

インテル® コンパイラ コード・カバレッジ・ツールおよび テスト優先化ツール

エグゼクティブ・サマリ

このホワイトペーパーでは、インテル® コンパイラの Profile-Guided Optimization (PGO : プロファイルに基づく最適化) テクノロジーをベースにした 2 つの関連ツールを紹介します。最初に紹介するのが、コード・カバレッジ・ツールです。このツールを利用すると、アプリケーションが特定のワークロード (データ) を処理した際に実行されるソースレベルのコード量を把握できます。次に紹介するのが、テスト優先化ツールです。このツールを利用すると、アプリケーションのソースコードに修正を加えても効率よくテストの優先順位を管理できます。いずれのツールも、Windows* および Linux* 向けのインテル® C++ コンパイラとインテル® Fortran コンパイラに付属しています。これらのツールを使えば、ソフトウェアの品質とパフォーマンスが向上するほか、プログラマの生産性が増加します。

コード・カバレッジ・ツールの紹介

コード・カバレッジ・ツールを利用すると、ソフトウェア開発者は、特定のワークロード (データ) をアプリケーションが処理した際に実行されるソースレベルのコード量を把握できます。コード・カバレッジ・ツールは、どのコードが実行されたかを判断するにあたって、Profile-Guided Optimization (PGO : プロファイルに基づく最適化) テクノロジーを使用します。コンパイラによって生成された静的プロファイル情報を分析するだけでなく、インストールされた形式のアプリケーション・バイナリをワークロードに対して実行することにより生成された動的プロファイル情報も分析します。分析結果は HTML 形式で出力され、色分けとコメントが付加されたソースコード・リストを参照することで、実行されたコードと実行されていないコードを簡単に区別できます。情報はフレーム形式で表示されるので、ソースコード内での移動が容易です。また、アプリケーションのファイルや関数を並べ替えて、最も使用されていないモジュールと関数を簡単に判別できます。

コード・カバレッジ・ツールは、以下のようにさまざまな方法で、開発作業の効率化、エラーの削減、アプリケーション・パフォーマンスの向上を実現します。

1. プロジェクト・テスト段階で、実際にテストされるコードの量を示すことにより、テストの全体的な品質を評価できます。
2. パフォーマンス・ワークロードのプロファイルをこのツールで解析すると、ワークロードがアプリケーションの主要コードをどの程度使用しているかが示されます。インテル® コンパイラが提供する PGO を十分に活かすには、パフォーマンスにとって重要なモジュールのカバレッジを高めることが欠かせません。

3. アプリケーションの基本ブロックごとに動的実行回数を表示できるオプションを提供しており、カバレッジとパフォーマンスの両方の解析に役立ちます。
4. 2 つの異なるワークロードの実行プロファイルを比較できます。この機能を利用すると、アプリケーション・コードの中で、テスト時には表示されないが、テスト空間外での実行時 (顧客による実行時など) には使用される部分を判別できます。

コード・カバレッジ・ツールは、Windows* または Linux* オペレーティング環境上で IA-32、インテル® エクステンデッド・メモリ 64 テクノロジーおよびインテル® Itanium® プロセッサ・ファミリを運用しているシステムに対応します。C、C++、Fortran をシームレスにサポートします。

コード・カバレッジ・ツールは、以下に示す 2 段階のカバレッジ情報を生成します。

- トップレベル・サマリ情報
- モジュールごとのソースビュー

トップレベル・カバレッジ

トップレベル・カバレッジ・サマリは、モジュールの全体的なコード・カバレッジを示します。コード・カバレッジ分析に含めるモジュールを選択するには、**-comp** オプションを使用します。選択したモジュールについて、使用されたモジュールのリスト、使用されていないモジュールのリスト、全体のカバレッジ・サマリが生成されます。サマリ情報には、各モジュール内の関数およびブロックの数と、使用された数が示されます。

リストを昇順または降順に並べ替えるには、表示されているテーブル上で対象の列のタイトルをクリックします。リストは、以下のデータに基づいて並べ替えられます。

- 基本ブロック・カバレッジ
- 関数カバレッジ
- 関数名

図 1 に、あるプロジェクトのトップレベル・カバレッジ・サマリの例を示します。モジュール名 (SAMPLE.C など) をクリックすると、そのモジュールのカバレッジ・ソース・ビューがブラウザに表示されます。

図 2 に、SAMPLE.C のカバレッジ・ソース・ビューを示します。

コード・カバレッジ・ツールの色分け方式

コード・カバレッジ・ツールは、使用されたコード、使用されていない基本ブロック、使用されていない関数、部分的に使用されたコードを判別します。カバレッジの分類には、複数の色が使用されます。デフォルトの色は、HTML で有効な色に置き換えられます。表 1 に、デフォルトの色を示します。

色分けによるソースへのコメント付けに関して、コード・カバレッジ・ツールは、コンパイラと異なる方式でアプリケーション・ソースを解析します。色分けには、以下の規則が使用されます。

1. プロファイル情報によって基本ブロックの始まりと判定される位置に到達するまで、ソースの文字をスキャンします。
2. 基本ブロックのプロファイル情報が色の変更の必要性を示している場合、コード・カバレッジ・ツールは該当する色変更を HTML ファイルに対して行います。ただし開発者は、コードの前後関係に応じて色を解釈しなければならないことがあります。例えば、実行されていない基本ブロックの後に続くコメント行は、使用されていないブロックと同じ色になります。C/C++ アプリケーション中の閉じ括弧でも、同じ処理が行われる可能性があります。

図 1. サンプル・プロジェクトのトップレベル・カバレッジ・サマリ

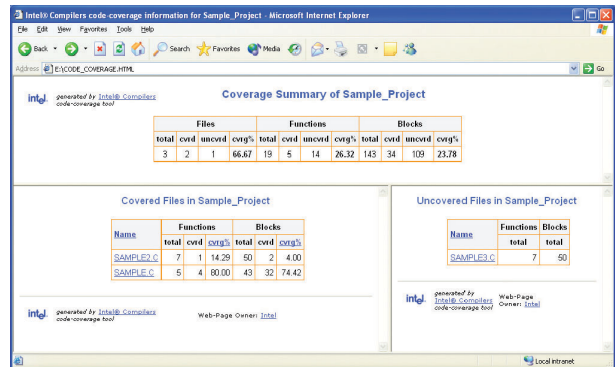


図 2. サンプル・モジュールのカバレッジ・ソース・ビュー

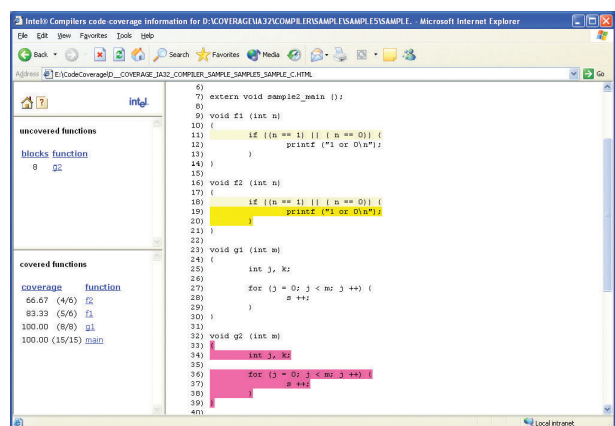


表 1. コード・カバレッジ・ツールのデフォルト色

色	意味
使用されたコード	この色で表示されたコードは、テストで使用されています。この色を変更するには、 -ccolor スイッチを使用してください。
使用されていない基本ブロック	この色で表示された基本ブロックは、テストで使用されていません。ただし、テスト中に実行された関数内では、使用されています。この色を変更するには、 -bcolor スイッチを使用してください。
使用されていない関数	この色で表示された関数は、テスト中に呼び出されていません。この色を変更するには、 -fcolor スイッチを使用してください。
部分的に使用されたコード	この色で表示されたコードでは、複数の基本ブロックが生成されています。使用されたブロックもあれば、使用されていないブロックもあります。この色を変更するには、 -pcolor スイッチを使用してください。
不明	この色で表示されたソースでは、実行されるコードが生成されていません。通常は、コメント、ヘッダファイルのインクルード、変数の宣言です。この色を変更するには、 -ucolor スイッチを使用してください。

ヘルパフレームの閲覧

コード・カバレッジ・ツールで生成されるフレームを利用すると、閲覧しながら、使用されていないコードを判別できます。上のフレームには使用されていない関数のリストが表示され、下のフレームには使用された関数のリストが表示されます。使用されていない関数については、関数ごとの基本ブロック数が表示されます。使用された関数については、ブロック数、使用されたブロックの数、その割合（カバレッジ率）が表示されます。例えば、66.67（4/6）は、該当する関数内で6ブロック中4ブロックが使用されたことを示しています。リストは、カバレッジ率、ブロック数、関数名に基づいて並べ替えられます。関数名は、ソースビュー内で関数の本体が始まっている部分にリンクされています。そのため、1回クリックするだけで、リスト内で最も使用されていない関数を表示できます。もう1回クリックすると、その関数の本体がブラウザ上に表示されます。ソースビューを下にスクロールしていけば、関数の本体をすべて見る事ができます。

動的カウンタ

コード・カバレッジ・ツールでは、カバレッジを示す色分けに加えて、動的実行回数を表示するように設定できます。表示設定を行うには、**-counts** オプションを使用してください。実行回数の情報は、ソース中で該当の基本ブロックが始まる位置のすぐ下にある「**^**」記号の後に表示されます。1カ所のコードに複数の基本ブロックが生成されている場合（マクロの場合など）、生成されたブロックの数と実行されたブロックの数が実行回数の前 [S1] に表示されます。

図 3. 動的実行回数の表示

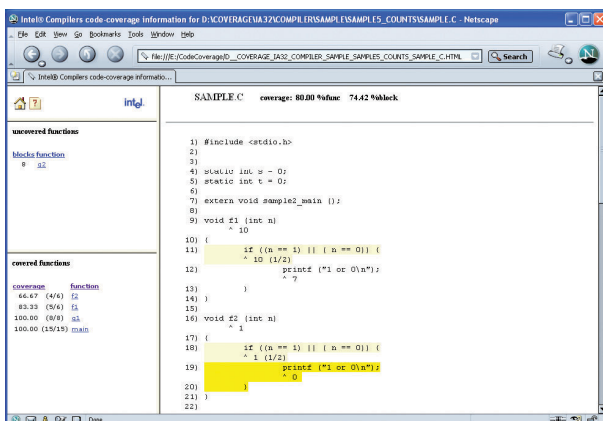


図 3 では、11 行目と 12 行目の下にあるテキストに、以下の情報が含まれています。

- 11 行目の **if** 文は、10 回実行されました。
- 11 行目の **if** 文には、基本ブロックが 2 つ生成されました。
- 2 ブロックのうち 1 ブロックのみが実行されたので、部分的なカバレッジの色が適用されています。

- 変数 **n** の値は、10 回の実行のうち 7 回のみが、0 または 1 です。

1カ所に生成されたすべてのブロックを単一の要素とみなすことが望ましい場合もあります。つまり、1カ所に生成されたブロックのうち少なくとも 1 つが実行された場合にも、すべてのブロックが使用されたときとみなしたい場合があります。これを行うには、**-nopartial** オプションを使用します。すると、カバレッジ判定が無効になり、関連した統計情報が調整されます。図 2 の 11 行目と 12 行目では、12 行目の **print** 文が使用されているが、11 行目の条件のうち 1 つのみが真です。このような場合、**-nopartial** オプションを使えば、部分的に使用されたコード（11 行目のコードなど）を、すべて使用されたものとして扱うように設定できます。

差分カバレッジ

コード・カバレッジ・ツールは、参照実行と新規実行におけるアプリケーションのプロファイルを比較し、新規実行時には使用されたが、参照実行時には使用されなかったコードを識別します。この機能を利用すると、アプリケーション・コードの中で、テスト時には使用されないが、テスト以外で実行された（顧客による実行時など）部分を判別できます。また、アプリケーションのテスト項目に新しくテストを追加する際の、カバレッジへの段階的な影響も把握できます。差分カバレッジ用に参照実行の動的プロファイル情報を指定するには、以下のコマンドのように、**-ref** スイッチを使用します。

```
codecov -prj Project_Name -dpi customer.dpi -ref appTests.dpi
```

差分カバレッジのカバレッジ統計情報には、新規実行時には使用されたが、参照実行時には使用されなかったコードの割合が表示されます。この場合にコード・カバレッジ・ツールに表示されるのは、使用されなかったコードを含むモジュールのみです。ソースビュー内の色については、以下のガイドラインに従って解釈してください。

- いずれの実行でもコードのカバレッジ状態が同じ場合（使用されたか否かにかかわらず）、差分カバレッジのソースビューには、コードは使用されたものとして示されます。
- コードが新規実行時には使用されたが、参照実行時には使用されなかった場合、差分カバレッジのソースビューには、コードは使用されていないものとして示されます。
- コードが新規実行時には使用されなかったが、参照実行時には使用された場合、差分カバレッジのソースビューには、コードは使用されたものとして示されます。

コード・カバレッジ・ツールの実行に必要な環境

アプリケーションに対してコード・カバレッジ・ツールを実行するには、以下の3アイテムが必要です。

- アプリケーション・ソース
- **-Qprof_genx** スイッチ (Windows* の場合) または **-prof_genx** スイッチ (Linux* の場合) を使用してアプリケーションをインストルメント済みバイナリにコンパイルする際に、インテル® コンパイラによって生成される .spi ファイル
- 動的プロファイル情報ファイル (*.dyn) をマージした結果インテル コンパイラの profmerge ツールによって生成される .dpi ファイル。または **-Qprof_use** オプション (Windows の場合) か **-prof_use** オプション (Linux の場合) を使用してアプリケーションをコンパイルする際に、インテル コンパイラによって暗黙的に生成される .dpi ファイル

図 4 に、PGO の利用モデルを示します。

表 2 では、この利用モデルの例について、4 つのステップで説明します。ここでは、仮想のインストルメント済み 32 ビット Windows ベース・アプリケーション myApp.c を使用しています。

インテル コンパイラの profmerge ツールは、アプリケーションを再コンパイルしなくても、.dyn ファイルを .dpi ファイルにマージできます。また、**-a** オプションを使用すると、複数の .dpi ファイルを単一の .dpi ファイルにマージできます。出力 .dpi の名前を選択するには、profmerge の **-prof_dpi** オプションを使用します。

図 4. PGO の利用モデル

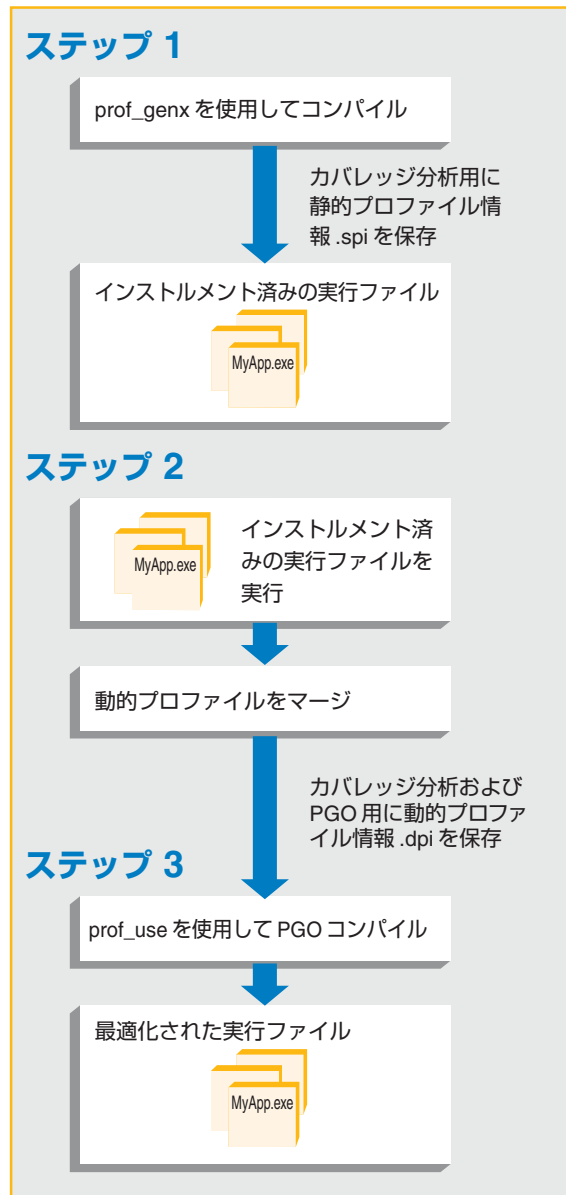


表 2. アプリケーションの動的プロファイル情報の生成

ステップ	コマンド
1. プロファイルを生成するアプリケーションが格納されたディレクトリを宣言します。	<code>set PROF_DIR=c:\myApp\prof_dir</code>
2. プログラムをコンパイルし、インストルメント済みバイナリ myApp.exe と、それに対応する静的プロファイル情報 pgopti.spi を生成します。	<code>icl /Qprof_genx myApp.c</code>
3. インストルメント済みアプリケーションを実行します (各呼び出しがインストルメント済みアプリケーションを実行し、PROF_DIR によって指定されたディレクトリ内に、.dyn 拡張子を持つ新しい動的プロファイル情報ファイルを 1 つ以上生成します)。	<code>myApp.exe</code>
4. すべての .dyn ファイルを、アプリケーションの総合的なプロファイル情報を示す単一の .dpi ファイルにマージし、最適化されたバイナリを生成します (デフォルトの .dpi ファイルの名前は pgopti.dpi)。	<code>profmerge</code>

アプリケーション・ソースの .spi ファイルと .dpi ファイルが利用可能な場合は、以下のコマンドを実行すると、コマンドラインからコード・カバレッジ・ツールを起動できます。

```
codecov -prj Project_Name -spi pgopti.spi -dpi pgopti.dpi
```

-spi オプションと -dpi オプションを使えば、これらのファイルへのパスを指定できます。

コード・カバレッジ・ツールでは、指定された連絡先に電子メッセージを送信するためのリンクを、各 HTML ページの下部に作成できます。このようなリンクを作成するには、以下のオプションを使用してください。

```
codecov -prj Project_Name -mname John_Smith -maddr js@company.com
```

注意：

profmerge ツールは、指定されたディレクトリ内に存在するすべての .dyn ファイルをマージします。関係のない実行によって生成された無関係の .dyn ファイルが残っている可能性もあるので、そのようなファイルがディレクトリ内に存在しないことを確認してください。存在していた場合、無効なプロファイル・データによってプロファイル情報が不正確なものになってしまいます。この結果、不正確なカバレッジ情報が提供され、最適化されたコードのパフォーマンスに悪影響を与える可能性があります。

アプリケーション・モジュールのサブセットに対するカバレッジ分析

コード・カバレッジ・ツールは、アプリケーション内のモジュールのサブセットに対して効率的なカバレッジ分析を行います。**-Qprof_genx** オプションを使用してアプリケーション・モジュールの特定のサブセットのみをコンパイルする場合は、そのモジュールのカバレッジ情報のみが生成されるので、不要なモジュール内でのプロファイル生成によって生じるオーバーヘッドを回避できます。開発者は、アプリケーション全体またはサブセットのプロファイル情報を生成することも、使用されたモジュールを個別のコンポーネントに分割し、コード・カバレッジ・ツールを利用して各コンポーネントのカバレッジ情報を取得することもできます。

対象のモジュールを指定するには、**-comp** オプションを使用します。このオプションには、引数としてファイル名が必要です。ファイルは、以下のコマンドライン例に示すように、分析対象のモジュールやディレクトリの名前が記載されたテキスト・ファイルでなければなりません。

```
codecov -prj Project_Name -comp component1
```

コンポーネント・ファイル内の各行には、モジュール名を 1 つずつ記載します。アプリケーション内のモジュールのうち、フルパス名に含まれたサブストリングがコンポーネント・ファイル内の名前はいずれかに一致したものが、カバレッジ分析の対象として選ばれます。例えば、**component1** ファイル内の行に **mod1.c** と記載されている場合、アプリケーション内でこの名前を持つモジュールがすべて選択されます。特定のモジュールを指定するには、より具体的なパス情報を提供します。例えば、**/cmp1/mod1.c** と記載されている場合は、**cmp1** のディレクトリ内の **mod1.c** のモジュールのみが選択されます。**/cmp2/**のみが記載されている場合は、**cmp2** のディレクトリ内のモジュールがすべて選択されます。コンポーネント・ファイルが指定されていない場合は、**-Qprof_genx** を用いてコンパイルされたすべてのファイルがカバレッジ分析の対象として選ばれます。

表 3 に、コード・カバレッジ・ツールの全オプションを示します。

テスト優先化ツールの紹介

アプリケーション・ソフトウェアに変更が加えられるにつれて、機能テストやパフォーマンス・テストの品質を維持し、最新かつ「的を射た」ものにすることが困難になります。テスト優先化ツールを利用すれば、アプリケーション・プロファイルの変化に応じてアプリケーション・テストの選択と優先順位付けが可能です。

開発者がアプリケーション・モジュールを変更すると、テスト優先化ツールは、変更の影響を受ける可能性が高いテストを指摘します。これは、インテル® コンパイラに搭載された PGO テクノロジーを利用して行われます。テスト優先化ツールは、アプリケーションが過去に実行したプロファイル・データを収集し、アプリケーション・コンポーネントとそれに対応するテストとの依存関係を解明します。この情報を参考に、その後のテストにおいて開発者を支援します。

このツールに最適な用途の 1 つが、アプリケーションのコード・カバレッジに基づいた効果的な階層テストの作成です。例えば、このツールを利用すれば、アプリケーション・テストの中で、テスト項目全体と全く同じコード・カバレッジを達成する最小サブセットを特定できます。

表 3. インテル®コンパイラ コード・カバレッジ・ツールのオプション

オプション	説明	デフォルト
-help	コード・カバレッジ・ツールのオプションをすべて出力します。	
-spi <file>	静的プロファイル情報ファイル .spi のパス名を設定します。	pgopti.spi
-dpi <file>	動的プロファイル情報ファイル .dpi のパス名を設定します。	pgopti.dpi
-prj	プロジェクト名を設定します。	
-counts	動的実行回数を生成します。	
-nopartial	部分的に使用されたコードを、完全に使用されたコードと同じものとして扱います。	
-comp	対象ファイルのリストが含まれたファイルの名前を設定します。	
-ref	ref_dpi_file との差分カバレッジを検索します。	
-demang	C++ 関数名とその引数の両方を明確にします。	
-mname	Web ページ所有者の名前を設定します。	
-maddr	Web ページ所有者の電子メールアドレスを設定します。	
-bcolor	使用されていないブロックについて、HTML の色名や色分けを設定します。	#ffff99
-fcolor	使用されていない関数について、HTML の色名や色分けを設定します。	#ffcccc
-pcolor	部分的に使用されたコードについて、HTML の色名や色分けを設定します。	#fafad2
-ccolor	使用されたコードについて、HTML の色名や色分けを設定します。	#ffffff
-ucolor	不明なコードについて、HTML の色名や色分けを設定します。	#ffffff

テスト優先化ツールは、テストのターンアラウンド・タイムも大幅に削減できます。このツールを使えば、長い時間をかけて大量のエラーを見付ける代わりに、変更に伴う退行に関連したエラーを判別可能な、少数のテストを迅速に特定できます。このツールによる時間の節約と品質の確保はとりわけ、テストがプロジェクトの重要な時期に行われている、追い込まれたバグ修正作業で効果を発揮します。開発者はこのツールを使うと、アプリケーションのサブセットで特定のカバレッジ・レベルを達成するのに必要なテスト数を削減できます。また、各テストの所要時間がわかれば、特定のコード・カバレッジ・レベルを最小限の時間で達成可能なテストを選択し、優先順位付けが行えます。

テスト優先化ツールは、Windows* または Linux* オペレーティング環境上で IA-32、インテル® EM64T、およびインテル® Itanium® プロセッサ・ファミリを運用しているシステムに対応します。C、C++、Fortran をシームレスにサポートします。

テスト優先化ツールの実行に必要な環境

アプリケーションのテストに対してテスト優先化ツールを実行するには、以下のアイテムが必要です。

- **-Qprof_genx** スイッチ (Windows* の場合) または **-prof_genx** スイッチ (Linux* の場合) を使用してアプリケーションをインストルメント済みバイナリにコンパイルする際に、インテル® コンパイラによって生成される .spi ファイル。
- 各アプリケーション・テストの動的プロファイル情報ファイル (*.dyn) をマージした結果インテル® コンパイラの profmerge ツールによって生成される .dpi ファイル。個々のテスト用に生成されたすべての *.dyn ファイルに profmerge ツールを適用し、結果として生成された .dpi ファイルに対して、各テストを一意に識別可能な方法で名前を付ける必要があります。profmerge ツールは、指定されたディレクトリ内に存在するすべての .dyn ファイルをマージします。関係のない実行によって生成された無関係の .dyn ファイルが残っている可能性もあるので、そのようなファイルがディレクトリ内に存在しないことを確認してください。存在していた場合、無効なプロファイル・データによってプロファイル情報が不正確なものになってしまいます。この結果、不正確なカバレッジ情報が提供されることとなります。

- 優先順位付けするテストのリストを用意し、個々のテストを一意に識別可能な方法で、.dpi ファイルに名前を付ける必要があります。該当するすべての .dpi ファイルの名前を、dpi-list ファイル（テスト優先化ツールに提供されるテキスト・ファイル）内で指定してください。dpi-list ファイル内の各行には、dpi 名を 1 つずつ記載します。この名前の後にはオプションで、対応するテストの所要時間を **dd:hh:mm:ss** 形式で続けることができます。所要時間はオプションですが、指定しない場合、所要時間短縮を目的にした優先順位付けの要求は拒否さ

れます。ただし、テスト数を減らすための優先順位付けは可能です。

図 5 に、テスト優先化ツールの利用モデルを示します。

表 4 では、仮想の 32 ビット Windows ベース・アプリケーション myApp.c を使って、テスト優先化ツールの使用方法について説明します。

表 4. テスト優先化ツールの使用方法

説明	コマンド
アプリケーションが格納されたディレクトリを宣言します。	set PROF_DIR=c:\myApp\prof_dir
プログラムをコンパイルし、インストルメント済みバイナリ myApp.exe と、それに対応する静的プロファイル情報 pgopti.spi を生成します。	icl /Qprof_genx myApp.c
不要な .dyn ファイルが存在しないことを確認します。	rm PROF_DIR *.dyn
インストルメント済みアプリケーションを実行し、PROF_DIR によって指定されたディレクトリ内に、.dyn 拡張子を持つ新しい動的プロファイル情報ファイルを 1 つ以上生成します。	myApp.exe < data1
profmerge ツールによって、すべての .dyn ファイルを、Test1 におけるアプリケーションの総合的なプロファイル情報を示す単一のファイル (Test1.dpi) にマージします。	profmerge -prof_dpi Test1.dpi
不要な .dyn ファイルが存在しないことを確認します。	rm PROF_DIR *.dyn
インストルメント済みアプリケーションを実行し、PROF_DIR によって指定されたディレクトリ内に、.dyn 拡張子を持つ新しい動的プロファイル情報ファイルを 1 つ以上生成します。	myApp.exe < data2
profmerge ツールによって、すべての .dyn ファイルを、Test2 におけるアプリケーションの総合的なプロファイル情報を示す単一のファイル (Test2.dpi) にマージします。	profmerge -prof_dpi Test2.dpi
不要な .dyn ファイルが存在しないことを確認します。	rm PROF_DIR *.dyn
インストルメント済みアプリケーションを実行し、PROF_DIR によって指定されたディレクトリ内に、.dyn 拡張子を持つ新しい動的プロファイル情報ファイルを 1 つ以上生成します。	myApp.exe < data3
profmerge ツールによって、すべての .dyn ファイルを、Test2 におけるアプリケーションの総合的なプロファイル情報を示す単一のファイル (Test3.dpi) にマージします。	profmerge -prof_dpi Test3.dpi
次の 3 行からなる、tests_list という名前のファイルを作成します。1 行目には Test1.dpi、2 行目には Test2.dpi、3 行目には Test3.dpi を記載します。各アイテムの準備が整うと、PROF_DIR ディレクトリ内からコマンドラインでテスト優先化ツールを起動できます。	tselect -dpi_list tests_list -spi pgopti.spi

注意： .spi ファイルへのパスは、**-spi** オプションを使って指定します。

図 5. テスト優先化ツールの利用モデル

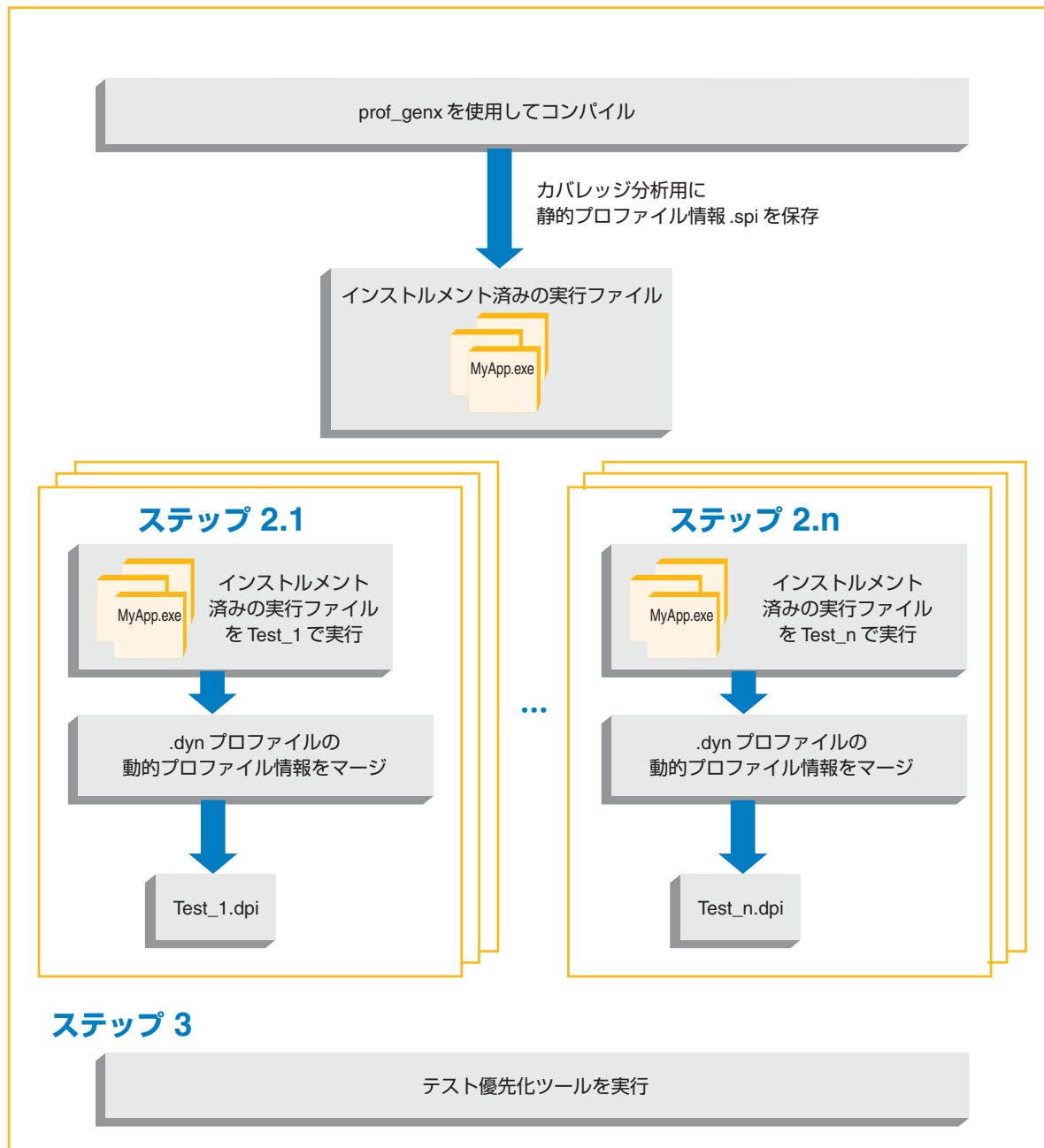


表 5 に、テスト優先化ツールからの出力例を示します。

表 5. テスト優先化ツールからの出力例

番号	カバレッジ率 (%)	ブロック・カバレッジ (%)	関数カバレッジ (%)	オプションのテスト名
1	87.50	45.65	37.50	Test3.dpi
2	100.00	52.17	50.00	Test2.dpi

テストの総数 = 3

合計ブロック・カバレッジ ~ 52.17

合計関数カバレッジ ~ 50.00

この例では、テスト優先化ツールから以下の情報が提供されています。

- 3つのテストをすべて実行すると、ブロック・カバレッジは 52.17%、関数カバレッジは 50.00% になります。
- Test3 だけでアプリケーションの基本ブロックの 45.65% が使用されており、これは、3つのテストで得られた合計ブロック・カバレッジの 87.50% に相当します。
- Test2 を追加すると、累計ブロック・カバレッジは 52.17% になり、これは Test1、Test2、Test3 の合計ブロック・カバレッジの 100% に相当します。
- Test1 を省いても、合計ブロック・カバレッジに悪影響はありません。

次に、テストの所要時間が以下のように tests_list ファイルに記載されているとします。

```
Test1.dpi      00:00:60:35
Test2.dpi      00:00:10:15
Test3.dpi      00:00:30:45
```

次のコマンドを実行すると、所要時間の短縮が図られます。

```
tselect -dpi_list tests_list -spi pgopti.spi -
mintime
```

このコマンドの実行結果が、表 6 に示された出力例です。

表 6. インテル® コンパイラ テスト優先化ツールからの出力例

番号	所要時間	カバレッジ率 (%)	ブロック・カバレッジ (%)	関数カバレッジ (%)	オプションのテスト名
1	10:15	75.00	39.13	25.00	Test2.dpi
2	41:00	100.00	52.17	50.00	Test3.dpi

テストの総数 = 3

合計ブロック・カバレッジ ~ 52.17

合計関数カバレッジ ~ 50.00

合計所要時間 = 1:41:35

この結果によれば、すべてのテストを連続して実行すると 1 時間 41 分 35 秒かかりますが、テストを選択して実行するとわずか 41 分で同じ合計ブロック・カバレッジを達成できます。優先順位付けが時間短縮に基づく場合 (Test2 の次に Test3 を実行) と、テスト数削減に基づく場合 (Test3 の次に Test2 を実行) では、テストの順序が異なります。表 6 の例では、Test2 が所要時間当たりで最も高いカバレッジを達成しています。そのため、最初に行うべきテストとして Test2 が選択されています。

次のコマンドラインに示されたように **-cutoff** オプションを使用すると、一定レベルの基本ブロック・カバレッジに達した時点でテスト優先化ツールが終了するように設定できます。

```
tselect -dpi_list tests_list -spi pgopti.spi -
cutoff 85.00
```

この例の cutoff 値を使ってツールを実行した場合、Test3 のブロック・カバレッジは 45.65% に達しているため、Test3 のみが選択されます。これは、3 つのテストで得られる合計ブロック・カバレッジの 87.50% に相当します。テスト優先化ツールはまず、すべてのプロファイル情報をマージすることによって、全テストの合計カバレッジを計算します。時間を節約する場合は、**-nototal** オプションを使用すると、この計算を省略できます。**-nototal** オプション使用時には、合計カバレッジが不明になるため、カバレッジ率の代わりに絶対的なカバレッジ情報のみが報告されます。

テスト優先化ツールには、表 7 に示されたオプションが用意されています。

表 7. インテル® コンパイラ テスト優先化ツールのオプション

オプション	説明	デフォルト
-help	テスト優先化ツールのオプションをすべて出力します。	
-spi <file>	静的プロファイル情報ファイル .spi のパス名を設定します。	pgopti.spi
-dpi_list <file>	動的プロファイル情報ファイル .dpi の名前が記載されたファイルのパス名を設定します。ファイル内の各行には .dpi 名を 1 つずつ記載し、オプションで所要時間を続けることができます。名前は、各テストを一意に識別可能なものにする必要があります。	
-o <file>	出力レポートファイルのパス名を設定します。	
-comp	対象ファイルリストが含まれたファイルの名前を設定します。	
-cutoff <value>	累計ブロック・カバレッジがあらかじめ計算された合計カバレッジの <value>% に達した時点でツールが終了するように設定します。<value> には、0.0 より大きく、100 を超えない値を指定してください。	
-nototal	ツールが合計カバレッジをあらかじめ計算しないように設定します。	
-mintime	テストの所要時間を最小限に短縮します。このオプションを機能させるには、dpi_list ファイル内のテスト名に続けて、同一行中にテストごとの所要時間を dd:hh:mm:ss 形式で記載してください。	
-verbose	プログラムの進行状況に関する詳細なログ情報を生成します。	

まとめ

インテル® コンパイラに新しく追加された2つの設計ツールは、以下のようにしてソフトウェアの品質を高めることを目的にしています。

コード・カバレッジ・ツール

コード・カバレッジ・ツールは、Windows* および Linux* 向けのインテル® C++ コンパイラとインテル® Fortran コンパイラに付属しています。このツールを利用すると、ソフトウェア開発者は、特定のワークロードがアプリケーションが実行したときに使用されるアプリケーション・コードの量を把握できます。分析結果は、HTML ページに表示されます。色分けされ、コメントが付けられたソースコードのリストによって、コードの使用状況を分類し、詳細な情報が得られます。ソースコード内での移動は、フレーム形式の採用により容易に行えます。また、アプリケーションのファイルや関数を並べ替えて、最も使用されていないモジュールや関数を簡単に判別できます。

コード・カバレッジ・ツールは、さまざまなメリットをソフトウェア開発者に提供します。第一に、テストされたコードの量を示すと、コードの全体的な品質を評価できます。第二に、コード・カバレッジ情報には、特定のワークロードがアプリケーションの主要コードをどの程度使用しているかが示されます。インテル® コンパイラの PGO とインテル® アーキテクチャのパフォーマンスの両方を十分に活かすには、パフォーマンスにとって重要なモジュールのカバレッジを高めることが欠かせません。第三に、アプリケーションの基本ブロックごとに動的実行回数を取得できるオプションを提供しており、カバレッジとパフォーマンスの両方のチューニングに役立ちます。最後に、2つの異なるアプリケーションの実行プロファイルを比較できるので、継続的なアプリケーションのサポートが容易です。このようにして使用すると、アプリケーション・コードの中で、アプリケーション・テストでは使用されないが、ユーザによるアプリケーション実行時には使用される部分を判別できます。

テスト優先化ツール

テスト優先化ツールは、Windows および Linux 向けのインテル C++ コンパイラとインテル Fortran コンパイラに付属しています。このツールを利用すれば、アプリケーション・プロファイルの変化に応じてアプリケーション・テストの選択と優先順位付けが可能です。

開発者がアプリケーション・モジュール内のコードを変更すると、テスト優先化ツールは、変更の影響を受ける可能性が高いテストを指摘することによって、テストスイートを最新状態に保てるようにします。

このツールは、プロジェクトのクリティカル・パスで行われることが多いテスト工程で、複数の方法によりテスト時間の削減が可能です。第一に、アプリケーション・テストの中で、テストセット全体とまったく同じコード・カバレッジを達成する最小サブセットを特定できます。また、長い時間をかけて大量のエラーを見付ける代わりに、少数のテストを迅速に実行して、変更に伴う退行に関連したエラーを判別できます。このツールを使うと、アプリケーションのサブセットで特定のカバレッジ・レベルを達成するのに必要なテスト数を削減できます。テストに割ける時間が極めて限られている場合、各テストの所要時間がわかれば、希望するコード・カバレッジ・レベルを最小限の時間で達成可能なテストを選択し、優先順位付けも行えます。

参考情報

トレーニングおよびサポートの詳細については、以下のサイトを参照してください。

インテル® ソフトウェア・カレッジ

<http://www.intel.com/software/college> (英語)

インテル® プレミア・サポート

<https://premier.intel.com/> (英語)

製品情報と購入方法については、www.intel.com/software/products/geo/jpn を参照してください。

インテル、Intel、Intel ロゴ、Itanium は、アメリカ合衆国およびその他の国における Intel Corporation またはその子会社の商標または登録商標です。

* その他の社名、製品名などは、一般に各社の商標または登録商標です。

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスを許諾するためのものではありません。

製品に付属の売買契約書「Intel's Terms and conditions of Sales」に規定されている場合を除き、インテルはいかなる責を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証（特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他、知的所有権を侵害していないことへの保証を含む）に関して一切責任を負わないものとします。

インテル製品は、医療、救命、延命措置などの目的への使用を前提としたものではありません。

インテル製品は、予告なく仕様が変更される場合があります。

インテル株式会社

〒300-2635 茨城県つくば市東光台 5-6

<http://www.intel.co.jp/>

© 2005, Intel Corporation. 無断での引用、転載を禁じます。
2005年 6月