの）不具合とそれに起因する誤った出力が防止されるか検知されることが高い確度で確信できる*」場合に、そのコンパイラはツール認定される必要はない（ISO26262, Part8, 11.4.5.2.b.1）。これは非常に多くの作業と費用を費やすことになるでしょう。というのも、そのような確信を得るには「*マシンコードのレベルでアプリケーションをターゲット上でテスト実行*」してそのカバレッジ（文、分岐、MC/DC）を解析することが必要だからです。

「*ターゲット上で*」とは、アプリケーション用に作成されたテストが実際のターゲットプロセッサが搭載されるハードウェアでテストされなければならないことを意味し、そのコンパイラが一連のテストプロセス内で使用されるということです。

マシンコードレベルでのカバレッジと MC/DC 解析

ISO26262 の Part 6（製品ソフトウェア）表 12 では単体テストの一部として、ASIL レベルに応じて指定されるカバレッジ解析のレベルを要求しています。これには、ステートメント、分岐、かつ/または MC/DC 解析が入るでしょう。統合テストでは、表 15 において関数と関数コールカバレッジのテストが追加されます。

もしコンパイラを信頼できる確証がない状態で、コンパイラによる最適化が行われたら、この解析はソースコードではなくマシンコードのレベルで実行されなくてはなりません。ソースコードのカバレッジ解析だけでは不十分です。その理由はコンパイラによる最適化はアプリケーションのコードを大きく変換してしまうためで、最適化の結果、ソースコードでは 100%のカバレッジでもマシンコードのレベルでは多くの分岐やコードブロックがカバーされないことになり得ます。それゆえ、ソースコードカバレッジだけでは、「*(コンパイラの) 不具合とそれに起因する誤った出力が防止されるか検知されることが高い確度で確信できる*」ということにならないのです。

付録でループのある単純な関数を解析します。それによってソースコードのカバレッジとマシンコードレベルでのカバレッジの間のギャップがどれほど大きいかを示します。

コンパイラのツール認定がされない場合、コンパイラの不具合が検出されるという確信を得る唯一の方法は、マシンコードレベルでの十分なコードと分岐のカバレッジを示すことです。

注意事項：Section 9.4.5, Note 4

ISO26262 の Part 6（製品ソフトウェア）Section 9.4. 5, Note 4 ではソフトウェアの単体テスト（ここではカバレッジ解析）はソースコードレベルで、その後に「back-to-back」テストでの単体テストを行うことで実施できると述べています。「back-to-back」テストとはターゲット上でのテスト実行の結果とホスト上（エミュレーションかシミュレーション）でのテスト実行の結果を比較することを意味します。

この注はアプリケーションソフトウェアテストに対する要求を述べたもので、そのアプリケーションソフトウェアが十分にテストされ、実際のハードウェアでも動作することがそれによって主張できます。しかし、前節で議論した理由によって、これはそのコンパイラが信頼できるということにはつながりません。

コンパイラツール認定の効用

代わって、ツール認定されたコンパイラでのアプリケーションテストを考えます。コンパイラのツール認定は、例えばコンパイラの仕様に対するテストでは、コンパイラの正しさに十分な確信を得るために機能安全標準に記載されているプロセスです。それは開発中のアプリケーションとは独立していますが、コンパイラの「ユースケース」：そのアプリケーションをどのようにコンパイルするか、には依存します。例えば、コンパイラの特定のオプション設定や最適化などが含まれます。

ツール認定されたコンパイラを用いると、アプリケーション開発者は、コンパイラの不具合がツール認定プロセスの中で検出されているものと信頼できます。このことは、コンパイラが欠陥なし（そのようなことはほとんどありませんが）ということではなく、その欠陥がアプリケーション開発者に分かっており避けられ得るということを意味します。

ツール認定されたコンパイラを使用するなら、アプリケーションのテストはそのコンパイラによってもたらされる生成物を考慮する必要がなくなります。コンパイラは信頼できるので、その生成物をカバレッジテストで評価する必要がありません。カバレッジテストはソースコードレベルで進めることができます。（ここまでは ISO26262 に対するもので、DO-178C に対しては追加の処置が必要になります。）

信頼できるコンパイラの効用は、アプリケーションのテスト手順を非常に単純に、より効率的にするということです。確かにコンパイラのツール認定自体は簡単なものではありませんが、そのプロセスはそれほどオーバーヘッドなく組織内でなんとかできるプロセスです。以下、いくつかの利点をまとめています。

- コンパイラのツール認定は、新しいコンパイラ（更新）があるときだけなされます。それは多くても年 2～3 回であり、またアプリケーション開発自体に直接影響する作業ではありません。もしコンパイラがツール認定されて無い場合は、マシンコードレベルのカバレッジ解析が必要になるテストは開発作業の一部となり、アプリケーションが更新されるごとに発生します。したがって、ツール認定されたコンパイラを用いることで、製品化までの時間を短縮することができます。

- 一つのアプリケーションが複数のコンパイラでコンパイルされて多くのターゲットプラットフォーム上に展開される場合、ツール認定されたコンパイラはターゲット上でのテストの必要性を大幅に削減します。上で述べたように back-to-back テストはやはり必要ですが、そこでコンパイラの挙動を考慮する必要はありません。信頼できるコンパイラなら、アプリケーションテストはソースコードの検証にフォーカスすることができます。

- コンパイラによって生成されるコードや分岐を、マシンコードレベルでカバーするアプリケーションの単体テストを書くことは、そのようなテストをコンパイラ依存にします。同じテストでも別のコンパイラ、もしくは同じコンパイラの更新バージョンが生成したコードをカバーするには不十分かもしれません。コンパイラが信頼できれば、カバレッジ解析はソースコードのカバレッジにフォーカスできるので、ターゲットのコンパイラやプラットフォームに依存しません。コンパイラに特定した単体テストは必要ないのです。

- コンパイラは高い最適化レベルではコードに対して変換以上のことを行います。そのことは十分なカバレッジを満たすテストを書くことを難しくします。それが理由で使用するコンパイラの最適化レベルを下げることもありますが、パフォーマンスの劣化やアプリケーションによる資源の使用超過にもつながります。そこで、ツール認定されたコンパイラで望ましい最高レベルの最適化によって、より効率的なアプリケーションコードで、ハードウェア資源に対する要求を軽減させることができます。

- ソースコードからマシンコードを生成する重要なステップが正しくなされているという確信が上がります。特にコンパイラのツール認定が、アプリケーションの展開で使用されるものと同じコンパイルオプションの組み合わせによってなされるときにそういえます。

ツール認定されたコンパイラを使用することの最大の利点は、アプリケーションテストが、コンパイラによる生成物ではなくアプリケーションのソースコードにフォーカスできることです。アプリケーション開発者はアプリケーションの正しさのために作業するのであって、使用するツールの正しさのためではないので、アプリケーションのソースコードにフォーカスできることは彼らにとって重要です。アプリケーションへの関心とコンパイルツールに対する関心を分離することで、アプリケーションを開発してそれを複数のターゲットに展開することがより容易に、効率的になります。

コンパイラ使用者のためのコンパイラツール認定

コンパイラツール認定はコンパイラ供給者でなくアプリケーション開発者によって行われることがベストです。なぜなら、ツール認定はコンパイラのアプリケーションユースケースに可能な限り適合しなければならないからです。コンパイラそれぞれがオプションの膨大な組み合わせをもっていることを考慮すると、コンパイラ供給者でアプリケーション開発者が使用する実際のコンパイルオプションをテストしているとは考えにくいことです。

コンパイラのツール認定は注意深く構成されなければなりませんが、過大にチャレンジングなプロセスではありません。必要なものは ISO26262 ガイドライン等の機能安全標準、そしてコンパイラと言語仕様に基づくコンパイラテストスイートを提供する SuperTest for C and C++ です。それらによって別なコンパイラ、もしくは同じコンパイラの更新バージョンに対して容易に繰り返すことができる自動的なツール認定プロセスが構成されます。

以下の例では、GCC に SuperTest の C99 のテストを実行した結果が、言語仕様やライブラリに対するトレーサビリティと共にレポートされています。C/C++言語標準規格を要件とする SuperTest のテストスイートによるテスト結果との対応付けは、コンパイラツール認定のエビデンスとして活用されています。

157	C99:6.7.4	Function specifiers	REVIEW	Ill-formed test compiled with warnings	C99/6/7/4/x5.c
158	C99:6.7.4	Function specifiers	REVIEW	Ill-formed test compiled with warnings	C99/6/7/4/x4.c
159	C99:6.7.4	Function specifiers	REVIEW	Ill-formed test compiled with warnings	C99/6/7/4/x1.c
160	C99:6.7.4	Function specifiers	REVIEW	Ill-formed test compiled with warnings	C99/6/7/4/x2.c
161	C99:6.7.5	Declarators	X-FAILED	Compile time failure	3/5/4/xdeclmax.c
162	C99:6.7.5.1	Pointer declarators	PASSED	All tests passed	
163	C99:6.7.5.2	Array declarators	REVIEW	Ill-formed test compiled with warnings	C99/6/7/5/2/xordin.c
164	C99:6.7.5.2	Array declarators	REVIEW	Ill-formed test compiled with warnings	3/5/4/2/x1.c
165	C99:6.7.5.3	Function declarators (including prototypes)	REVIEW	Ill-formed test compiled with warnings	3/5/4/3/xrfg8.c
166	C99:6.7.5.3	Function declarators (including prototypes)	REVIEW	Ill-formed test compiled with warnings	3/5/4/3/x9.c
167	C99:6.7.5.3	Function declarators (including prototypes)	REVIEW	Ill-formed test compiled with warnings	3/5/4/3/x6.c
168	C99:6.7.5.3	Function declarators (including prototypes)	C-FAILED	Compile time failure	3/5/4/3/t_0001.c
169	C99:6.7.5.3	Function declarators (including prototypes)	REVIEW	Ill-formed test compiled with warnings	3/5/4/3/x2.c
170	C99:6.7.6	Type names	PASSED	All tests passed	
171	C99:6.7.7	Type definitions	PASSED	All tests passed	
172	C99:6.7.8	Initialization	R-FAILED	Runtime Failure	C99/6/7/8/t7.c

317	C99:7.8.2.2	The imaxdiv function	PASSED	All tests passed	
318	C99:7.8.2.3	The strtoimax and strtoumax functions	PASSED	All tests passed	
319	C99:7.8.2.4	The wcstoimax and wcstoumax functions	R-FAILED	Runtime Failure	C99/7/8/2/4/t_0134.c
320	C99:7.8.2.4	The wcstoimax and wcstoumax functions	R-FAILED	Runtime Failure	C99/7/8/2/4/t_0162.c
321	C99:7.9	Alternative spellings <iso646.h>	PASSED	All tests passed	

まとめ

セーフティクリティカルなドメインの、コンパイラユーザーのユースケースに対して、コンパイラ供給者によるコンパイラのツール認定を当てにすることはできません。またコンパイラが正しいことを証明するアプリケーションテストにも多くは期待することはできません。アプリケーションテストとコンパイラツール認定を分離することで、アプリケーション開発はより効率的になり、その結果として費用対効果が高くなります。Solid Sands 社は SuperTest の提供を通じて、御社のコンパイラのツール認定プロセスを支援します。

付録
カバレッジ解析とコンパイラ最適化

以下は一つのループをもつ単純な関数です。

```
int f(int n) {
    int total = 0;
    for(int i = 0; i < n; i++) {
        total += i & n;
    }
    return total;
}
```

ソースコードレベルで解釈すると、このコードは条件 i < n に由来する条件分岐を一つ含みます。この関数に対するテストケースで、引数 n に対してゼロでない正の数で呼び出せば、関数内のすべての文を完全にカバーできます。さらにこの一つの単体テストでこの条件分岐の両方向：少なくとも 1 回ループ本体に入るものと 1 回ループを脱出するもの、を起こせます。したがって、ソースコードレベルでは一つの単体テストがループの MC/DC カバレッジに十分です。

このコードを最適化レベル -O0 でコンパイル（ここでは LLVM ベースの x86 コンパイラ）すると、大体そのままの形でマシンコードに変換されます。いつもそうだという保証はありません（コンパイラツール認定が望ましいという他の論点です）が、調べると、上と同じ単体テストでマシンコードでの完全な MC/DC カバレッジが得られます。

しかし、-O0 レベルではコンパイラはレジスタ割り付けを行わず、すべての変数の操作はスタック上でなされます。結果、得られるコードは実行時に効率が悪くコンパクトでもありません。少なくとも -O1 レベルでコンパイルしたいと思うものです。

最適化レベル -O1 ではコードはずっとコンパクトになり、先の 3 倍の速度になりますが、十分ソースコードに似ていて手作業で比較することもできます。それにもかかわらず、コンパイラは追加の分岐分を生成してループ本体が 1 度も実行されない場合（変数 n が 0 に等しいとき）にショートカットします。この分岐は上の単体テストでは扱えず、この関数を引数値 0 で呼び出す追加の単体テストを生成しなければなりません。そうでなければマシンコードレベルでの完全な分岐カバレッジは得られません。

レベル -O2 でコンパイルすると、さらに 6 倍！のパフォーマンスが得られます。コンパイラはループ展開とベクトル化を実施します。生成されたマシンコードは一目では理解できず、見直しても同じです。コードは 13 個の基本ブロックと 9 個の条件分岐からなります。先の二つのユニットテストでは完全なコードと分岐のカバレッジを得るにはまったく不十分であるばかりか十分な単体テストを作成する容易な方法もありません。マシンコードレベルで満足な MC/DC 解析を行えるツールを持ち合わせていないことがその理由です。ソースコードレベルで MC/DC 解析を行うツールは多く存在しますが、マシンコードレベルでの解析のものはほとんどなく、対応しているプロセッサも限られます。

ところで、-O0 より高い最適化レベルでコンパイラを使用する必要はあるでしょうか？最適化がなければターゲットのコードはソースコードの構造に近く、ソースコードレベルのカバレッジを満たすのに十分なテストで事足りるでしょう。そして最適化については、アプリケーションに依存します。ここでの単純な例では -O0 から -O2 でパフォーマンスに 18 倍の差が出ます。言い換えれば、電力を含め 18 倍のリソースが必要ということです。多くのアプリケーション分野では、性能向上が分かっていてそのままにされるのは受け入れられないでしょう。

FUJI SETSUBI

富士設備工業株式会社 電子機器事業部　www.fuji-setsu.co.jp