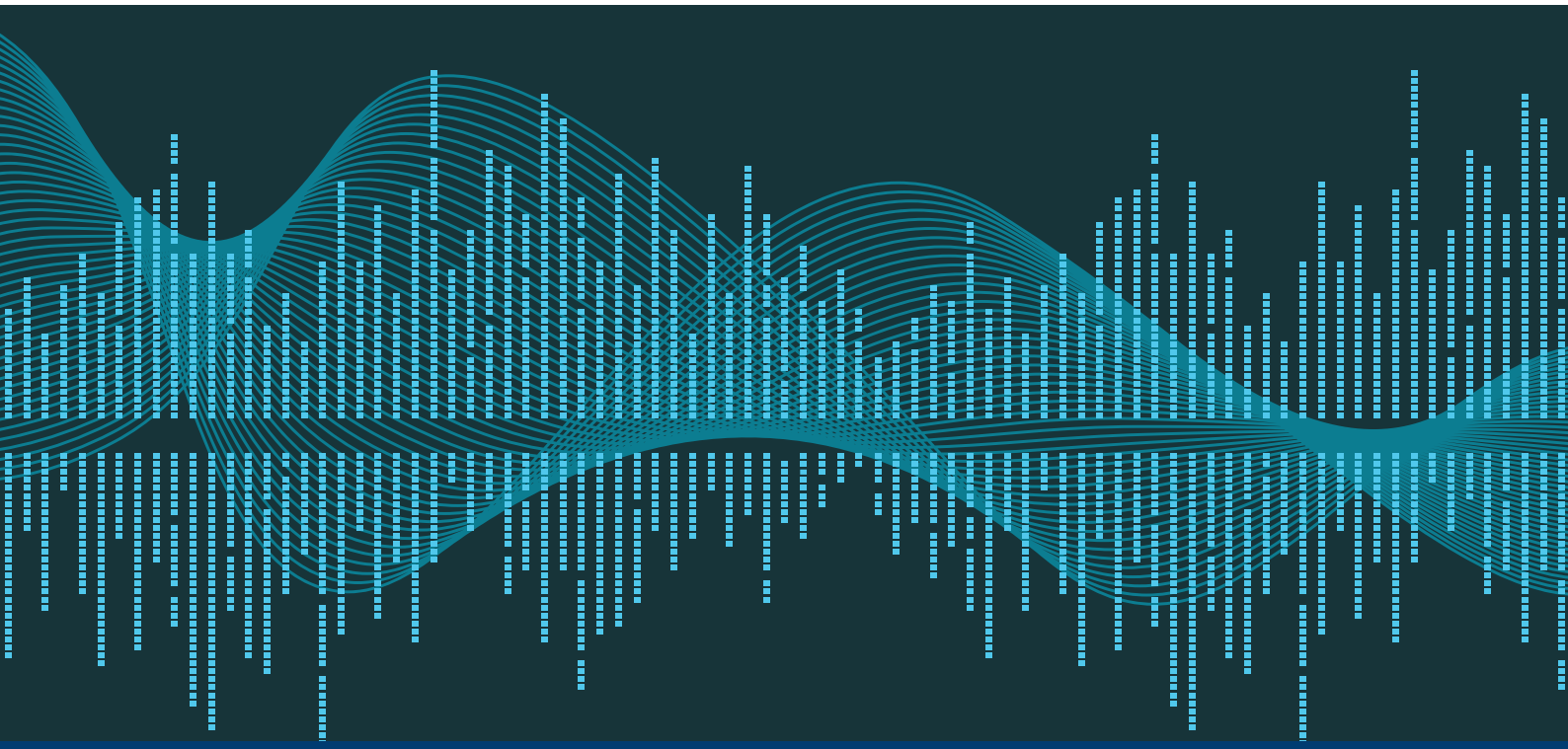


インテル® VTUNE™ AMPLIFIER XE による JAVA* および PYTHON* コードのプロファイル

Java* および Python* ベースのアプリケーションで CPU 性能を引き出す

Sukruv Hv インテル コーポレーション ソフトウェア・テクニカル・コンサルティング・エンジニア

長年にわたり、Java* は開発者に人気のある言語で、エンタープライズ、組込み、IoT (モノのインターネット) 向けアプリケーションを牽引してきました。インテル® VTune™ Amplifier XE の電力およびパフォーマンス・プロファイラー (インテル® ソフトウェア・ツール・スイートに含まれる) は、長い間、Java* や .NET のようなマネージドコードのプロファイルを行ってきました。Python* も、使いやすさとシステムを効率良く統合できることから、人気のプログラミング言語になりました。

この 2 つの異なるプログラミング言語の人気の高まりを受け、これらの言語で開発されたアプリケーションが CPU 性能を効率良く利用できるように、インテル® VTune™ Amplifier XE のプロファイル機能が拡張され、Java* および Python* ベースのアプリケーションに対応しました。ここでは、インテル® VTune™ Amplifier XE を使用して、アプリケーションの振る舞いを詳しく理解します。

インテル® VTune™ Amplifier XE とは ?

インテル® VTune™ Amplifier XE は、多くの CPU 時間を費やしているモジュール / プロセス (hotspot) の特定を支援する、パフォーマンスおよび電力プロファイラーです。サンプリングにより、最小限のオーバーヘッドで、マイクロアーキテクチャー・レベルの問題 (例えば、キャッシュミス、ページウォーク、TLB 問題など) を見つけることができます。

アプリケーション・パフォーマンスをプロファイルする理由

アプリケーション・パフォーマンスのプロファイルは、多くの CPU クロックサイクルを費やしているさまざまなコードブロックを特定するのに必要です。この情報は、手動でタイミング API を挿入して取得することもできます。ただし、プロジェクトに多数のモジュールが含まれる場合、手動では問題のあるモジュールの検証と特定に多くの時間がかかります。インテル® VTune™ Amplifier XE の各種プロファイラーは、問題にドリルダウンするのに役立ちます。

インテル® VTune™ Amplifier XE の機能

インテル® VTune™ Amplifier XE には、直感的で使いやすいさまざまな機能があります。

- **サンプリング :** インストルメンテーションと比べて最小限のオーバーヘッドでプロファイルできます。
- **使いやすいプロファイラー :** さまざまなグループ / フィルター / 呼び出し元 - 呼び出し先オプションにより、問題のコード領域を絞り込むことができます。
- **異なるレベルの情報 :** ソースレベルとアセンブリー・レベルで情報を提供します。
- **マイクロアーキテクチャーに関する情報 :** アプリケーションのマイクロアーキテクチャーに関するさまざまな情報を提供します。

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。

Java* アプリケーションのプロファイル

Java* アプリケーションのプロファイルは、4 つのステップから成ります。

1. インテル® VTune™ Amplifier XE プロジェクトを作成します。
2. プロファイルするアプリケーション / プロセスを選択します。
3. 解析タイプを選択します。
4. 結果を収集して、解釈します。

Windows® 上でのインテル® VTune™ Amplifier XE プロジェクトの作成

最初に、amplxe-vars バッチファイルを使用して環境変数を設定します。例えば、インテル® VTune™ Amplifier XE がデフォルトのディレクトリーにインストールされている場合は、次のコマンドを実行します。

```
C:\[Program Files]\IntelSWTools\VTune Amplifier XE\amplxe-vars.bat
```

バッチファイルは、製品名とビルド番号を表示します。

次に、インテル® VTune™ Amplifier XE を起動します。スタンドアロン GUI インターフェイスの場合は、**amplxe-gui** コマンドを実行します。コマンドライン・インターフェイスの場合は、**amplxe-cl** コマンドを実行します。

インテル® VTune™ Amplifier XE プロジェクトを作成します (スタンドアロンの場合のみ)。

- 右上のメニューボタンをクリックして、[New] > [Project] を選択します。
- [Create Project] ダイアログボックスでプロジェクトの名前と場所を指定します。

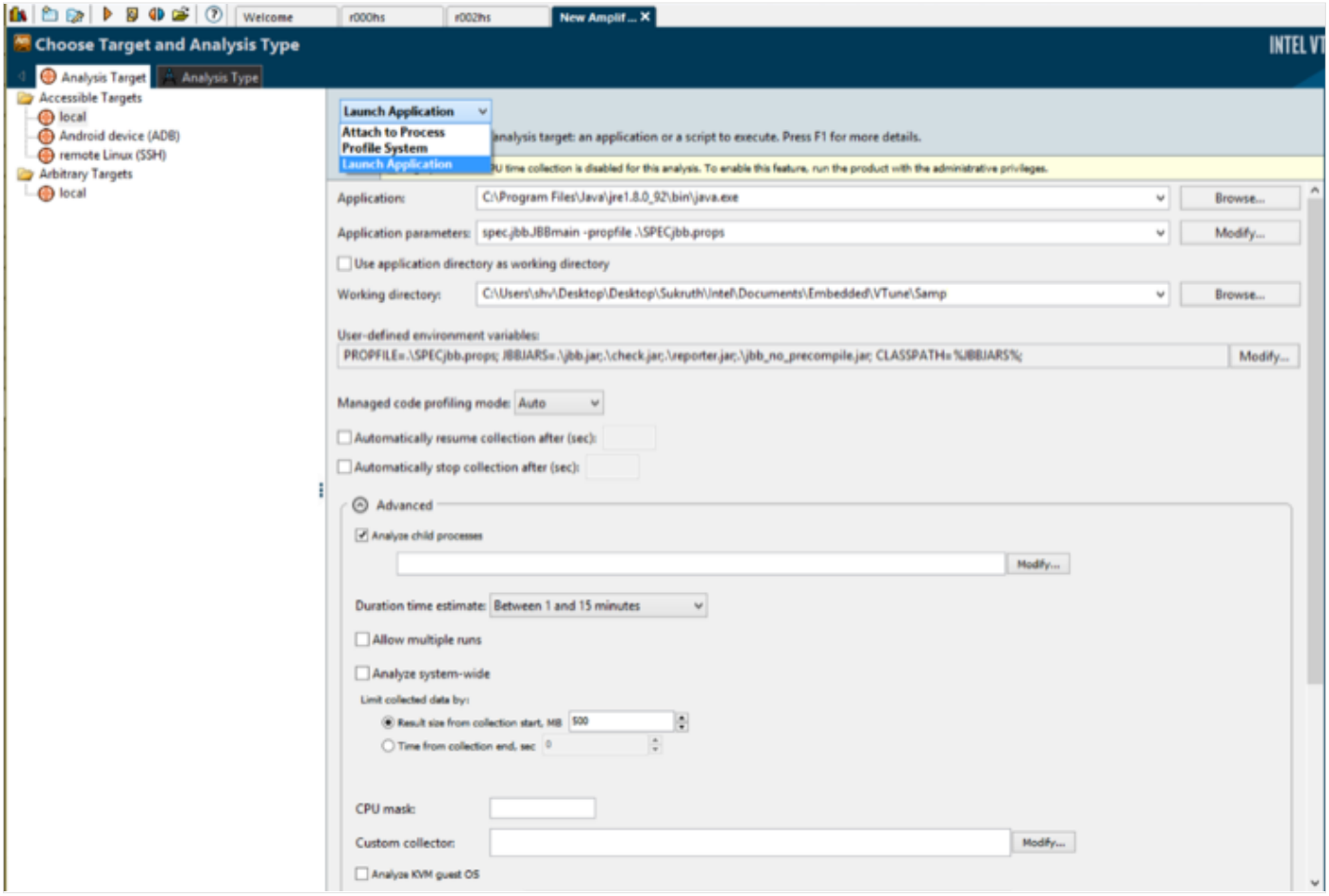
注 : Linux* プラットフォーム上でプロジェクトを新規作成する方法は、**「インテル® VTune™ Amplifier XE 2017 for Linux* 入門」** (英語) を参照してください。

プロファイルするアプリケーション / プロセスの選択

[Analysis Target] タブの左ペインでターゲットシステムを選択し、右ペインで解析ターゲットタイプを選択します。次のいずれかを選択できます。

- **Launch Application:** プロファイルするアプリケーションを起動します。
- **Attach to Process:** すでに実行中のアプリケーションにプロファイラーをアタッチします。
- **Profile System:** アプリケーションとシステムコールの相互作用を検証します。

[Launch Application] を選択する場合、[Application] フィールドに java.exe ファイルのパスを、[Application parameters] に "Java options" や "prop files" のようなパラメーターを指定します (**図 1**)。そして、[User-defined environment variables] フィールドで、アプリケーションの実行に必要な環境変数を指定します。別の方法として、*.bat ファイルで各種環境変数、java.exe バイナリー、パラメーターを定義して、[Application] フィールドで java.exe ファイルの代わりにその *.bat ファイルを指定することもできます。



1 ターゲットと解析タイプの選択

解析タイプの選択

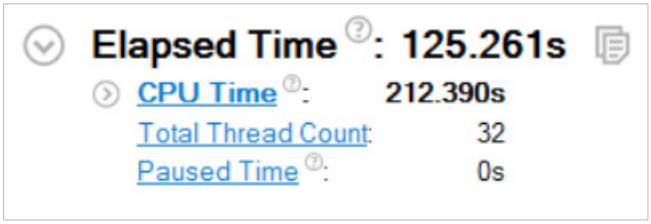
[Analysis Type] タブに移動して、[Basic Hotspots] 解析タイプを選択し、[Start] をクリックします。

結果の収集と解釈

ここでは、SPECjbb2000 ベンチマークを使用しています。

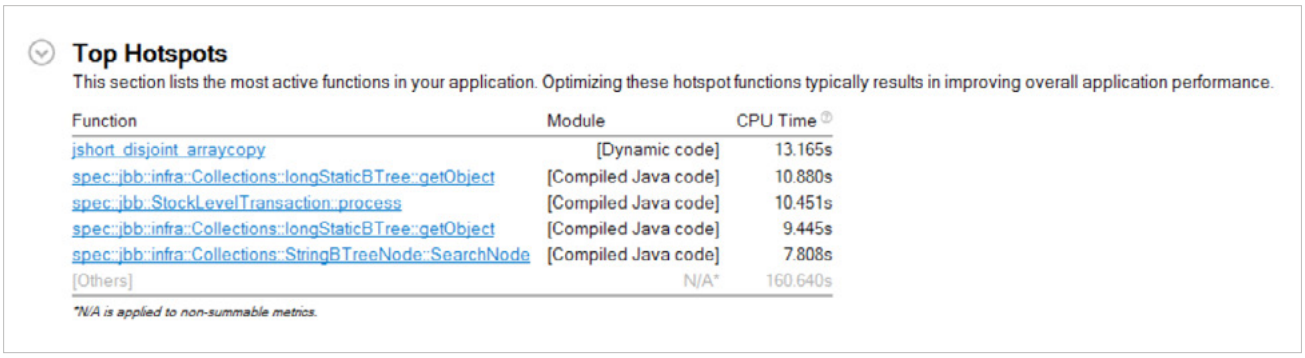
データ収集とファイナライズが完了すると、[Summary] タブが開き、各種メトリックが表示されます。いくつかのメトリックについて見ていきましょう。

- **[Summary] タブの最初のセクション (図 2)** には、アプリケーションの Elapsed Time (経過時間 : アプリケーションの開始から終了までのウォールクロック時間)、CPU Time (CPU 時間 : アプリケーションが CPU を使用していた時間)、Total Thread Count (合計スレッド数 : アプリケーションで使用されたスレッド数) が表示されます。



2 経過時間

- **[Summary] タブの 2 つ目のセクション (図 3)** には、Top Hotspots (上位の hotspot: 多くの CPU 時間を費やしている関数) が表示されます。この例では、5 つの hotspot が見つかり、1 つ目の hotspot は 13.165 秒を費やしています。[Module] 列に表示されている [Dynamic code] と [Compiled Java code] は Java* メソッドで、ネイティブバイナリーのモジュールに関連付けることはできません。すべての Java* ユーザーメソッドは、[Compiled Java code] モジュールに関連付けられます。また、JVM は内部使用目的にいくつかのメソッドを生成し、Java* プロファイラーにレポートしますが、これらは [Dynamic code] モジュールに関連付けられます。



Top Hotspots

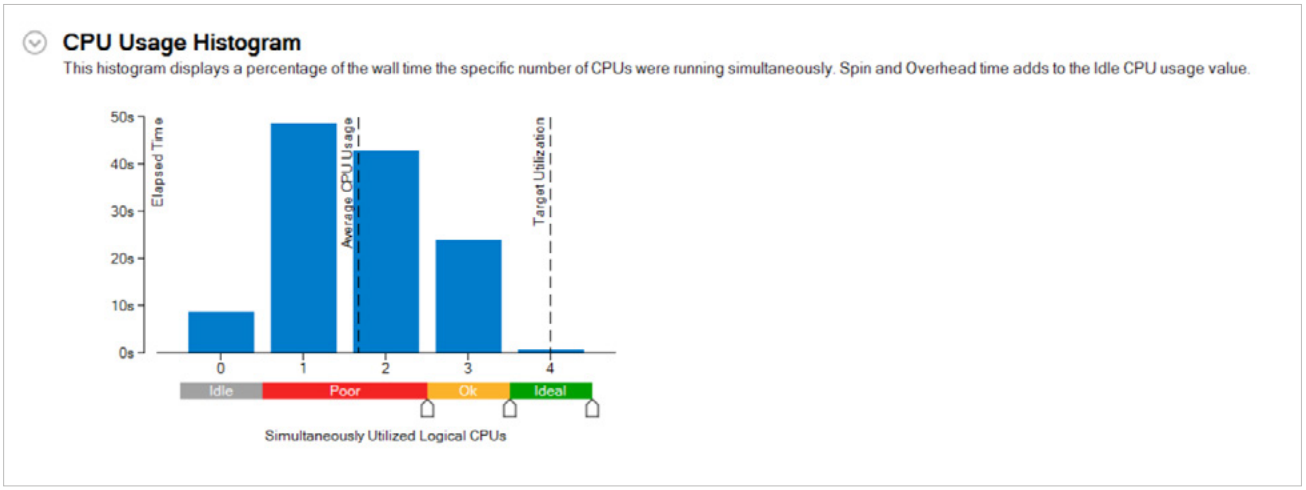
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time
jshort_disjoint_arraycopy	[Dynamic code]	13.165s
spec::jbb::infra::Collections::longStaticBTree::getObject	[Compiled Java code]	10.880s
spec::jbb::StockLevelTransaction::process	[Compiled Java code]	10.451s
spec::jbb::infra::Collections::longStaticBTree::getObject	[Compiled Java code]	9.445s
spec::jbb::infra::Collections::StringBTreeNode::SearchNode	[Compiled Java code]	7.808s
[Others]	N/A*	160.640s

*N/A is applied to non-summable metrics.

3 上位の hotspot

[Summary] タブの 3 つ目のセクション (図 4) には、CPU Usage Histogram (CPU 使用ヒストグラム : アプリケーションの実行に使用された論理コア数のグラフ) が表示されます。インテル® VTune™ Amplifier XE は、アプリケーションで使用されたコア数を Idle (アイドル)、Poor (低)、OK (許容範囲)、Ideal (理想) に分類し、色分けして表示します。



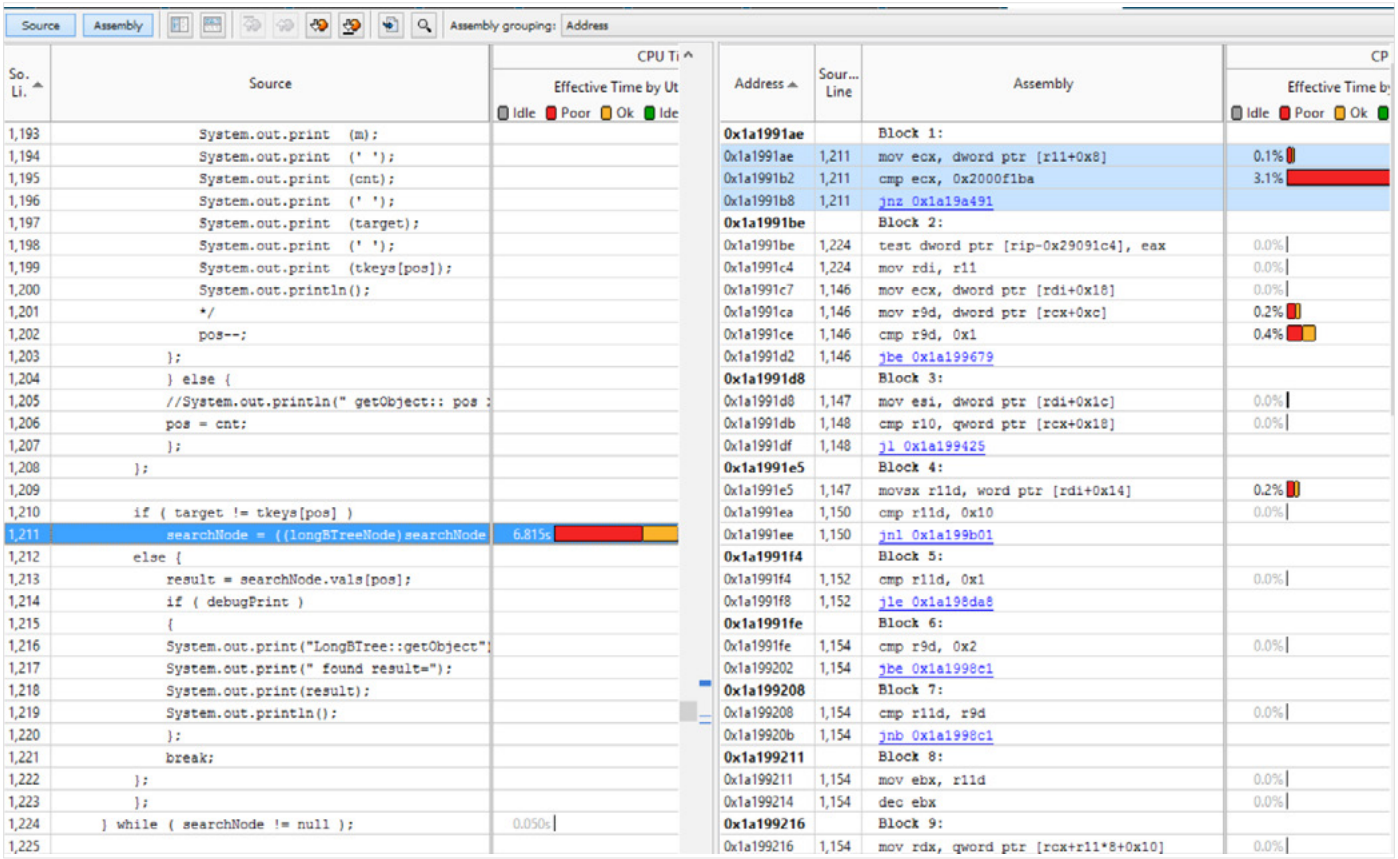
4 CPU 使用ヒストグラム

[Bottom-up] タブからは、さまざまな詳細が得られます (図 5)。**グループ**や**フィルター**を使用したり、特定の期間に絞って、各スレッドのワークロードをタイムライン表示することができます。関数 / モジュールをダブルクリックすることで、対応するソース行へドリルダウンすることも可能です。

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle / Poor / Ok / Ideal / Over)	Spin Time	Ove... Time	Module	Function (Full)	Sour... File	Start Address
jshort_disjoint_arraycopy	13.165s	0s	0s	[Dynamic code]	jshort_disjoint_ar ...		0x19c22980
spec::jbb::infra::Collections::longStaticBT	10.880s	0s	0s	[Compiled Java code]	spec::jbb::infra::C ...	long ...	0x1a1991ae
spec::jbb::StockLevelTransaction::process	10.451s	0s	0s	[Compiled Java code]	spec::jbb::StockL ...	Stoc...	0x19fb6413
spec::jbb::infra::Collections::longStaticBT	9.445s	0s	0s	[Compiled Java code]	spec::jbb::infra::C ...	long ...	0x1a19931f
spec::jbb::infra::Collections::StringBTreeN	7.808s	0s	0s	[Compiled Java code]	spec::jbb::infra::C ...	Strin...	0x19e08420
spec::jbb::infra::Collections::longStaticBT	5.999s	0s	0s	[Compiled Java code]	spec::jbb::infra::C ...	long ...	0x1a199283
spec::jbb::infra::Collections::longStaticBT	5.988s	0s	0s	[Compiled Java code]	spec::jbb::infra::C ...	long ...	0x1a198da8
java::util::Hashtable::put	3.499s	0s	0s	[Compiled Java code]	java::util::Hashta ...		0x19cf9b20
spec::jbb::Orderline::process	3.188s	0s	0s	[Compiled Java code]	spec::jbb::Orderli ...	Orde...	0x19e256a9
spec::jbb::infra::Util::DisplayScreen::privIn	2.846s	0s	0s	[Compiled Java code]	spec::jbb::infra::U ...	Displ...	0x1a0807ec
spec::jbb::infra::Util::DisplayScreen::putDc	2.804s	0s	0s	[Compiled Java code]	spec::jbb::infra::U ...	Displ...	0x19efa0e0
spec::jbb::infra::Collections::longStaticBT	2.557s	0s	0s	[Compiled Java code]	spec::jbb::infra::C ...	long ...	0x19f40ebf
spec::jbb::infra::Collections::longStaticBT	2.349s	0s	0s	[Compiled Java code]	spec::jbb::infra::C ...	long ...	0x19f41479
spec::jbb::Orderline::validateAndProcess	2.218s	0s	0s	[Compiled Java code]	spec::jbb::Orderli ...	Orde...	0x1a199144
[Unknown stack frame(s)]	2.140s	0s	0s		[Unknown stack ...		0
WriteFile	1.823s	0s	0s	KERNELBASE.dll	WriteFile		0x180001 ...
Selected 1 row(s):	10.880s	0s	0s				

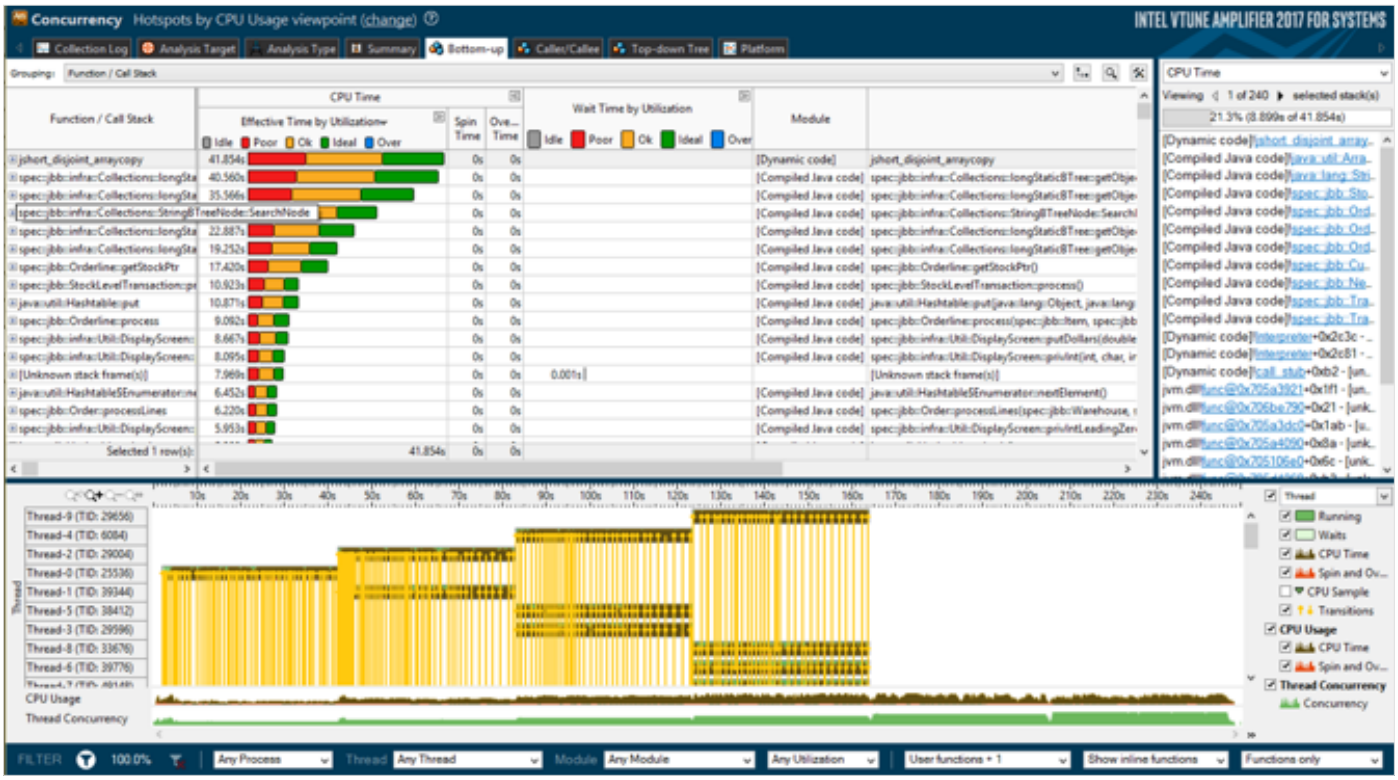
5 [Bottom-up] タブ

コードの最適化への取り掛かりとして、特定のソース行へのドリルダウンが役立ちます。 また、ハイライトしたソース行に対応するアセンブリー・コードも確認することができます (**図 6**)。



6 ソース行に対応するアセンブリー

Basic Hotspots 解析タイプで多くの CPU リソースを消費しているモジュールを把握したら、**Concurrency** 解析タイプを使用して、コンテキスト・スイッチとスレッド間の遷移を確認できます (**図 7**)。黄色の線は、スレッド間の遷移を示します。



7 Concurrency 解析

General Exploration 解析タイプは、**アーキテクチャー解析**を実行して、キャッシュ、TLB、ページウォーク、フォルス・シェアリングなどに関するさまざまな問題の詳細を提供し、これらの問題を理解するのを助けます。

Python* アプリケーションのプロファイル

Python* プロファイルは、インテル® VTune™ Amplifier XE 2017 の新機能です。

インテル® VTune™ Amplifier XE で Python* アプリケーションをプロファイルする方法は、**Parallel Universe 25 号**の「インテル® VTune™ Amplifier XE を利用した Python* コードの高速化」を参照してください。本記事では、混在モードのプロファイル、*.pyd モジュールのシンボルの解決、**コンパイラー**の設定、Cython* コンパイル向けのリンカーオプションについて説明します。

混在モードのプロファイル

Python* はインタープリター型言語で、コンパイラーを使用しないでバイナリー実行コードを生成します。Cython も (C 拡張ではありますが) インタープリター型言語で、ネイティブコードをビルドできます。インテル® VTune™ Amplifier XE 2017 は、ホットな関数のレポートにおいて Python* コードと Cython コードの両方をサポートしています。

Cython を利用するには、次の 2 つのステップを実行します。

1. *.pyx (Cython) ファイルを *.c ファイルに変換します。
2. 変換済みの *.c ファイルを *.pyd ファイル (Linux* の .so ファイルに相当) にコンパイルします。

この例で使用するサンプルは、512 × 512 行列乗算です。

以下は、*.pyd ファイルを生成し、Windows* 環境でインテル® VTune™ Amplifier XE を使用してシンボルを解決するためのステップです。

1. Cython を使用して *.pyx ソースコードを Python* から C へ変換するため、setup.py ファイルで拡張モジュール、コンパイラー・オプション、リンクオプションを設定します。

```
#setup.py
from distutils.core import setup
from Cython.Build import cythonize
from distutils.extension import Extension
from Cython.Distutils import build_ext

setup(
      cmdclass = {'build_ext': build_ext},
      ext_modules = [Extension('Matrix_mul', sources=["Matrix_mul.pyx"],
                        extra_compile_args=['/Z7'],
                        extra_link_args=['/DEBUG']),
                  ],
      )
```

上記の setup.py ファイルでは、デバッグシンボルを生成するため、/Z7 コンパイラー・オプションと /DEBUG リンカーオプションを指定して拡張モジュールを作成しています。インテル® VTune™ Amplifier XE でソースへドリルダウンする場合、オプションを指定することが重要です。Linux* プラットフォーム上では、extra_compile_args と extra_link_args の両方で -g を指定します。

2. 次のコマンドで Python* を呼び出し、Cython を使用して C に変換します。

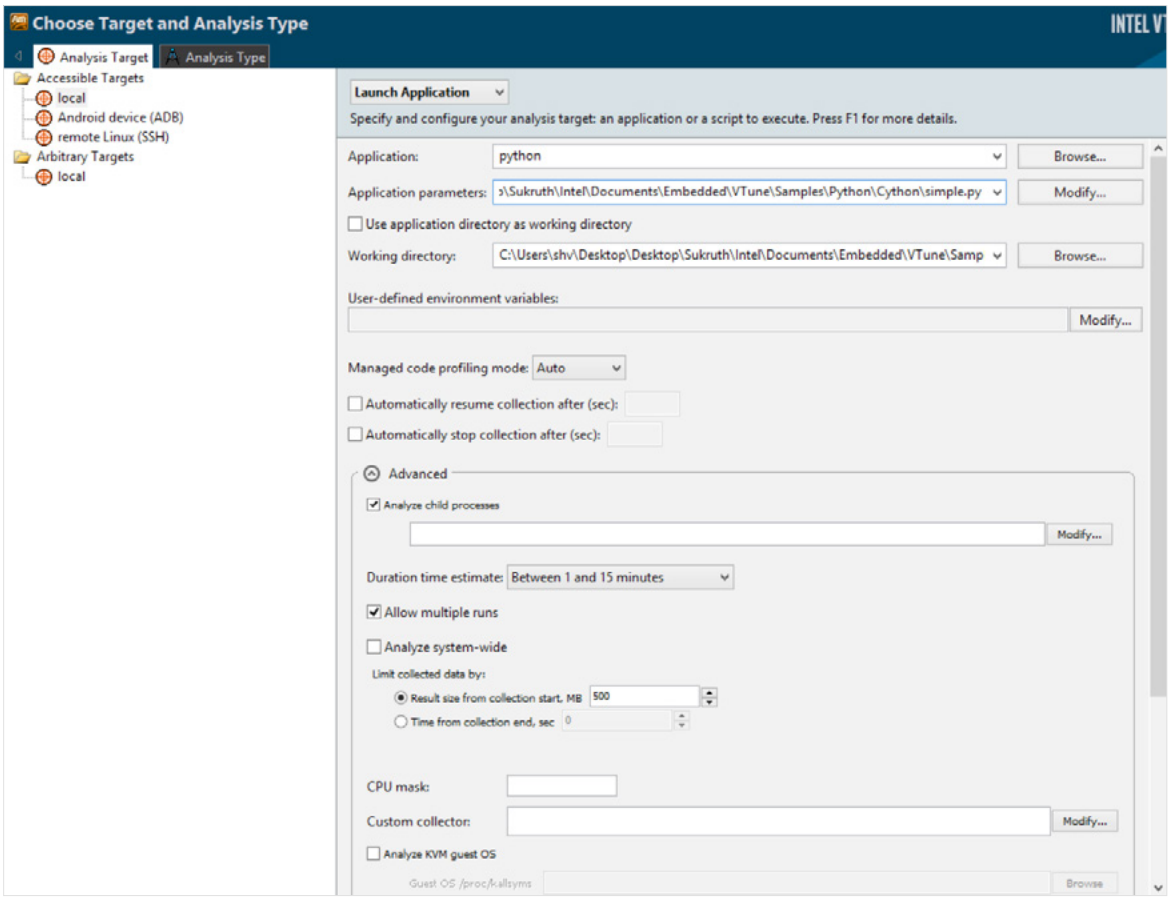
   python setup.py build_ext

   このコマンドは、対応する *.c と *.pyd モジュールを生成します。この例では、Matrix_mul.c と Matrix_mul.pyd が生成されます。ここでは、シンボル情報の生成も指定したため、*.pdb ファイルも生成されます。

3. Matrix_mul.pyd モジュールをインポートする Python* スクリプトを作成します。

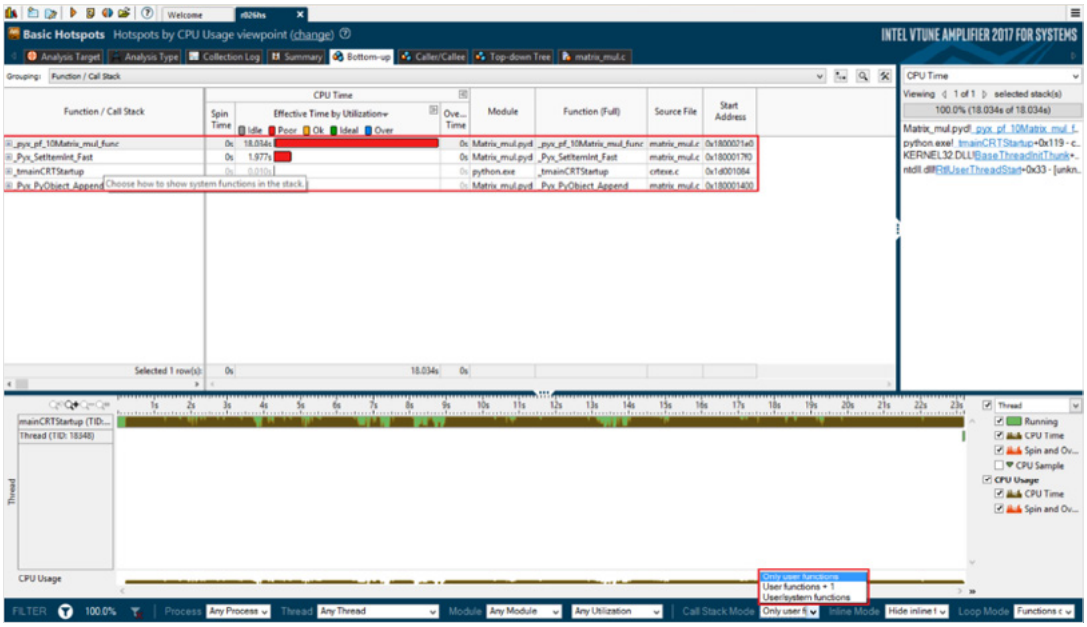
```
#simple.py
import Matrix _ mul
Matrix _ mul.func();
```

4. インテル® VTune™ Amplifier XE 2017 の Basic Hotspots 解析タイプで、Python* インタープリターへのパラメーターに simple.py を指定して、アプリケーションを解析します (**図 8**)。



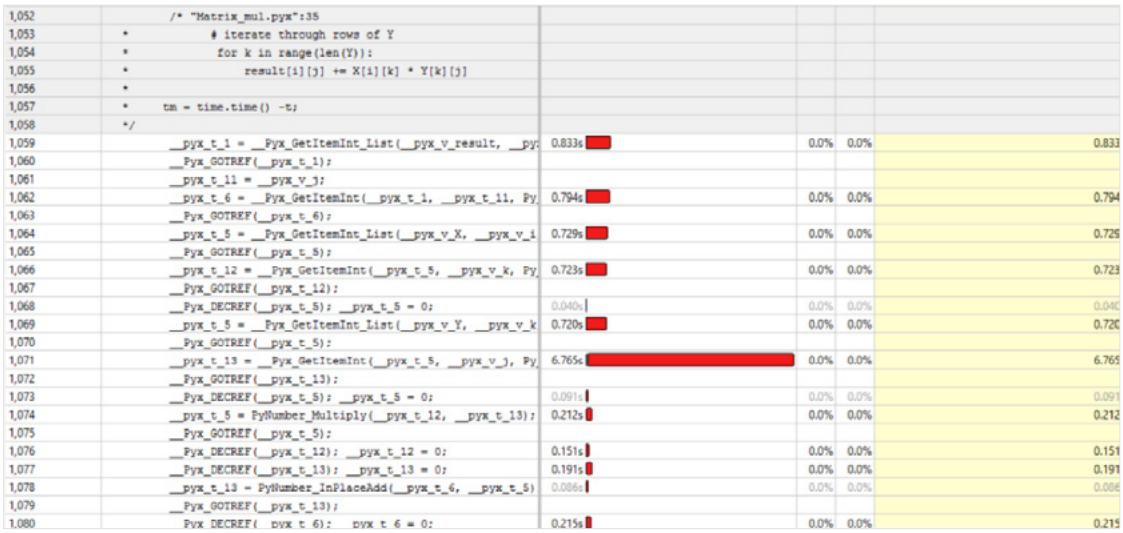
8 ターゲットと解析タイプの選択

5. Basic Hotspots 解析が完了したら、[Bottom-up] タブでモジュール / 関数を確認します (**図 9**)。



9 モジュール/関数

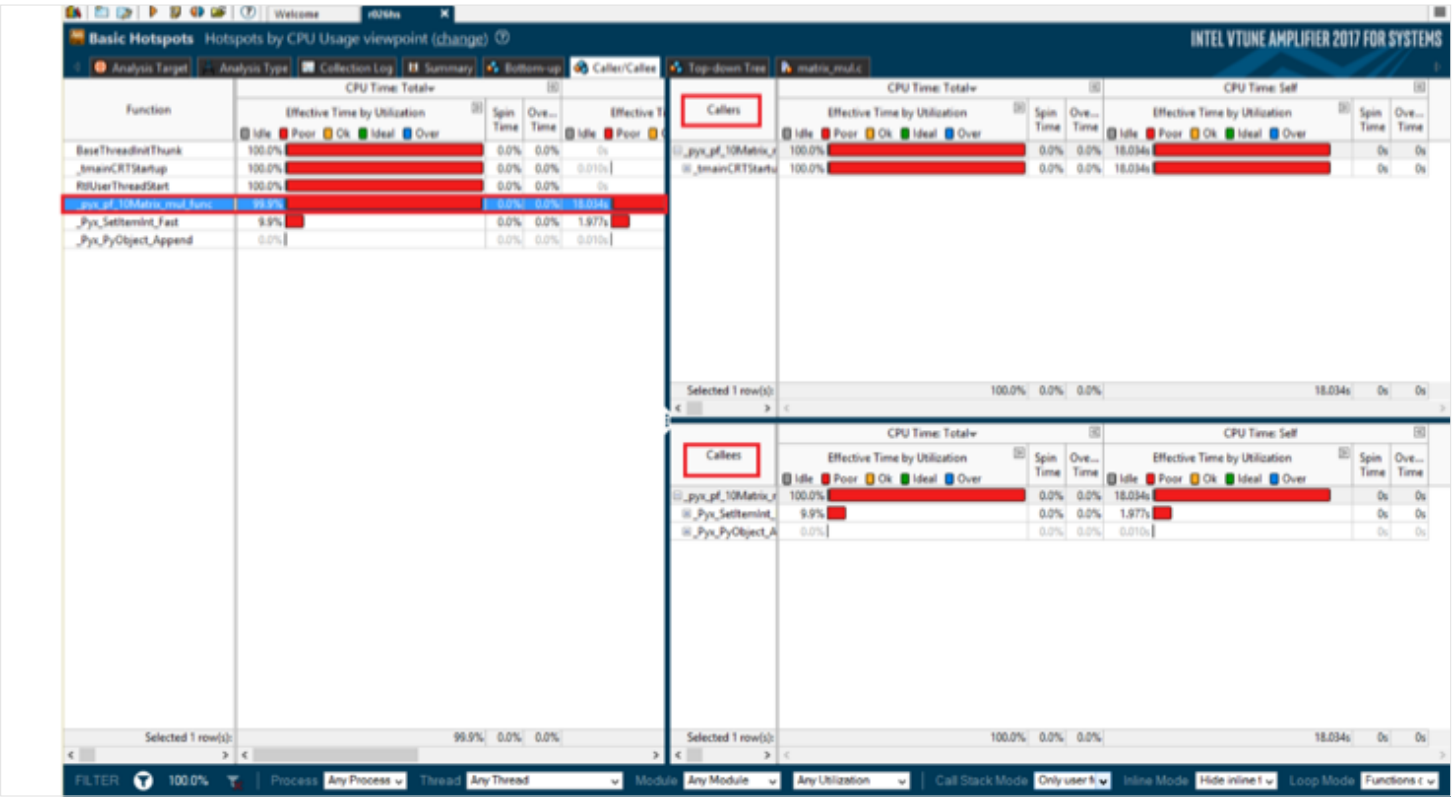
この例では、Matrix_mul.pyd モジュール (変換後のモジュール) が、多くの CPU 時間を費やしている hotspot の 1 つであることが分かります。ここでは、ユーザーコードの解析 / プロファイルのみ確認したいため、ページ下部の [FILTER] セクションにある [Call Stack Mode] で "Only user functions" (ユーザー関数のみ) を選択しています。ユーザーおよびシステムレベルの関数とそれらの相互作用を表示するには、"User/system functions" (ユーザー / システム関数) を選択します。さらに、ソースファイルにドリルダウンして、モジュールの特定のソース行を調査できます。また、C コードに対応するアセンブリーも確認できます。上記のスクリーンショットにある最初のユーザー関数 hotspot をダブルクリックすると、**図 10** のように、対応するソースファイルが表示されます。



10 最初のユーザー関数 hotspot をクリック

[Source] ビューを利用して、生成された C コードと対応する Python* コード (行 1052-1058) を関連付けることができます。このデータを使用して、インテル® VTune™ Amplifier XE で特定された Python* コードの問題の個所 / モジュールに絞って、最適化に取り掛かることができます。

[Caller/Callee] タブで、選択した関数 / モジュールの呼び出し元 - 呼び出し先を確認できます (**図 11**)。



11 [Caller/Callee] タブ

まとめ

インテル® VTune™ Amplifier XE は、Java* や Python* アプリケーションのパフォーマンス解析を、多くの CPU 時間を費やしている特定のコード (hotspot) に絞り込むことができます。これは、標準の Python* アプリケーションだけでなく、Python*/Cython が混在するコードでも可能です。

インテル® VTUNE™ AMPLIFIER XE の詳細 >