インテル® VTune™ パフォーマンス・アナライザ 8.0 Linux* 版 入門ガイド

資料番号: 309564-001J

インテル® VTune™ パフォーマンス・アナライザは、コードのパフォーマンスに関する情報を提供します。チューニングの焦点を置くべき箇所を示し、短時間で最高のパフォーマンスを引き出します。

本ガイドの目的

VTune アナライザの基本的な機能を紹介します。

本ガイドでは、VTune アナライザを使用してコードを解析し、最高のパフォーマンスを実現するためにはどこを最適化すべきか理解することを目的としています。

はじめに

本ガイドでは、gsexample2a アプリケーションのチューニングをとおして、パフォーマンス・チューニングを段階ごとに説明します。

- パフォーマンス問題の特定
- 問題解決のためのコード修正
- コードの修正前と修正後のパフォーマンス比較

目次

表記規則

本ガイドで使用される表記規則は次のとおりです。

スタイル	定義
`Code`	入力する文字を表します。
`code`	コマンドまたはディレクトリを表します。
用語	新しい用語または概念を表し、同じページのボックスにその説明が記載されています。

1. アプリケーションのビルド

最初に、VTune アナライザが有用なデータを収集できるような設定でアプリケーションをビルドする必要があります。コンパイラの設定による製品レベルの最適化とシンボル情報を有効にしてアプリケーションをビルドすると、(リリース時のように) 完全に最適化され、シンボル情報が含まれたコードが生成されます。これにより、VTune アナライザは最も有用なデータを収集することができます。

本ガイドで使用する例では、GNU* C コンパイラ (バージョン 3.2.3) を '`-g -O2`' オプションで使用しています。

注意:

- システムに C/C++ 開発パッケージがインストールされていることを確認してください。
- システムの構成やコンパイラのバージョンによって、結果は異なります。

アプリケーションをビルドするには、次の操作を行います。

1. `cd` コマンドを使用して、`opt/intel/vtune/samples/gsexample` ディレクトリに移動します。
2. `make` と入力します。
   または、インテル® コンパイラを使用している場合は、次のコマンドを入力します。
   `make CC=icc CFLAGS="-g –O2"`

デフォルトでは、`gsexample` ディレクトリは `/opt/intel/vtune/samples` ディレクトリ以下にあります。VTune アナライザを別の場所にインストールした場合、パスは異なります。

2. コードの解析

アプリケーションのビルド後、インストール・ディレクトリ内の `/samples/gsexample` ディレクトリにあるコードを使用して、パフォーマンスの解析を行います。**[First Use Wizard (簡易ウィザード)]** とそのビューを使用して、データの収集、結果の表示、ソースコードの特定の問題箇所へのズームインを行います。

2 a. VTune アナライザの設定

このステップでは、VTune アナライザを設定して、アクティビティを作成し実行します。*アクティビティ* は、パフォーマンス・データを収集して、VTune アナライザのプロジェクトに保存します。後でこのデータを使用して解析の続きを行ったり、修正前と修正後のアプリケーションの実行速度を確認したりできます。

アクティビティは、指定されたアプリケーションの特定のデータを収集します。

VTune アナライザは、データコレクタを使用して異なるタイプのパフォーマンス・データを収集します。VTune アナライザでは、データコレクタとデータ収集の設定を行うための 3 つのウィザードが用意されています。

アプリケーションのパフォーマンスをチューニングする前に、元のパフォーマンスの *ベンチマーク* を作成してください。

ベンチマークは、将来修正を比較する場合の基準となるように、計測可能かつ再現可能でなければなりません。

1. 次のコマンドを入力して `gsexample2a` アプリケーションを実行します。

```
#. /gsexample2a datafile.txt 100000
```

説明

`datafile.txt` はデータファイルで、`100000` はループの反復回数です。

2. 時間を記録します。これは、アプリケーションのチューニングにおける現段階のベンチマークです。アプリケーションの実行後、次のようにコマンドライン・コンソールで経過時間を確認できます。

```
****************************************
* Elapsed time was 11 seconds
* File character count: 1859075
* Total characters counted: 1223906272
* CRC calculated: 0xcf
****************************************
```

次に、**[First Use Wizard (簡易ウィザード)]** を使用して、アプリケーション実行のパフォーマンス・チューニングを行います。このウィザードは、システム全体に渡るパフォーマンス・データを干渉しないように収集し、アプリケーションにおける最もアクティブな 5 つの関数の基本データを提供します。

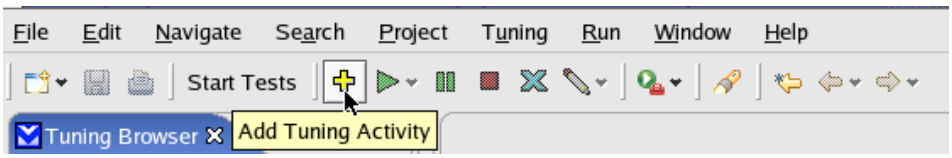
1. コマンドラインで、VTune アナライザのインストール・ディレクトリに移動し、`vtlec` を実行して VTune アナライザを起動します。次のコマンドを入力してください。

   ```
   # /opt/intel/vtune/bin/vtlec
   ```

   VTune アナライザが起動して、スタートアップ画面が表示されます。

2. スタートアップ画面の次に表示される **[Select A Workspace (ワークスペースの選択)]** ダイアログ・ボックスで、使用するワークスペースの名前または番号を入力するか、**[OK]** をクリックしてデフォルトの設定を使用します。

3. VTune アナライザのツールバーで、**[Add Tuning Activity (チューニング・アクティビティの追加)]** をクリックして、**[Select A Wizard (ウィザードの選択)]** ダイアログ・ボックスを開きます。



   a. **[First Use Wizard (簡易ウィザード)]** () を選択して、**[Next (次へ)]** ボタンをクリックします。

   b. **[First Use Wizard (簡易ウィザード)]** ダイアログ・ボックスで、次の操作を行います。

      a. **[Application to Launch (起動するアプリケーション)]** フィールドで **[Browse (参照)]** をクリックして、`gsexample2a` アプリケーションを参照します。

      b. 表示されたダイアログ・ボックスで次の操作を行います。

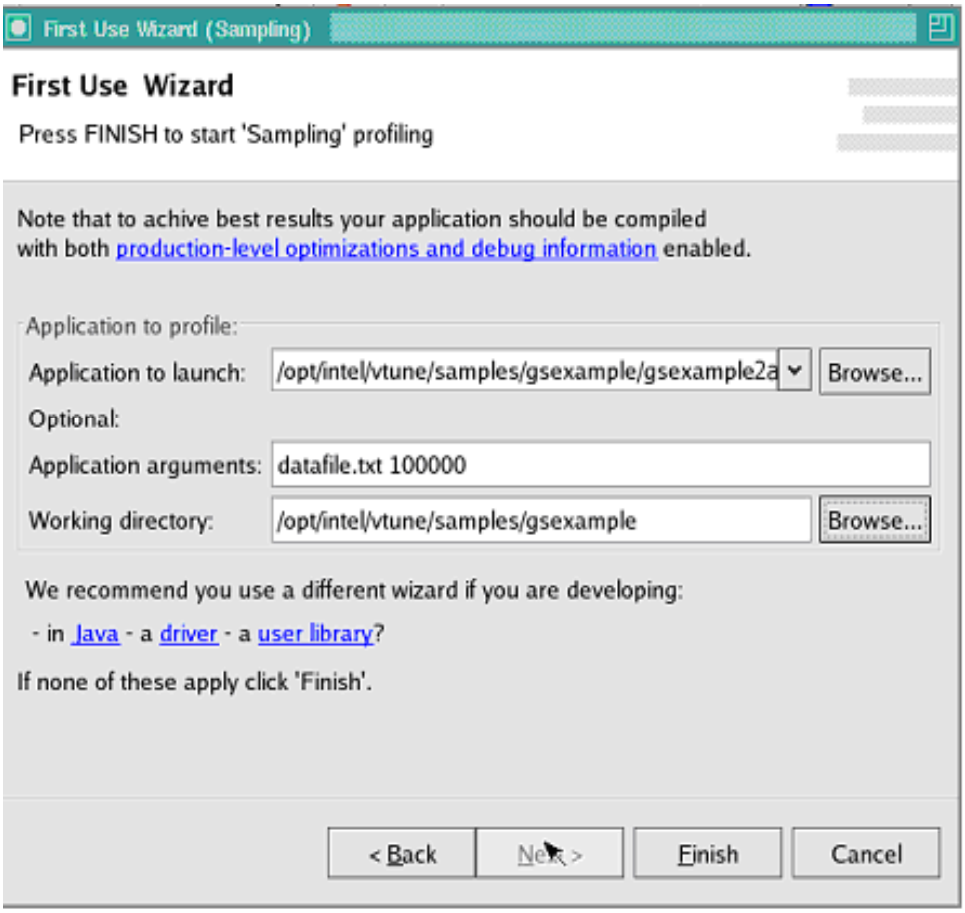
         - **[Folders (フォルダ)]** ペインで `/opt/intel/vtune/samples/gsexample` を選択します。
         - **[Files (ファイル)]** ペインで `gsexample2a` ファイルを選択します。
         - **[OK]** をクリックします。

      c. **[Application Arguments (アプリケーションの引数)]** フィールドに、データファイルの名前と反復回数を入力します。

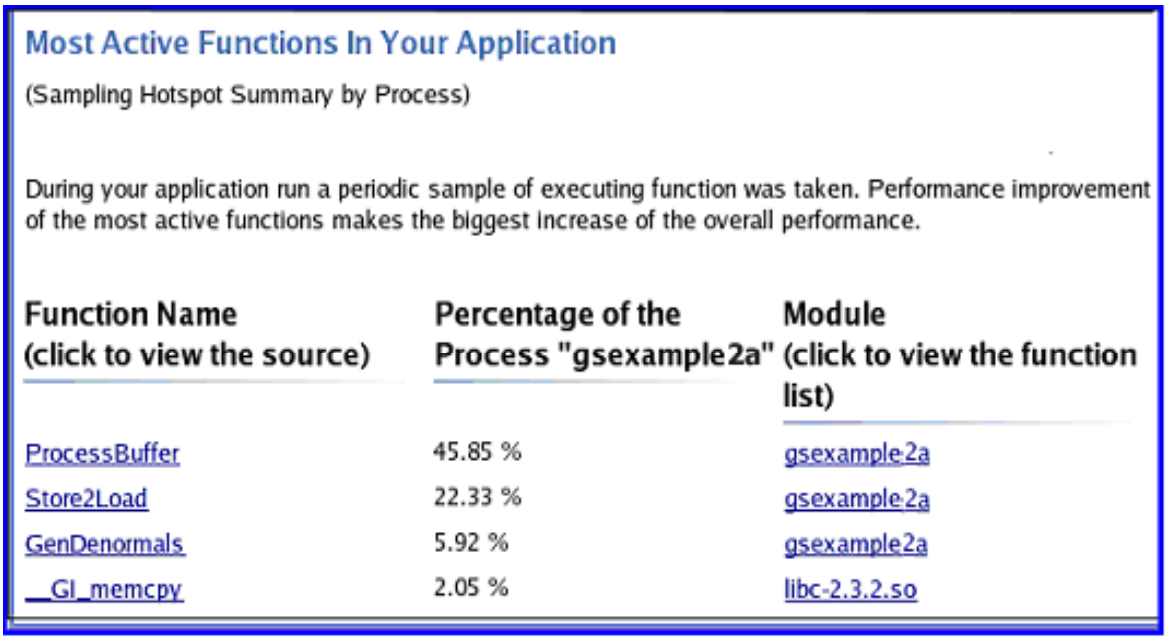
         例: `datafile.txt 100000`

d. **[Working Directory (作業ディレクトリ)]** で、**[Browse (参照)]** をクリックして `gsexample` ディレクトリを参照します。

次のようになります。



4. デフォルトの設定を使用して `gsexample2a` アプリケーションを解析するアクティビティを作成し実行するには、**[Finish (完了)]** をクリックします。完了後、次のような画面が表示されます。



[Sampling Summary (サンプリング・サマリ)] は、データ収集時にシステム内で最もアクティブだった 5 つの関数に関するデータを提供します。このビューでは、アプリケーション全体に対する関数の割合がパーセントで表示されます。さらに、関数をクリックして関数のソースビューを表示したり、関数が属するモジュールをクリックしてそのモジュール内の関数一覧を表示することもできます。

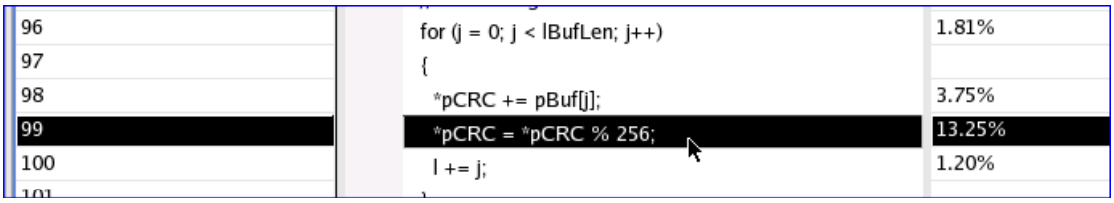
[First Use Wizard (簡易ウィザード)] を使用してアクティビティを作成し実行する場合、Clockticks (Clocktick の回数) イベントのデータが収集されます。

2 b. 結果の表示

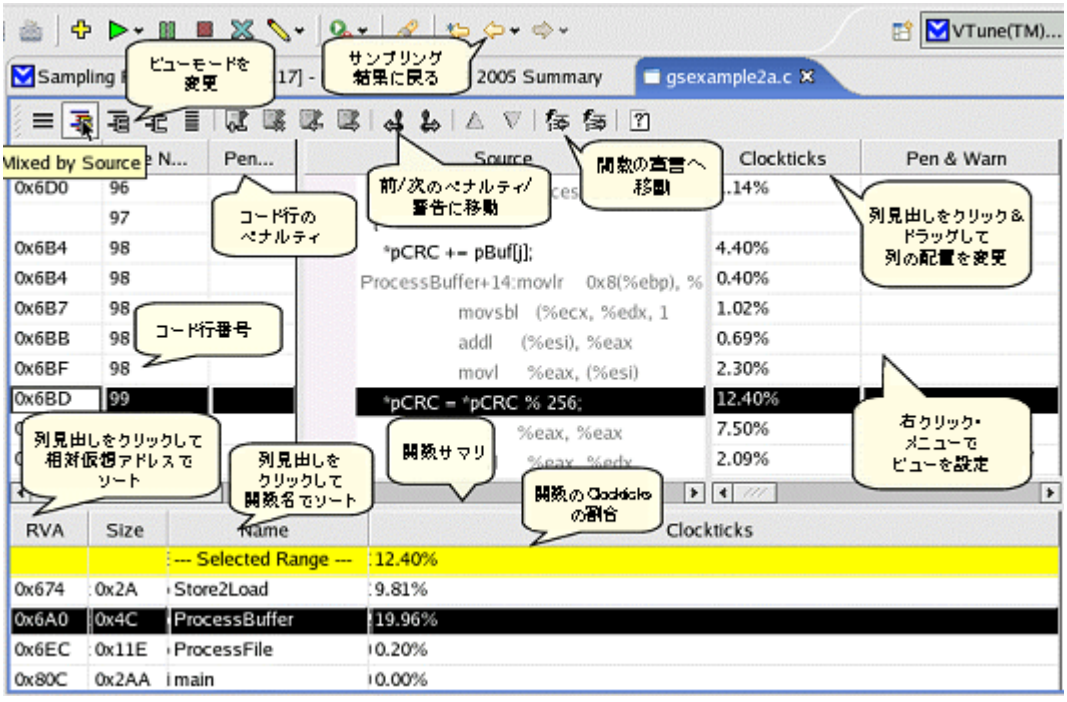
表示されている最初の関数は `ProcessBuffer` です。この関数に注目して、パフォーマンスを改善できるかどうか試してみましょう。

1. `ProcessBuffer` をクリックして、ソースコードを表示します。
2. **[Source (ソース)]** ビューを変更して、重要な情報を見やすくします。
   - **[Source (ソース)]** ビューで右クリックして、ドロップダウン・メニューから **[View Events as (イベントを表示)] → [% of Activity (% のアクティビティ)]** を選択します。

コードをスクロールして、最も時間を費やしている行を見つけます。この例では、99 行目に対して最も高い Clockticks (Clocktick の回数) の割合が報告されています。



この機会に、**[Source (ソース)]** ビューのその他の機能について知っておくと良いでしょう。次の図は、**[Mixed by Source (ソース行順による混合)]** を使用して表示する場合に、**[Source (ソース)]** ビューで利用可能な機能の説明です。**[Source (ソース)]** ビューのモードを変更するには、ツールバーの **[Mixed (混合)]** モード・アイコン  を使用します。



2 c. コードの修正

ステップ 2b で行ったデータの解析により、サンプル・アプリケーションにおいて ProcessBuffer 関数が最も多くの時間を費やしており、そのコードの 99 行目における Clockticks (Clocktick の回数) が最多であることがわかりました。この問題の主な原因は、メモリアクセスと不要なポインタ参照による遅延にあります。各反復時に pBuf 引数を読み込み、*pCRC パラメータをロードおよびストアしています。しかし、pBuf と *pCRC の間には依存関係がないため (実際のパラメータである ProcessBuffer – buf と iCRC は同じメモリを参照しないため)、冗長なストアを削除し、*pCRC 変数を 99 行目のループの外に移動することができます。*pCRC パラメータをローカル変数にロードすることにより、ポインタ値のロードとその値によって指定されたアドレスにあるメモリのロードが不要になり、メモリ参照を削減できます。

次の手順に従ってコードを修正するか、または /samples/gsexample/ ディレクトリにある修正済みコード (gsexample2c.c ファイル) を使用してください。

修正前のコードを次に示します。

```
long ProcessBuffer(char* pBuf, long lBufLen, int* pCRC)
{
  int j;
  long l = lBufLen;
  // parse buffer
  // calculating modulo 256
  for (j = 0; j < lBufLen; j++)
  {
        *pCRC += pBuf[j];
        *pCRC = *pCRC % 256;
        l += j;
  }
#if MYDEBUG
  printf("...lBuflen = %ld\n", lBufLen);
#endif
  return l / 2;
}
```

gsexample2a.c ファイルのコードを修正する場合の手順を次に示します。

1. アプリケーション開発に使用しているエディタでファイルを開きます。
2. ProcessBuffer 関数のソースを表示します。
3. INTEGER 型の新しいローカル変数 iChkSum を作成して、ループの前に *pCRC の値で初期化し、合計を累計するのに使用します。
4. ループの後に iChkSum を pCRC へ格納します。これにより、コンパイラは iChkSum にレジスタを割り当て、メモリ操作を減らすことができます。さらに、コンパイラは修正後のループに対してより積極的な最適化を実行することもできるようになります。この修正は、pCRC と pBuf に依存関係がないことを前提としている点に注意してください。
5. 修正後のコードを次に示します。

```
long ProcessBuffer(char* pBuf, long lBufLen, int* pCRC)
{
  int j, iChkSum = *pCRC;
  long l = lBufLen;
  // parse buffer
```

```
        // calculating modulo 256
        for (j = 0; j < lBufLen; j++)
        {
              iChkSum += pBuf[j];
              iChkSum = iChkSum % 256;
              l += j;
        }
    #if MYDEBUG
        printf("...lBuflen = %ld\n", lBufLen);
    #endif
        *pCRC = iChkSum;
        return l / 2;
    }
```

6. 次の設定でアプリケーションを再ビルドします。

```
cc -g -O2 gsexample2c.c -o gsexample2c
```

2 d. コードの修正前と修正後のパフォーマンス比較

ステップ 2c では、元のコードよりもパフォーマンスが向上するようにコードを修正しました。しかし、場合によっては、コードを修正することにより 1 つの問題が解決しても、また新たな問題が引き起こされることがあります。このため、修正されたコードによって、実際にパフォーマンスが改善されたかどうかを確認する必要があります。

このステップでは、コードの修正前と修正後のパフォーマンスを比較します。

1. 新しいバージョンのアプリケーションを生成するために、(元のアプリケーションの生成時と) 同じスイッチを使用して、修正後のアプリケーションをコンパイルします。または、インストール・ディレクトリの `/samples/gsexample` フォルダにあるバージョンを使用します。デフォルトのインストール・ディレクトリは、`/opt/intel/vtune` です。
2. 次のコマンドを入力して `gsexample2a` アプリケーションを実行します。

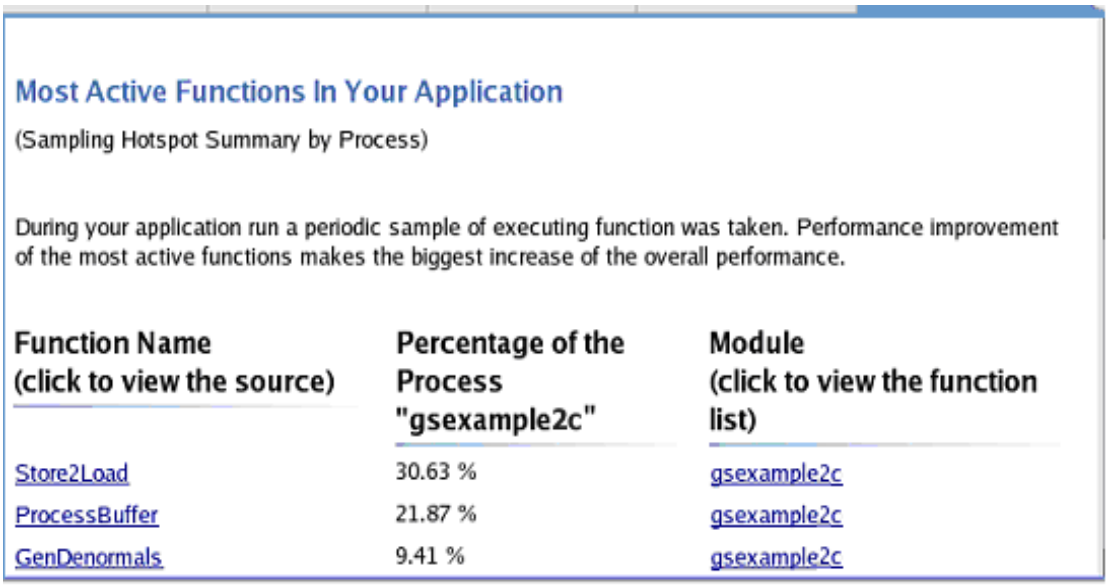
```
./gsexample2c gsexample2c.c 100000
```

経過時間が 11 秒から 7 秒へと著しく減少したことが確認できます。

```
**************************************
* Elapsed time was 7 seconds
* File character count: 1865615
* Total characters counted: 1877906272
* CRC calculated: 0x75
**************************************
```

次に、VTune アナライザを実行して、アプリケーションのパフォーマンスをチェックします。前回と同じ手順で **[First Use Wizard (簡易ウィザード)]** を使用します。

[First Use Wizard (簡易ウィザード)] はアクティビティを作成して実行します。実行後、**[Sampling Summary (サンプリング・サマリ)]** が表示され、最もアクティブな 5 つの関数が確認できます。前回 (修正前のアプリケーションで) アクティビティを実行した際に表示された **[Sampling Summary (サンプリング・サマリ)]** の内容と比較します。



Most Active Functions In Your Application

(Sampling Hotspot Summary by Process)

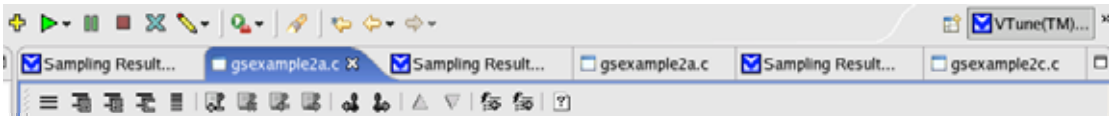
During your application run a periodic sample of executing function was taken. Performance improvement of the most active functions makes the biggest increase of the overall performance.

Function Name (click to view the source)	Percentage of the Process "gsexample2c"	Module (click to view the function list)
Store2Load	30.63 %	gsexample2c
ProcessBuffer	21.87 %	gsexample2c
GenDenormals	9.41 %	gsexample2c

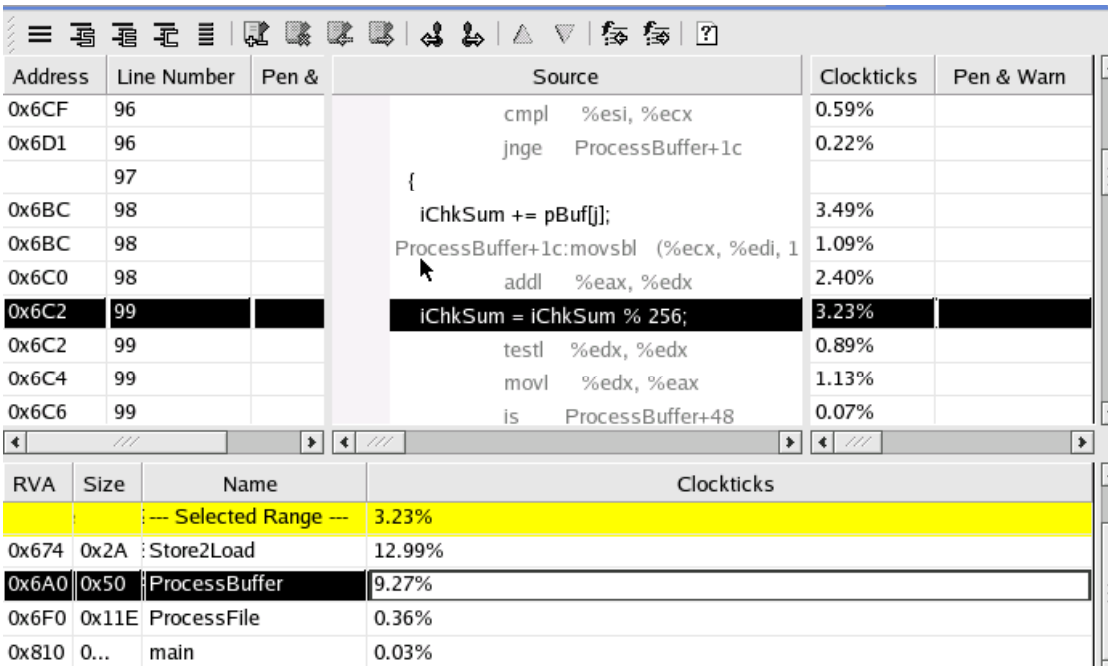
`ProcessBuffer` で費やされたプロセッサ時間の割合に注目すると、減少していることが確認できます。

次に、`ProcessBuffer` 関数の **[Source (ソース)]** ビューを表示した際の手順に従って、修正後のアプリケーションの **[Source (ソース)]** ビューを表示します。

`gsexample2a.c` タブと `gsexample2c.c` タブをクリックすると、修正前の **[Source (ソース)]** ビューと修正後の **[Source (ソース)]** ビューを切り替えることができます。



3. 以前と比べると、`ProcessBuffer` 関数における Clockticks (Clocktick の回数) の割合が減少し、アプリケーションのパフォーマンスが改善されたことが確認できます。



Address	Line Number	Pen &	Source	Clockticks	Pen & Warn
0x6CF	96		cmpl %esi, %ecx	0.59%	
0x6D1	96		jnge ProcessBuffer+1c	0.22%	
	97		{		
0x6BC	98		iChkSum += pBuf[j];	3.49%	
0x6BC	98		ProcessBuffer+1c:movsbl (%ecx, %edi, 1	1.09%	
0x6C0	98		addl %eax, %edx	2.40%	
0x6C2	99		iChkSum = iChkSum % 256;	3.23%	
0x6C2	99		testl %edx, %edx	0.89%	
0x6C4	99		movl %edx, %eax	1.13%	
0x6C6	99		is ProcessBuffer+48	0.07%	

RVA	Size	Name	Clockticks
		--- Selected Range ---	3.23%
0x674	0x2A	Store2Load	12.99%
0x6A0	0x50	ProcessBuffer	9.27%
0x6F0	0x11E	ProcessFile	0.36%
0x810	0...	main	0.03%

3. さらに詳細なコードの解析

ステップ 2 で **[First Use Wizard (簡易ウィザード)]** を使用してパフォーマンスを改善した後、VTune アナライザの **[Sampling Wizard (サンプリング・ウィザード)]** を使用して、パフォーマンスをさらに向上させることができます。**[Sampling Wizard (サンプリング・ウィザード)]** は、コードのパフォーマンスにおける、プロセッサのマイクロアーキテクチャの影響について検証します。

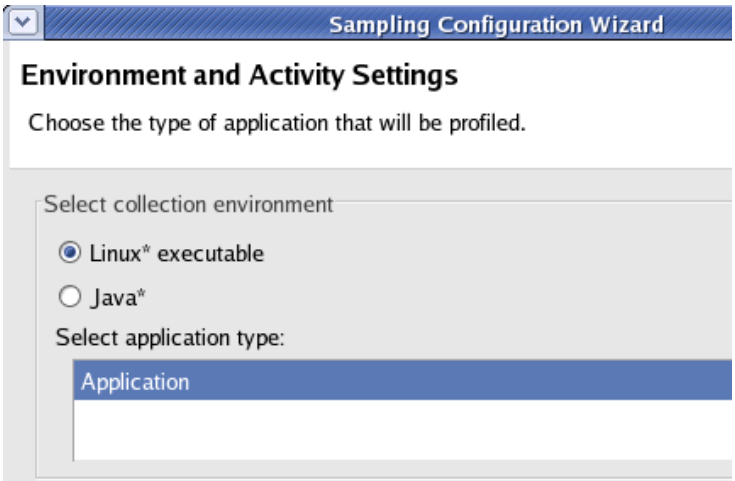
[Sampling Wizard (サンプリング・ウィザード)] では、データ収集中に発生したプロセッサ・イベントに関する情報を収集します。サンプリング・データ・コレクタは、**[First Use Wizard (簡易ウィザード)]** を設定する際にも使用されますが、**[Sampling Wizard (サンプリング・ウィザード)]** では、データを収集したいイベントを複数定義することができます。このため、プログラムの実行時における特定のデータを、さまざまな観点から収集することができます。

アプリケーションの解析を始める前に、選択可能なイベントに関する情報を入手しておくと便利です。オンライン・ヘルプの「**リファレンス**」にある「**プロセッサのイベントとアドバイス(Processor Events and Advice)**」ブックの対象プロセッサを参照してください。

3 a. [Sampling Wizard (サンプリング・ウィザード)] の使用

次の手順に従い、イベント・ベース・サンプリング (EBS) を使用してアプリケーションを解析します。

1. `gsexample3a` アプリケーションを実行して、時間を記録します。
   これは、アプリケーションのチューニングにおける現段階のベンチマークです。
2. VTune アナライザのツールバーで、**[Add Tuning Activity (チューニング・アクティビティの追加)]** をクリックして、**[Select A Wizard (ウィザードの選択)]** ダイアログ・ボックスを開きます。
3. **[Sampling Wizard (サンプリング・ウィザード)]** () を選択します。
4. **[Next (次へ)]** をクリックして、**[Environment and Activity Settings (環境とアクティビティ設定)]** ページに進みます。
5. **Linux* executable** 収集環境を選択し、**[Select application type (アプリケーションの種類を選択)]** ボックスで **[Application (アプリケーション)]** を選択します。

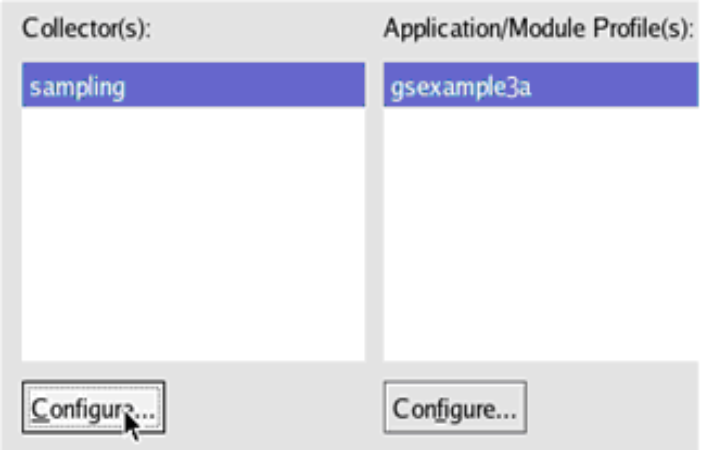


サンプルが収集される頻度は、サンプリング・データ収集の間にシステム上で実行中のソフトウェアがイベントを発生させる頻度によって決まります。デフォルトでは、Clockticks (Clocktick の回数) イベントが選択されています。

イベント・ベース・サンプリング中に、サンプリング・コレクタは複数回実行されます。各実行で、サンプリング・コレクタは次のいずれかの操作を行います。

- 選択されたイベントの **サンプリング間隔の値 (SAV)** のキャリブレーションを行います。データ収集中、キャリブレーションが行われた**サンプリング間隔の値 (SAV)** に基づいてデータは収集されます。
- デフォルトのサンプリング間隔の値 (SAV) を使用して、選択されたイベントからデータを収集します。

6. **[Next (次へ)]** をクリックし、**[Application/Module Profile Settings (アプリケーション/モジュール・プロファイル設定)]** ページに進みます。
7. **[Application parameters (アプリケーション・パラメータ)]** で `gsexample3a` アプリケーションを参照します。**[Working Directory (作業ディレクトリ)]** フィールドの内容は、アプリケーション・パスを指定すると自動的に設定されます。
8. **[Application Arguments (アプリケーションの引数)]** フィールドで、次の値を入力します。
   `datafile.txt 200000`
9. **[Next (次へ)]** をクリックして、**[Sampling Collection Settings (サンプリング収集設定)]** ページに進みます。
10. **[Sampling Collection Settings (サンプリング収集設定)]** ページで、**[When the application terminates (before duration completes) (アプリケーションの終了時 (サンプリング間隔前))]** チェックボックスをオフにします。
11. **[Next (次へ)]** をクリックして、**[Options (オプション)]** ページに進みます。**[Modify Configuration (設定を変更)]** で、**[Change the Sampling Events and set advanced options such as interval, calibration, and delay (サンプリング・イベントを変更して間隔、キャリブレーション、遅延などの詳細情報を設定)]** チェックボックスをオンにします。
12. **[Finish (完了)]** をクリックした後にアクティビティを実行するには、**[Run this Activity (アクティビティを実行)]** チェックボックスをオンにします。
13. **[Finish (完了)]** をクリックして、アクティビティを作成し、**[Activity Configuration (アクティビティの設定)]** ダイアログ・ボックスを開きます。
14. **[Duration (継続期間)]** フィールドで **[second(s) (秒)]** ラジオボタンをクリックして、編集可能なボックスに `10` と入力します。これにより、アクティビティを実行してデータを収集する時間が `10` 秒に設定されます。
15. **[Collector(s) (コレクタ)]** リストの下にある **[Configure... (設定...)]** ボタンをクリックして、サンプリング・コレクタを設定します。



16. 表示される **[Configure Sampling (サンプリングの設定)]** ダイアログ・ボックスで、左側にあるタブの中から **[Events (イベント)]** タブを選択します。

17. **[Available Events (選択可能なイベント)]** フィールドで、MOB Load Replays Retired (リタイアした MOB ロードのリプレイ) イベントにスクロールします。このイベントを選択し、次の図に示すように矢印をクリックして、**[Selected Events (選択されたイベント)]** リストに追加します。

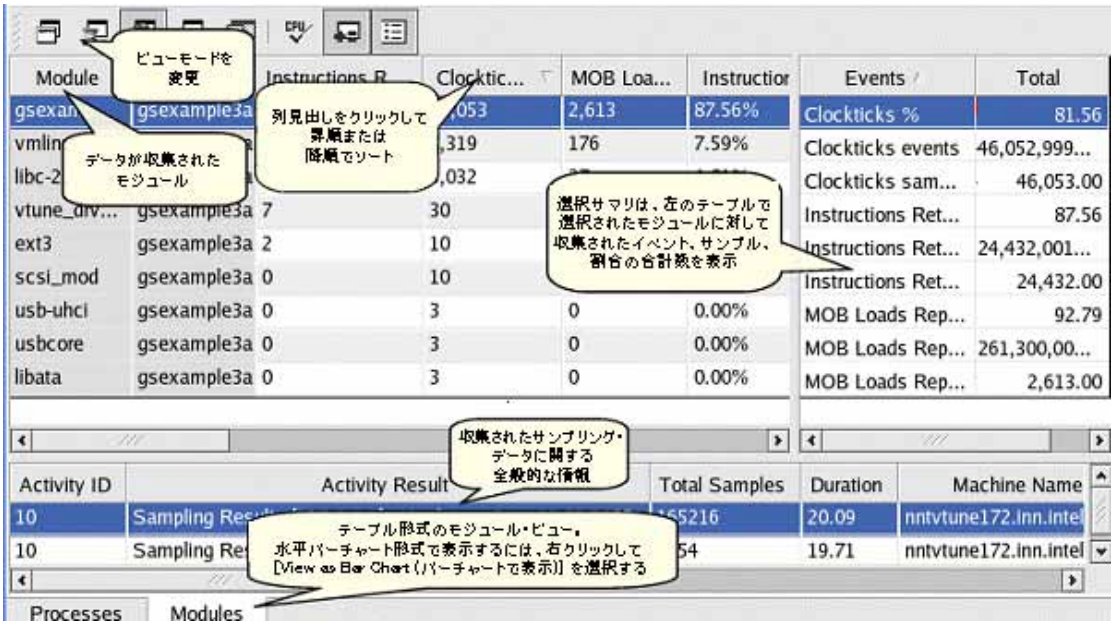
18. **[OK]** をクリックしてダイアログ・ボックスを閉じ、データ収集を開始します。VTune アナライザは、アクティビティを実行して、終了時にサンプリング結果を表示します。

3 b. サンプリング結果の表示

アプリケーションの分析は、**[Process (プロセス)]** ビューから始めてドリルダウンすると良いでしょう。

サンプリング収集アクティビティを表示する際には、デフォルトで **[Sampling (サンプリング)]** ビューが表示されます。ツールバーのボタンを使用して、ビューを変更できます。

任意のデータを表示できるサンプリング・ビューの機能について次に示します。



ビューには、次のような有益な情報が表示されます。

- *[Process (プロセス)] ビュー*: サンプリング・データが収集されたときにシステム上で実行されていたすべてのプロセスを表示します。特定プロセス中の高いイベント数は、潜在的なパフォーマンス・ボトルネックになる可能性のある、高い CPU 時間を示します。
- *[Module (モジュール)] ビュー*: これらの情報を使用して、**[Module (モジュール)]** ビューから [Hotspot (ホットスポット)] ビューにドリルダウンします。サンプリング・データ収集中に頻繁に呼び出されたモジュールは、最大イベント数または最大 CPU 時間のモジュールとして表示されます。

複数回実行されるシングルスレッド・アプリケーションでは、**[Thread (スレッド)]** ビューを使用しないでください。

[Sampling (サンプリング)] ビューを使用して、問題のあるコード部分を特定します。

1. デフォルトで表示される **[Process (プロセス)]** ビューの `gsexample3a` プロセスのセクションを確認します。

2.  をクリックして、**[Module (モジュール)]** ビューに切り替えます。
3. `gsexample3a` モジュールをクリックして選択します。
4.  をクリックして、`gsexample3a` モジュールで最もアクティブな関数を表示します。
5. `Store2Load` 関数で、最も多くのイベントと *MOB Load Replays Retired (リタイアした MOB ロードのリプレイ)* が発生していることが確認できます。ここが、このコードの問題箇所です。
6.  をクリックして、この関数の **[Source (ソース)]** ビューを表示します。

3 c. コードの修正

コードの解析では、ブロックされたストア・フォワード問題が見つかりました。次の手順に従ってコードを修正するか、または `opt/intel/vtune/samples/gsexample` ディレクトリにある修正済みのコードを使用してください。

1. アプリケーション開発に使用しているエディタでファイルを開きます。
2. `Store2Load` 関数のソースを開きます。

コードの問題部分を次に示します。

```
*(((char*) (pBits + i)) + 1) = 0;
clr = *(pBits + i);
```

ここで問題となっているのは、最初の行で 32 ビットの値の中の 1 バイトをクリアした後に、その 32 ビットの値をロードしていることです。プロセッサはバイトの書き込みが終わるまで、問題のバイトを含む 32 ビットの値をロードすることができないため、ブロックされたストア・フォワードが発生します。

この問題を回避するには、同じサイズのデータを読み取り、書き出す必要があります。32 ビットの値を読み取り、この値の中の 1 バイトを 32 ビットの整数でクリアし、32 ビットの値として格納するように、コードを変更します。

修正後のコードを次に示します。

```
clr = *(pBits + i);
clr &= 0xffff00ff;
*(pBits + i) = clr;
```

3. 前回と同じ設定でアプリケーションを再ビルドします。

MOB Load Replays Retired (リタイアした MOB ロードのリプレイ) - このイベントは、store-to-load フォワード制限が守られなかったため、メモリ・オーダー・バッファ (MOB) のリプレイに直面した、リタイアしたロード命令の数をカウントします。MOB リプレイは、複数の理由により起こります。このイベントは、store-to-load フォワードがブロックされたロードにより発生する MOB リプレイをカウントするようにプログラムされています。

3 d. コードの修正前と修正後のパフォーマンス比較

ステップ 3c で修正したコードの結果を比較するには、ステップ 3a の手順に従って、別のサンプリング・アクティビティを作成します。最適化されたコードを使用するか、または `/opt/intel/vtune/samples/gsexample` ディレクトリにある `gsexample3c` アプリケーションを使用してください。アクティビティを作成して実行した後、結果を確認します。

Name	Instructions Retired ...	Clockticks samples	MOB Loads Replays Retired s
ProcessBuffer	22,376	29,380	16
GenDenormals	130	4,782	0
Store2Load	2,011	1,099	2

MOB Load Replays Retired (リタイアした MOB ロードのリプレイ) の数が減少し、アプリケーションのパフォーマンスが向上したことが確認できます。

次のステップ

VTune アナライザでは、データコレクタを使用してアプリケーションのさらなる解析を行うことができます。製品の機能を理解する上で役立つ `Web` ベースのチュートリアルが `/opt/intel/vtune/traning/gs_vtl/index.htm` にあります。ご活用ください。

インテル® ソフトウェア開発製品の詳細については、次のインテル Web サイトを参照してください。

http://www.intel.com/cd/software/products/ijkk/jpn/index.htmT

テクニカル・サポートおよび製品の制限事項については、製品のリリースノートを参照してください。

著作権/法律に基づく表示