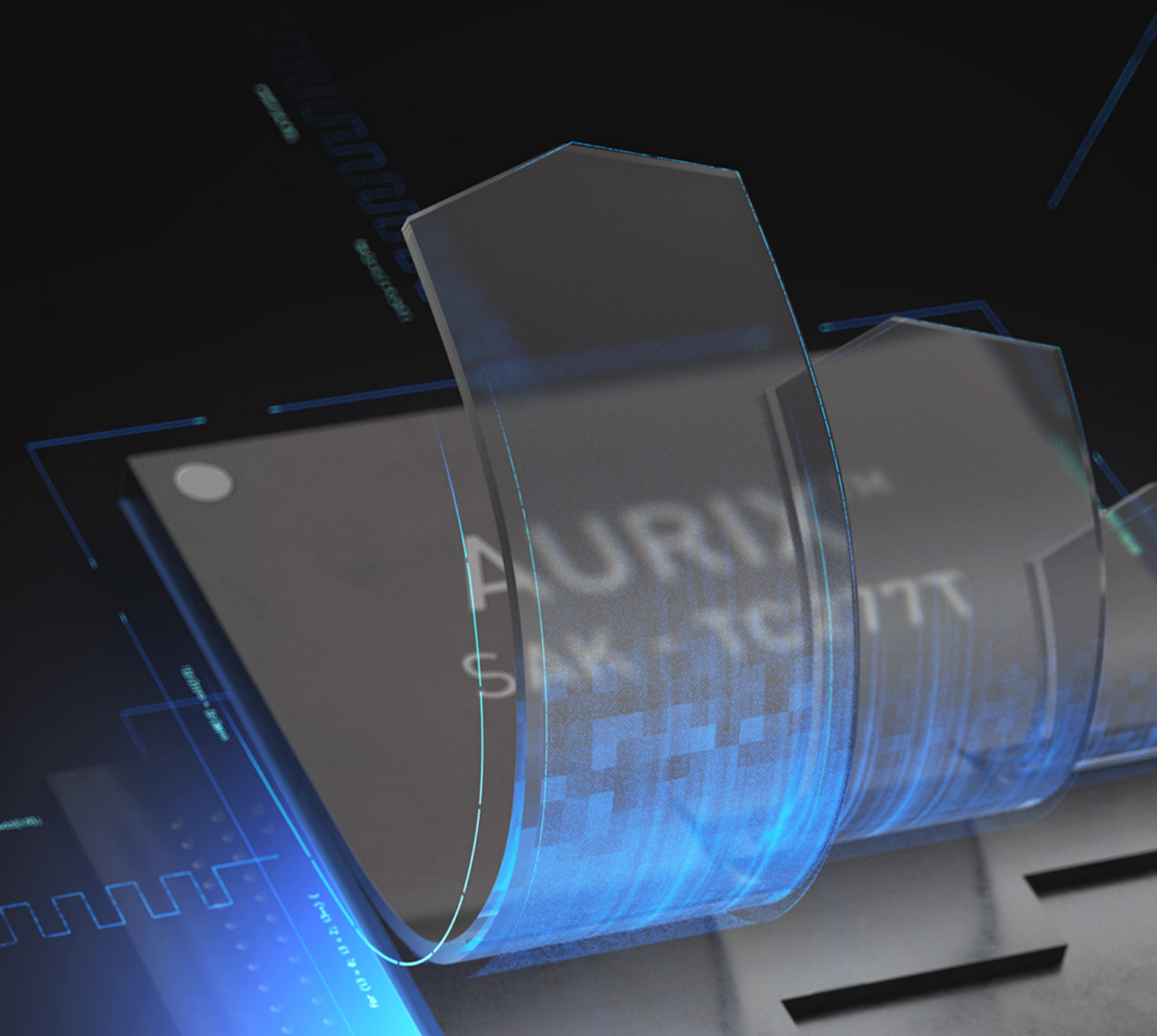
TASKING®

Embedded Profiler
製品概要



いままでにないスマートなプロファイリングテクノロジー - 従来の測定型プロファイリングより短時間でボトルネックを解消

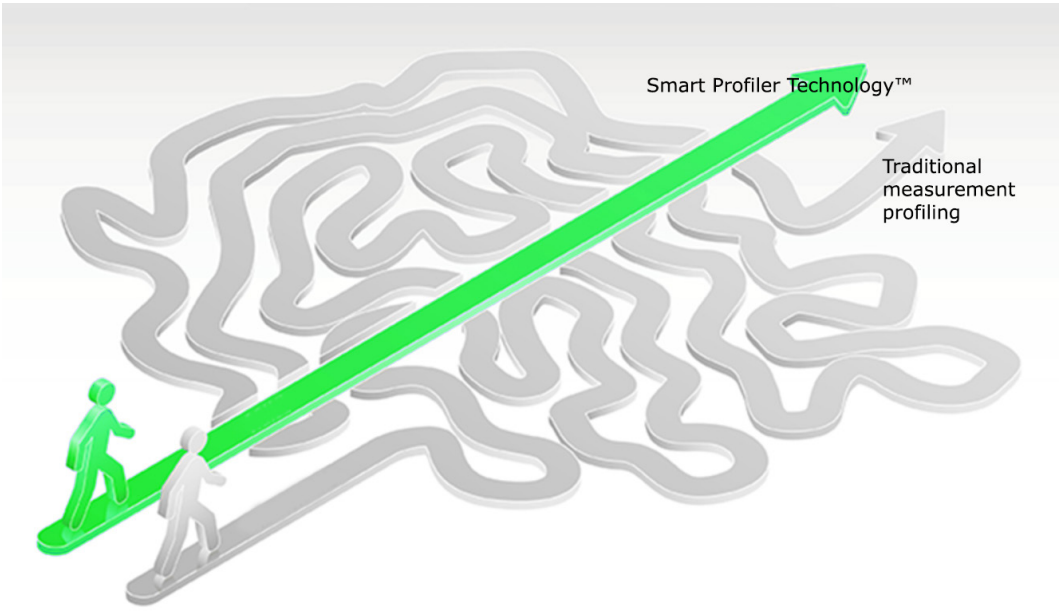
TASKINGエンベデッドプロファイラー ― 概要

従来、プロファイリングは関数の実行時間の測定に関連付けられてきました。関数の実行にかかる具体的なクロックサイクル数や実行時間（ミリ秒）を知ることは、関数自体の改善可能性を特定する方法としては、それほど有効ではありません。特定のターゲットデバイスで実際に関数の実行速度を上げるために、コードまたは設定をどのように変更する必要があるのかを突き止める方がより早く改善の可能性を特定することができます。

関数の実行時間だけを測定する既存のプロファイラーとは異なり、**TASKINGエンベデッドプロファイラー**は、AURIXの内部動作に関する専門的なデータが組み込まれているため、以下の利点があります。

- AURIX内部のハードウェアリソースの使用法が間違っているためにコアの処理時間を大量に浪費している関数やコード行を特定します
- これらのパフォーマンス・ボトルネックとなるの原因を明らかにします
- 最小限の作業で迅速にボトルネックを解決するための具体的な改善案を提供します

AURIXボードに接続したUSBケーブルまたは低価格なminiWigglerを使用し、コードのランタイム動作を負荷をかけずに短時間解析するだけで、**TASKINGエンベデッドプロファイラー**は、最大のパフォーマンス低下を引き起こしているソースコード行と設定、およびパフォーマンス低下の原因、さらに特定の問題を解決するために必要な作業を正確に特定します。問題を修正した後で、修正の前に測定した関数実行時間と比較して、修正によるパフォーマンスの改善を評価することができます。



新しいマイクロコントローラでのコード実行に対して、このスマート・プロファイリングテクノロジーを適用すると、（計算内容を変更せずに）コードを改善するための軽微な変更を即座に発見できるだけでなく、改善されたコードが、新しいマイクロコントローラ特有のハードウェアリソースを変更前に比べより最適な方法で利用するため、多くの場合、数時間の作業でパフォーマンスを大幅に向上することができます。

スマートプロファイリングテクノロジー

- AURIXの内部動作に関する専門的なデータを搭載
  - メモリシステムの遅延
  - キャッシュ動作
  - 分岐の予測
  - クロック周波数
  - その他
- アプリケーションによる特定のハードウェアリソースの不適切な使用を、負荷をかけずに短時間で解析。
- 最大のパフォーマンス低下を引き起こしている原因（ソース行および理由）を、わかりやすくグラフィカルに表示。
- 問題を迅速に修正するための具体的な改善案を原因別に提示。
- 変更前後の関数およびアプリケーションの実行時間を比較し、改善による実際のパフォーマンス向上を記録。
- 試行回数を削減し、数時間でアプリケーションパフォーマンスが大幅に向上。
- アプリケーション全体またはユーザーが選択した一部の関数に対してパフォーマンスボトルネックを解析し、重要性が高い順に表示 ー 最重要問題に優先的に対応可能。
- エキスパートでなくても簡単に操作可能: TASKINGエンベデッド・プロファイラーを起動し、デバイスに接続して測定した後、パフォーマンス改善提案に沿って作業を進めことができる。**高コストなプローブや特別なハードウェアに関する専門知識・準備は不要。**
- サポートされているターゲットコアで稼働するアプリケーション（コンパイラツールセットおよびプログラミング言語に関係なく）のすべてで使用できるスタンド・アロンツール。

ここからは、具体的な例を使用してTASKINGエンベデッドプロファイラーの標準的ワークフローについて解説し、各機能の詳細を説明します。

*ソースレベルの情報は、DWARF 3互換のツールチェーンでコンパイルされたCコードに限定されます。それ以外については、アセンブリレベルの情報が提供されます。

TASKINGエンベデッドプロファイラー ― 詳細

ストール - ハードウェアパフォーマンスの問題を引き起こす原因

ストールは、マイクロコントローラによる有益な処理（プログラムの実行等）が妨げられた場合に発生します。その場合、マイクロコントローラはストールを解消するためにパワーと時間を費やします。

例えば、ほとんどのマルチコアマイクロコントローラでは、メモリアクセス速度が一定ではありません。特定のコアは、特定のメモリに対してその他のコアよりも速くアクセスできます。アクセスに時間のかかるメモリに保存されたデータに対して、特定のコアが頻繁にアクセスする場合、このコアは大部分の時間を有益な計算ではなくデータの待機に費やすことになり、ストールが発生します。

それぞれのマイクロコントローラによって、数多くのストールの種類があり、これらのストールはユーザーには見えません。ストールがいくつ発生していても、プログラムはエラーや警告を表示することなく同じ結果が生成されます。

アプリケーションに認識されないストールが多種多様に存在するため、開発者に知識がない場合、成熟したアプリケーションであっても、50%以上の時間がストールに消費されることがあります。

通常、コンパイラ設定、プログラムコード、メモリレイアウトのいずれかをわずかに変更するだけで（変数を低速アクセスメモリから高速アクセスメモリに移動するなど）、大部分のストールを解消できるので、以前と同じ結果を算出しながら、何倍も高速にプログラムを実行できます。しかし、コンパイラ設定、プログラムコード、またはメモリレイアウトを不適切な方法で変更した場合、多数のストールが新たに引き起こされるおそれがあります。

従来型のプロファイリング

関数の実行時間だけを測定する従来型のプロファイラーを使用する場合、どのような変更がストール数を減らすのかについては推測に委ねられ、おそらくは、特定のハードウェアに関する長年の経験と勘に頼ることになります。期待が持てそうな変更を実施した前後で実行時間を比較し、長い時間をかけて試行錯誤すれば、いくつかの改善を見つけられるかもしれません。

スマートプロファイリング

TASKINGエンベデッドプロファイラーは、関数の実行時間だけでなく、大部分のストールを引き起こしている関数をグラフィカルにまとめて表示するので、最も大きくパフォーマンスを低下させている問題を、推測に頼ることなく容易に特定し、速く対処することができます。各関数を掘り下げて行くと、問題の原因となるソース行とアセンブリ命令を見つけることができます。

ハードウェアや各種ストールの原因について、ユーザーが詳しい知識を持っている必要はありません。このスマートプロファイラーは、ハードウェアリソースを有効活用するために、アプリケーションの既知のストール動作に関する情報と測定結果を比較します。この参考情報を使用して、ストール率の高い関数とソース行を評価し、これらの具体的な問題を修正するための改善案を提示します。改善案は、測定されたストールと、ターゲット別のストール発生原因に関する深い知識を組み合わせて導き出されます。この知識はスマートプロファイラー自体にコード化されています。

スマートプロファイラーのデータ収集機能は、負荷をかけずにハードウェアトレースを使用して、ターゲットのマイクロコントローラー内でストールを引き起こすプログラム命令に関する統計情報を集めます（デバッグ情報を使用して、アセンブラ命令からソース行に変換）。

TASKINGエンベデッドプロファイラーのワークフロー例

ここからは、具体的な例でスマートプロファイラーの動作について確認していきます。
一般に、ワークフローは以下のパターンに従います。

1. ターゲットアプリケーションをコンパイルしてフラッシュします。
2. スマートプロファイラーを使用して問題を解析し、新しい改善点を見つけます。
3. アプリケーションを修正します。
4. コンパイル、フラッシュ、プロファイリングを再実行し、結果を比較して改善されたかどうかを検証します。
5. アプリケーションで満足できる結果が出るまで、ステップ1から繰り返します。

1. ターゲットアプリケーションのコンパイルおよびフラッシュ

ここでは、以下の非常にシンプルなアプリケーションについて考察します。

```
#define ARRAY_SIZE  (16 * 1024)
volatile int x[ARRAY_SIZE];

int main(void)
{
       clock_t clockstart = clock();
for (int i = 0; i < ARRAY_SIZE; ++i){ x[i] = 1; }
       int duration = (int) (clock() - clockstart);
       printf("duration %i ticks\n", duration);
}
```

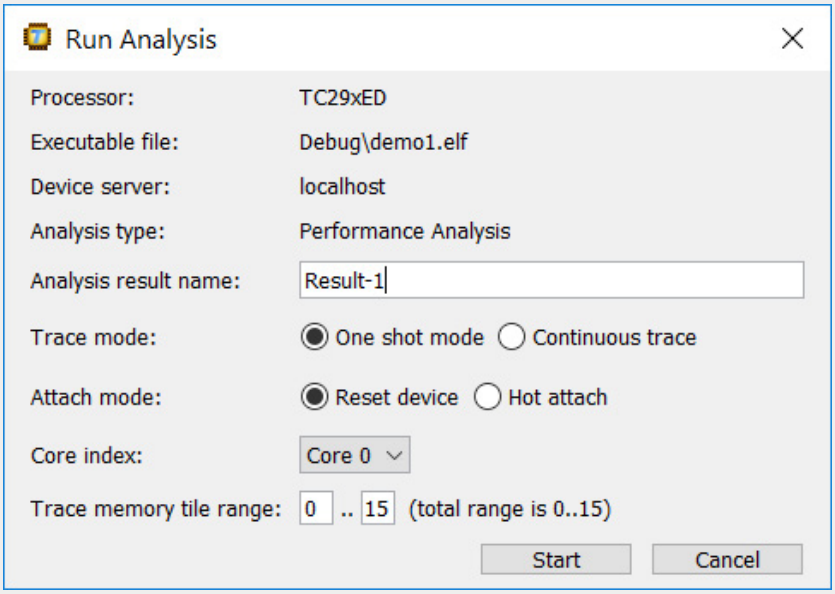
目的はパフォーマンス測定なので、はじめに、使用できる範囲で最大限に最適化したアプリケーションをコンパイルし、ターゲットデバイスにフラッシュする必要があります。アプリケーションを実行すると、従来型のプロファイラーを使用した場合と同様の情報が提供されます。例えば、clock()を使用したforループ計算の実行時間は、73754 tickになります。このアプリケーションで修正すべき個所はあまりありません。

2. スマートプロファイリング

TASKINGエンベデッドプロファイラーは、スタンドアロンアプリケーションです。このため、アプリケーションデータ（ターゲットアプリケーションおよびソースを含むフォルダへのリンク）とスマートプロファイリングの結果を格納するプロジェクトを作成する必要があります。ここでは簡単な設定をプロファイリング開始前に1回のみ行います。

プロジェクトの作成後、解析を使用して最適化する対象を選択できます。スマートプロファイラーでは、調査している問題の種類に応じていくつかの解析方法をサポートしています。通常は [Performance Analysis] から始めます。この解析タイプでは、最大の問題についての概要が提供されます。[+] ボタンをクリックし、ウィザードに従って新規の [Performance Analysis] を作成します。プロジェクトツリーの左側でこれを選択し、[Play] ボタンを押すと、解析が1回実行されます（図1を参照）。[Play] ボタンを押した後に表示される各種の解析設定オプションについては、ボックス内で説明しています。今回は [Start] ボタンを押して、そのままの設定で解析を開始します。

解析の設定



[Trace Mode]：トレースメモリがいっぱいになるまでトレースを続行するか、トレース途中にトレースメモリからアップロードします（トレースメモリの読み取り中に、トレース対象のアプリケーションが短時間停止する場合があります）。

[Attach Mode]：測定前にデバイスをリセットするか、現在のシステム状態を維持します。

[Core Index]：測定対象のコアを指定します（現在、一度に解析できるのは、1コアで実行されるコードのみです）。

[Trace memory tile range]：どの範囲のトレースメモリをトレースに使用するかを指定します（一部のトレースメモリを別のアプリケーションに割り当てることができます）。

図*1:* ウィザードを使用して、簡単に新しいパフォーマンス解析が作成可能です。

数秒後、ハードウェアから解析データが収集されて結果が計算されると、図2の右側に示すように解析サマリーが表示されます。

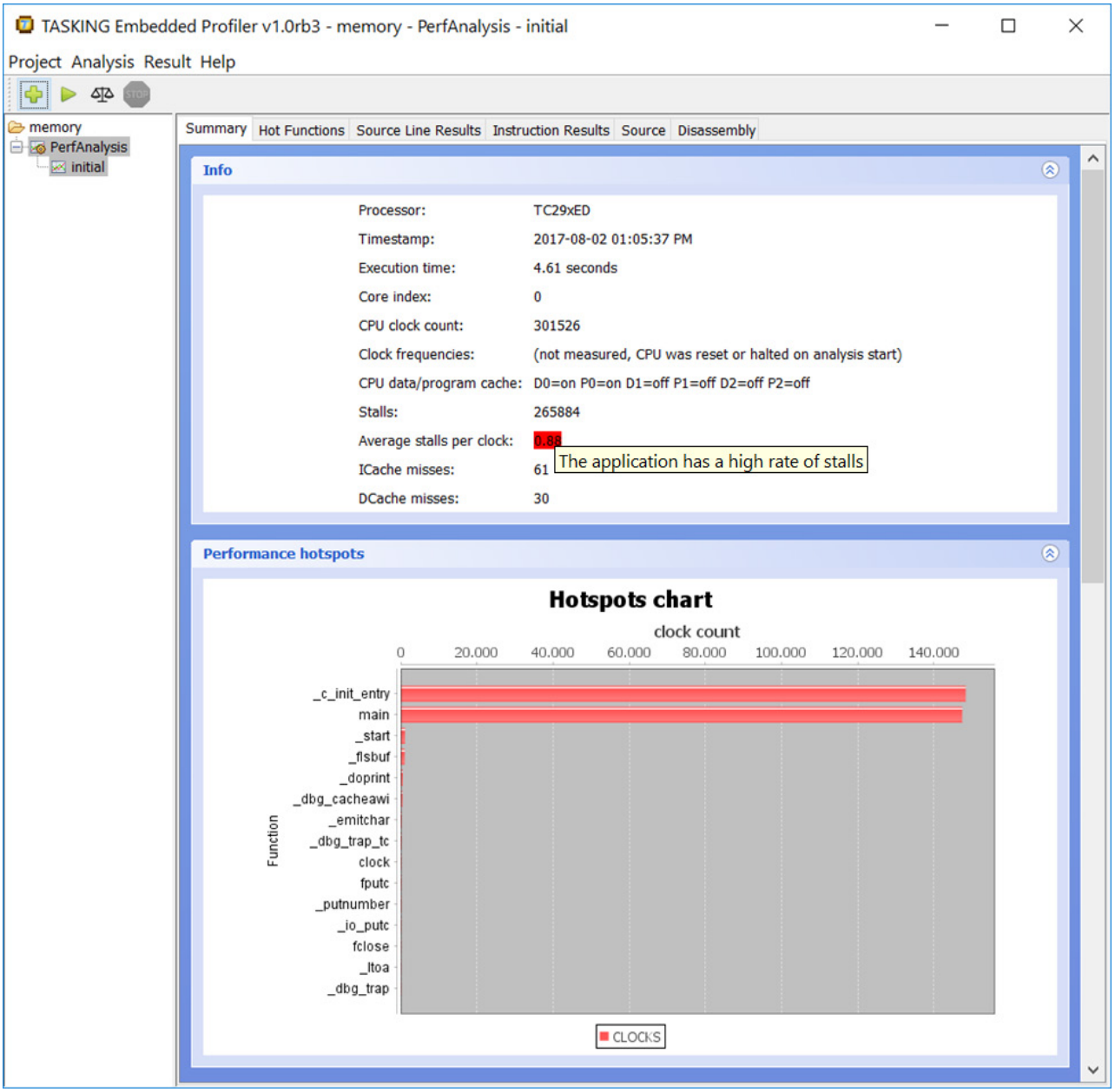


図2: 左側のプロジェクトツリーから、以前に設定した解析とその結果にアクセスできます。右側にはパフォーマンス解析のサマリーが表示されます。この例では、*main*関数で異常に多数のストールが発生しており、*CPU*時間の*88%*が浪費されています。グラフ上で *[main]* をクリックすると、*[Hotspots chart]* に示すように、関数レベルの結果を掘り下げて行くことができます。

サマリー情報から、アプリケーション実行時間の88%がストールに費やされていることが分かります（主要なパフォーマンスインジケーターは赤くハイライトされています）。これらのストールを解消した場合、現在の301kクロックが約36kクロックになり、おおよそ8倍のスピードで処理が終了します。

[Hotspots chart] には、プログラム実行中に最も多くの時間が費やされる関数が表示されます。これらの関数を修正するとパフォーマンスが最も向上するので、最初に修正することをお勧めします。このグラフで [main] の隣にある赤い棒をダブルクリックすると、図3に示すように、main関数のソースと解析結果を表示したビューが開きます。

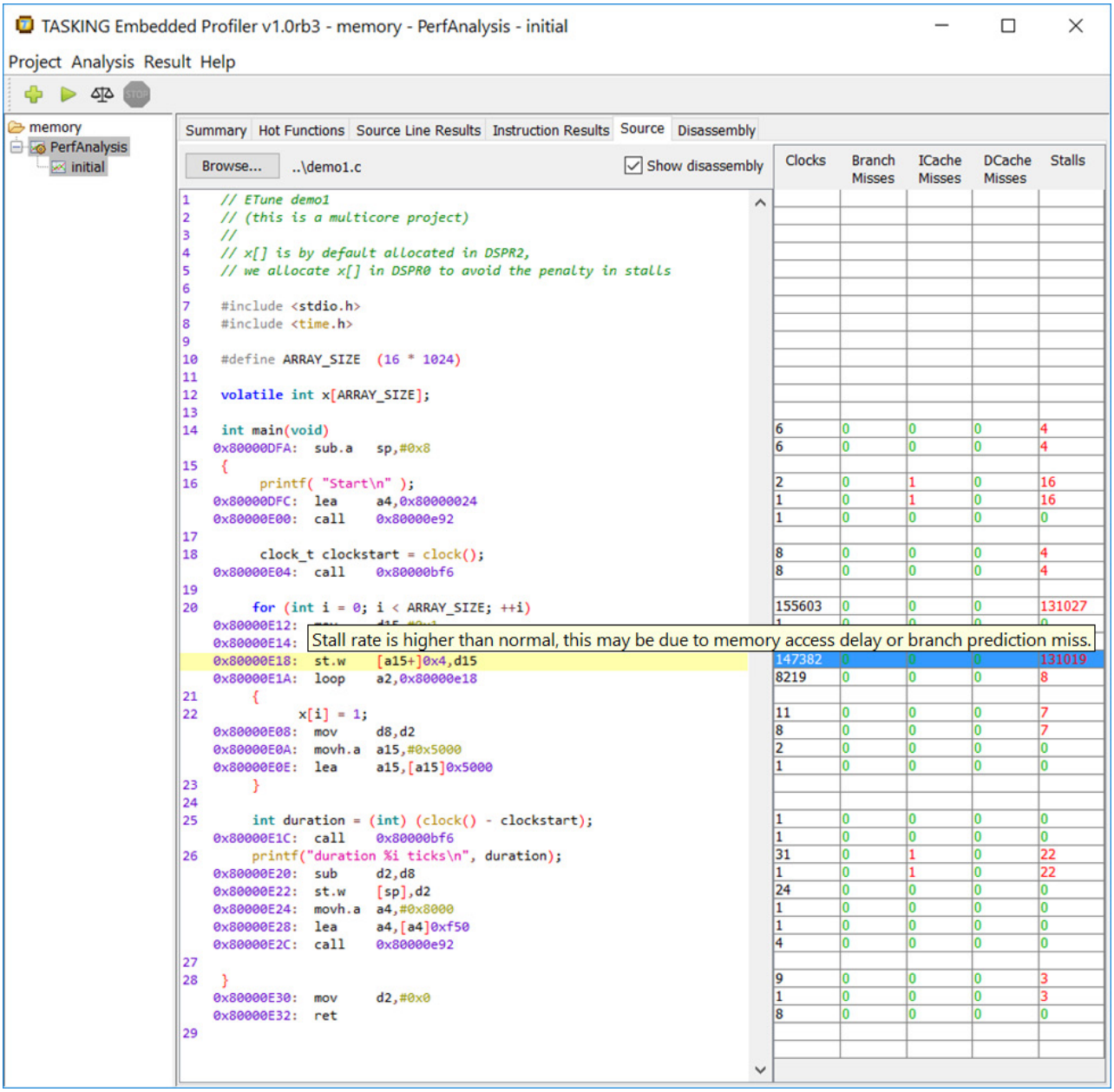


図3: *main*関数に対する関数レベルの解析結果右側の結果テーブルの *[stall]* 列を見ると、*22*行目の配列*x*への書き込みにより、過剰なストールが引き起こされていることが分かります（この命令に対して実行されたクロックあたりのストール数が*1*に近い）。

ソースビューの右側のテーブルには、対応するソース行またはアセンブリ命令に対して測定されたストール数が表示されます。このテーブルを使用すると、図3の20行目にあるforループで多数の一般ストールが発生していることが簡単に分かります（1クロックサイクルあたりのストール数が1に近い）。ストールを引き起こしているアセンブリ命令（黄色のハイライト）は、配列xを格納しているメモリに定数1を書き込んでおり、原因はメモリのサブシステムにあります。

[Performance Analysis] を使用することで、最大のパフォーマンス低下箇所を即座に特定し、その根本原因がメモリシステムにあることを突き止めました。メモリシステム内の原因を正確に特定し、問題の改善策を見つけるために、[Memory Analysis] を実行します。[+] ボタンを使用して [Memory Analysis] を作成し、デフォルトパラメーターを使用して実行すると、図4に示すように正確なストール原因が表示されます。

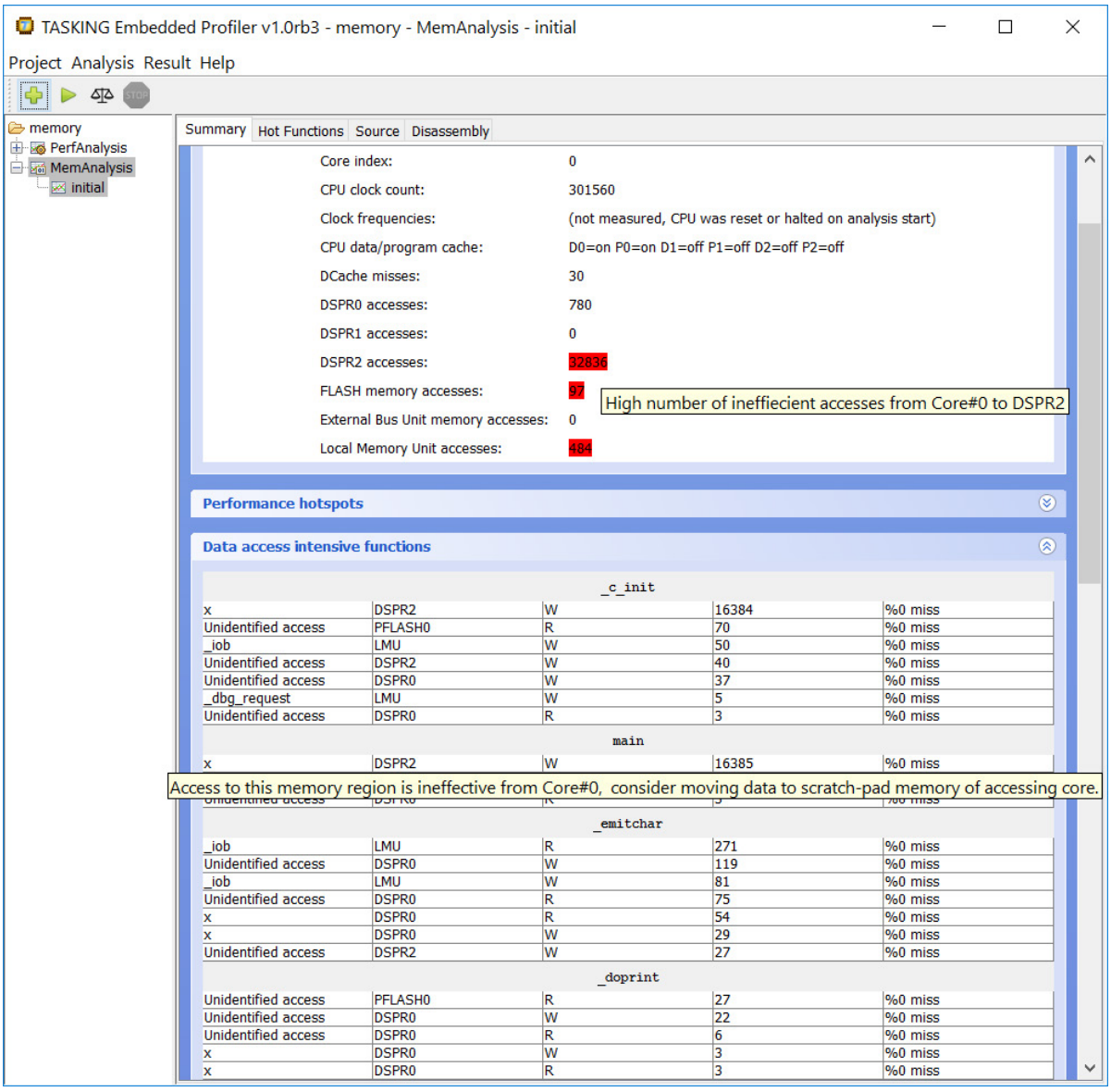


図*4:* コード例はコア*0*で実行されています。赤色でハイライトされた *[DSPR2 accesse] KPI*が主な問題を示しています。この*KPI*にカーソルを合わせると、コア*2*のコアローカルメモリである*DSPR2*に対して、コア*0*からのアクセスが多すぎるというメッセージが表示されます（コア*0*自体のメモリである*DSPR0*へのアクセスの方が大幅に高速）。*[Data access intensive functions]* テーブル内に*main*関数が表示されており、*main*内で最も非効率なアクセスを含む行にカーソルを合わせると、グローバル配列*x*へのアクセスが原因であることが示されます。この問題の改善策として、この*x*をコア*0*のスクラッチパッドメモリである*DSPR0*に移動するように提案するメッセージがツールチップに表示されます。

3. アプリケーションの修正

図2および図3に示したように、簡単な2回の測定とスマートプロファイラーによる解析結果を使用して、最大のパフォーマンスの問題に対する根本原因を特定したので、アプリケーションの改善を容易に実行できます。TASKINGツールセットを使用すると、lslリンカースクリプトを変更するか、グローバル配列の宣言を変更することで、図4に示したプロファイラーの提案に従って、変数xをDSPR0に移動することができます。

```
volatile __private0 int x[ARRAY_SIZE];
```

4. コンパイル、フラッシュ、プロファイリングの再実行

例2の小さい変更を含むアプリケーションのコンパイルとフラッシュを再実行した後で、問題が修正されたことを確認し、新しい最適化の機会を見つけるために、パフォーマンス解析をもう一度実行します。

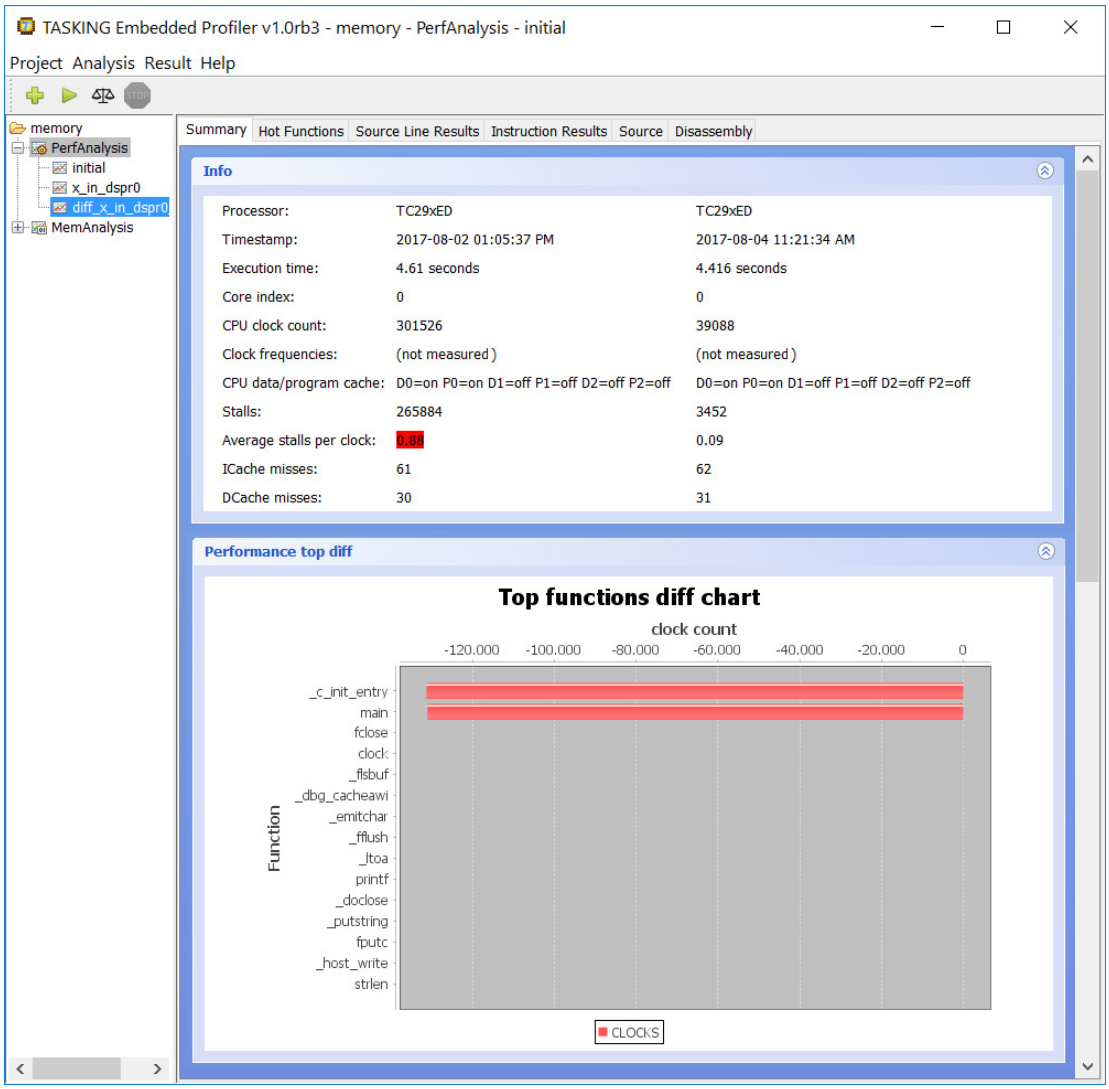


図5: 解析結果の違いがツールによって計算されるので、変更前後の結果を容易に比較できます。ここでは、*[initial]* 解析の結果と *xをDSPR0*に移動した後に実施した *[x_in_dspr0]* 解析の結果を表示しています。

図5に示した相違点から、改善が非常に有効であったことが分かります。改善されたアプリケーションは、元のコアと同じ結果を約1/8のクロックサイクルで計算しています。つまり、パフォーマンスが約8倍に向上しています。
1クロックあたりの平均ストール数が90%弱から約10%に減少しています。この変更による新しいストールは発生していません。

5.さらなる改善

最初の改善による効果が検証された後、2回目のパフォーマンス解析の結果を調査することで、次に大きいアプリケーションパフォーマンスの問題に対処できます。該当するパフォーマンスの問題をすべて解決するまで、この手順全体を繰り返すことができます。通常、1回の繰り返しにかかる時間は10 ～ 20分で、ツールには重要度が高い順に問題が表示されるので、手順を数回繰り返すことで、大幅なパフォーマンス向上を達成できます。

TASKINGエンベデッドプロファイラーは、この例で調査したメモリの問題に加えて、ハードウェアを介して明らかになるその他のストールをすべて表示します。このため、スマートプロファイラーに表示される問題を解消した後は、ハードウェアが最大限活用されることに確信を持てます。

パフォーマンスのリグレッション

アプリケーション、ドライバー、OS内でストールによって引き起こされた主なパフォーマンス低下を解決した後は、これらのコンポーネントへの変更を監視する必要があります。スマートプロファイラーのコマンドラインインターフェースから、以前に設定した解析を自動的に再実行し、結果（または差異）をCSVファイルにエクスポートすることができます。ビルドまたは開発プロセスの一部として、これまでのパフォーマンス統計と現在のパフォーマンスを比較しておくことで、不要なハードウェア・ストールによりパフォーマンスに悪影響を与えるソフトウェア・コンポーネントの変更を容易に特定して解決できます。

導入効果

以下のテーブルでは、従来型のプロファイリングとTASKINGによるスマートプロファイリングを比較した場合のコスト削減を（比較可能な結果を得るまでの時間、必要なハードウェア、ソフトウェアライセンスの観点から）確認します。

従来型のプロファイリング	スマートプロファイリング	相対的なコスト
代表的なパワートレインプロジェクトでは、リンカーレイアウトの最適化だけで、1か月で約2人のエキスパート・エンジニアが必要になります。	エキスパートではなくても、数時間で最大の問題を特定して解決できます。	> 20:1
既存のトレースツールでは、ハードウェアから「すべてのデータ」を取得するためには高価なハードウェアプローブが必要です。データから適切な情報を引き出すためには、大量のポストプロセスと専門知識が必要です。	TASKINGのスマートプロファイラーは、miniWiggler（約100ユーロ）またはUSBケーブルでハードウェアに接続できます。ポストプロセスが自動実行され、適切なデータのみが抽出されます。	> 10:1
試行錯誤で最適化を探しても、良い結果が得られるとは限らず、最悪の場合、何週間もかけても全体的な改善が見られない可能性もあります。	スマートプロファイリングでは、最も重要な問題に最初に対処でき、原因も表示されるので、何が問題なのかを推測する必要はなく、該当する問題の修正だけに時間を費やすことができます。	> 2:1
トレース用デバッガーソフトウェアのコスト	スマートプロファイラーソフトウェア	> 2:1

テーブル1: ROI

TASKINGエンベデッドプロファイラーの使用シナリオ

スマートプロファイリングツールの標準的なユースケースを以下に示します。

サプライヤー提供ソフトウェアの品質保証

サプライヤーから提供されたソフトウェアのパフォーマンス特性に対する自動監視および検証。サプライヤーが開発した関数に対して、最大ストール率、最大ジッター、最大実行時間などのパフォーマンスしきい値を定義しておき、提供されたコードでこれらの基準に対する違反を自動的に検出することで、最良のソフトウェア品質を保証します。

可能な限り最良のコードを顧客に提供する目的で、サプライヤー自身によってもこのツールは使用されています。

EMBEDDED PROFILER

関数開発中に予測可能な実際のタイミング（アプリケーション）

リアルタイムアプリケーションにとって、関数のタイミングに信頼性と予測可能性があることは不可欠です。ソフトウェア関数の開発プロセスの早い段階でスマートプロファイリングを利用すると、コードに隠れた予想外の厄介な問題を解消して、開発中に確認されたタイミングとコード導入時のタイミングを近づけることができます。

最適化（OS、ドライバー、ライブラリ、アプリケーション）

新機能の開発、既存機能の統合、新規ターゲットハードウェアへのアプリケーションの移植などでは、気づかないうちに新しいハードウェアストールが引き起こされていることがあります。コアの使用率やタイミング動作において受け入れがたい問題が明らかになった時点で、トラブルシューティング担当者にプロジェクトを引き渡すのではなく、早い段階でスマートプロファイリングを使用することで、ソフトウェアパフォーマンスをコントロールすることができます。より短期間で高品質なコードを提供し、コードのパフォーマンスを継続的に管理できます。

コード変更後のリグレッションテスト（全コード）

僅かなコード変更でも、ストールやその他のパフォーマンス低下を大量に引き起こす可能性があります。コード変更を行う前に自動化されたストールリグレッションテストを実行することで、コードのパフォーマンス特性の変化を常に意識し、何らかの問題が発生する前に誤りを特定することができます。

タイミングとパフォーマンスの問題のトラブルシューティング

プロジェクトの最終段階になって、エンドツーエンドのタイミングがプロジェクト要件に違反しているか、一部のコアの使用率が過剰に高いことに気づいたとします。修正を目的としたOSスケジュールの変更はリスクが極めて高く、一種の連鎖効果によってその他のイベントチェーンのタイミングが変わるおそれがあります。タスクを1つでも20%高速化できれば、すべてはうまくいくでしょう。スマートプロファイリングを使用した特定のタスクの調査をまだ実行していない場合、スマートプロファイリングを使用することで、1時間以内にターゲットタスクまたはランナブルの実行時間を50%以上短縮できる可能性が十分にあります。TASKINGエンベデッドプロファイラーのターゲットには、アプリケーション全体またはユーザーが選択した一部の関数を指定できます。

概要

TASKINGエンベデッドプロファイラーは、AURIXのような深く組み込まれたデバイスに対して**スマートプロファイリング**を可能にします。これまで、ハードウェアはブラックボックスとして扱われており、解明にはエキスパート、時間、ハードウェア、ソフトウェアへの多大な投資が必要でしたが、このツールを使用することでハードウェアをユーザーがコントロールできるようになります。

無償の評価版の登録を行うと、TASKINGエンベデッドプロファイラーをすぐに使用できます。

TASKING®

www.tasking.com