インテル® Advisor チュートリアル: 自動ルーフライン・グラフを使用して最適化の方針を決定

この記事は、インテル® デベロッパー・ゾーンに公開されている「Intel® Advisor Tutorial: Use the Automated Roofline Chart to Make Optimization Decisions」の日本語参考訳です。

バージョン: 2021.1 (更新日: 12/04/2020)

内容

このチュートリアルでは、C++ サンプル・アプリケーションを使用して、インテル® Advisor の自動**ルーフライン**・グラフでハードウェアによって課せられるパフォーマンスの上限と実際のパフォーマンスを視覚化し、制限の主な要因 (メモリー帯域幅や計算能力) を検証して、理想的な最適化ステップを得る方法を示します。

インテル® Advisor は、Fortran、C、C++ ネイティブ/マネージド・アプリケーションが、最新のプロセッサーでパフォーマンスを最大限に発揮できるよう支援するツールを提供します (ツールの一覧は「リリースノート」(英語) を参照)。

- **ベクトル化アドバイザー**は、影響の大きい最適化されていないループを特定し、ベクトル化を妨げている要因と安全にベクトル化を強制できる領域を理解するのに役立つベクトル化の最適化ツールです。また、問題を解決するため、コード別の推奨事項も提供します。
- **ルーフライン解析**は、ハードウェアによって課されるパフォーマンスの制限 (ルーフ) と実際のパフォーマンスを視覚化します。ボトルネックの場所、パフォーマンス最適化候補のループ、ボトルネックの原因の可能性、次の最適化ステップに関する情報を提供します。
- **スレッド化アドバイザー**は、通常の開発作業を中断することなく、スレッド化設計の選択肢を素早く解析、設計、チューニング、確認できるスレッド化設計/プロトタイピング・ツールです。
- **オフロード・アドバイザー**は、GPU オフロードに最適な候補と適していない領域を特定するのに役立つモデル化ツールです。
- **フローグラフ・アナライザー**は、インテル® oneAPI スレッディング・ビルディング・ブロック (インテル® oneTBB) のフローグラフ・インターフェイスを使用するアプリケーションのパフォーマンスを表現して解析するビジュアル・プロトタイプ作成ツールです。

重要
このドキュメントは、インテル® Advisor 2020 製品リリースで更新されました。新しいバージョンのインテル® Advisor では、ワークフロー名、解析タイプ名、ユーザー・インターフェイスが異なる場合があります。

チュートリアルの概要	このチュートリアルでは、以下のトピックついて説明します。 • ルーフライン解析の実行方法 • 最も興味のある**ルーフライン**・グラフのデータに注目する方法 • **ルーフライン**・グラフのデータの解釈方法 • **ルーフライン**・グラフのデータを基に最適化の方針を決定する方法 **注** これは、ベクトル化アドバイザーの上級者向けチュートリアルです。クイックスタートや基本操作の情報を含むベクトル化アドバイザーの初級者向けチュートリアルは、「インテル® Advisor チュートリアル: ベクトル化アドバイザーを使用して C++ コードに効率良い SIMD 並列処理を追加」(英語) を参照してください。
所要時間	20 分
目的	このチュートリアルでは、以下のトピックについて説明します。 • 最も正確で完全なベクトル化アナライザーの解析結果を得られるコンパイラー/リンカーオプションの特定 • ルーフライン解析の実行 • **ルーフライン**・グラフの表示/非表示 • **ルーフライン**・グラフの各種コントロール • **ルーフライン**・グラフを使用した最適化に最も適したループの特定 • インテル® Advisor のスナップショットの読み込み
関連情報	「インテル® Advisor 2017 のルーフライン解析」(英語) は、このチュートリアルの内容を、より概念的に詳しく説明したビデオです。 このチュートリアルの概念と手順は、プログラミング言語に関係なく適用されますが、インテル® ソフトウェア・ドキュメント・ライブラリー (英語) には、別のプログラミング言語のサンプル・アプリケーションを使用したチュートリアルも用意されています。このサイトには、ほかのインテル製品のチュートリアルもあります。 さらに、以下のサイトから追加のリソースを利用できます。 • インテル® Advisor 導入ガイド • インテル® Advisor ユーザー向けルーフライン・リソース • インテル® Advisor

インテル® Advisor チュートリアル: ルーフラインの使用例

ルーフライン解析は、マシンの**達成可能な最大パフォーマンス**に対して、アプリケーションの**実際のパフォーマンス**と**演算強度**を視覚的に示すオプションの解析です。

ルーフライン・グラフは、次の質問に対する答えを提供します。

- 現在のハードウェア・リソースで達成可能な最大パフォーマンスは?
- アプリケーションは現在のハードウェア・リソースで最適に動作するか?
- そうでない場合、最適化の最良の候補は?
- メモリー帯域幅や計算能力が各最適化候補のパフォーマンスを制限しているか?

ルーフライン解析はキャッシュを意識しており、DDR メモリーのトラフィックだけでなく、すべてのメモリー・サブシステムのトラフィックを計測します。シングルスレッド・コードとマルチスレッド・コードの両方に対応しています。

ベクトル化アドバイザーと C++ サンプル・アプリケーション (`roofline_demo_samples`) を使用して、以下の操作を行います。

- ルーフライン解析を実行します。
- 最も興味のある**ルーフライン**・グラフのデータに注目します。
- **ルーフライン**・グラフのデータを解釈します。
- **ルーフライン**・グラフのデータを基に最適化の方針を決定します。

ステップ	説明

ステップ 1:
チュートリアルの準備
	次のいずれかの操作を行います。
	• インテル® Advisor スタンドアロン GUI を使用する場合:
	  ○ ソフトウェア・ツールを入手して、サンプル・アプリケーションを展開します。
	  ○ サンプル・アプリケーションを準備します。
	  ○ インテル® Advisor を起動します。
	  ○ プロジェクトを準備します。
	• Visual Studio* IDE を使用する場合:
	  ○ ソフトウェア・ツールを入手して、サンプル・アプリケーションを展開します。
	  ○ Visual Studio* ソリューションを開きます。
	  ○ ソリューションを準備します。

ステップ 2: **ルーフライン解析の実行**	• ルーフライン解析を実行します。 • **ルーフライン**・グラフを表示/非表示にします。 • **ルーフライン**・グラフのコントロールを理解します。 • **ルーフライン**・グラフのデータを理解します。

ステップ	説明
ステップ 3: **メモリー帯域幅のボトルネックへの対応**	• 結果のスナップショットを開きます。 • 各種コントロールを使用して、最も興味のある**ルーフライン**・グラフのデータに注目します。 • **ルーフライン**・グラフのデータを解釈します。
ステップ 4: **計算能力のボトルネックへの対応**	• 結果のスナップショットを開きます。 • 各種コントロールを使用して、最も興味のある**ルーフライン**・グラフのデータに注目します。 • **ルーフライン**・グラフのデータを解釈します。
ステップ 5: **実際のボトルネックの特定**	• 結果のスナップショットを開きます。 • 各種コントロールを使用して、最も興味のある**ルーフライン**・グラフのデータに注目します。 • **ルーフライン**・グラフのデータを解釈します。

インテル® Advisor チュートリアル: スタンドアロン GUI: サンプル・アプリケーションの準備

インテル® Advisor のスタンドアロン GUI を使用して、浮動小数点パフォーマンスを描画するトレーニング用のサンプル・アプリケーションを試す場合は、以下の手順に従ってください。

ソフトウェア・ツールの入手とサンプル・アプリケーションの展開

次のツールが必要です。

- インテル® Advisor のインストール・パッケージとライセンス
- インテル® C++ コンパイラー・クラシック 15.x 以降、またはサポートされるほかのコンパイラー

  インテル® コンパイラーを使用すると、ベクトル化アドバイザーの**サーベイレポート**からより多くの利点が得られます (最大限に活用するには、バージョン 17.x 以降が必要です)。サポートされるほかのコンパイラーについては、「リリースノート」を参照してください。

- `.zip` ファイルの展開ユーティリティー

まだインテル® Advisor やインテル® C++ コンパイラー・クラシックをお持ちでない場合は、https://software.intel.com/content/www/us/en/develop/tools.html (英語) からダウンロードしてください。

`roofline_demo_samples` サンプル・アプリケーションを設定するには、次の操作を行います。

1. サンプルコードとサンプル結果の両方を、書き込み可能なディレクトリーまたは共有フォルダーにダウンロード (英語) します。
2. サンプルコードとサンプル結果の `.zip` ファイルを展開します。

サンプル・アプリケーションの準備

このチュートリアルで使用するサンプル・アプリケーションを準備します。

1. インテル® C++ コンパイラー・クラシック環境を設定します。例えば、コマンドプロンプトに次のように入力します。

   ```
   "<compiler-install-dir>\bin\compilervars.bat" intel64
   ```

   これで、インストールされているインテル® C++ コンパイラー・クラシックの最新バージョンの環境が設定されます。コンパイラー環境のパス `<compiler-install-dir>` は、インストール・ディレクトリーにより異なります。

コンパイラー環境の設定については、『インテル® C++ コンパイラー・クラシック・デベロッパー・ガイドおよびリファレンス』の「compilervars でコンポーネントの場所を指定」(英語) を参照してください。

2. コマンドプロンプトで、ディレクトリーを展開した `roofline_demo_samples\` に移動します。
3. `build.bat` を実行します。`release` サブディレクトリーが作成され、サンプル・アプリケーションがリリースモードでビルドされます。

独自のアプリケーションをビルドして、正確で完全なベクトル化アドバイザーの結果を生成するには、以下の設定でリリースモードの最適化されたバイナリーをビルドします。

操作	最適な C/C++ 設定
完全なデバッグ情報を生成する (コンパイラーとリンカー)	Linux* コマンドライン: -g Windows* コマンドライン: • /ZI • /DEBUG Microsoft* Visual Studio* IDE: • **[C/C++] > [General] > [Debug Information Format] > [Program Database (/Zi)]** • **[Linker] > [Debugging] > [Generate Debug Info] > [Yes (/DEBUG)]**
適度な最適化を有効にする	Linux* コマンドライン: -O2 以上 Windows* コマンドライン: /O2 以上 Visual Studio* IDE: **[C/C++] > [Optimization] > [Optimization] > [Maximum Optimization (Favor Speed) (/O2)]** 以上
コンパイラー診断を生成する (インテル® コンパイラー 15.0 では必要、16.0 以降では不要)	Linux* コマンドライン: -qopt-report=5 Windows* コマンドライン: /Qopt-report:5 Visual Studio* IDE: **[C/C++] > [Diagnostics [Intel C++]] > [Optimization Diagnostic Level] > [Level 5 (/Qopt-report:5)]**
ベクトル化を有効にする	Linux* コマンドライン: -vec Windows* コマンドライン: /Qvec
SIMD ディレクティブを有効にする	Linux* コマンドライン: -simd Windows* コマンドライン: /Qsimd

操作	最適な C/C++ 設定
OpenMP* ディレクティブに基づくマルチスレッド・コードの生成を有効にする	Linux* コマンドライン: -qopenmp Windows* コマンドライン: /Qopenmp Visual Studio* IDE: **[C/C++] > [Language [Intel C++]] > [OpenMP Support] > [Generate Parallel Code (/Qopenmp)]**

インテル® Advisor の起動

注

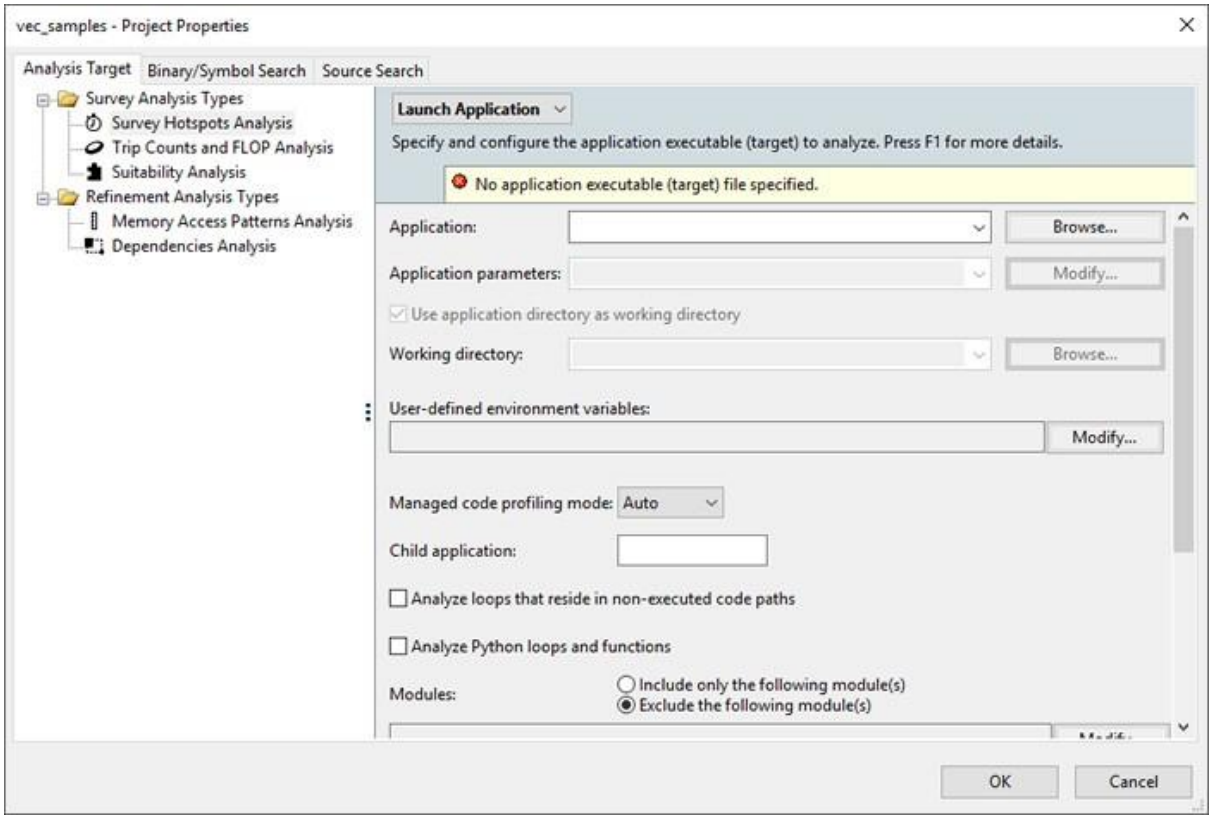
- インテル® Advisor 環境の設定は、`advixe-gui` コマンドでインテル® Advisor スタンドアロン GUI を起動する場合、または `advixe-cl` コマンドでコマンドライン・インターフェイスを実行する場合のみ必要です。
- インテル® Advisor のデフォルトのインストール・パス:
  - `<advisor-install-dir>` は製品をインストールした場所に依存し、`C:\Program Files (x86)\IntelSWTools\` (スタンドアロンの場合) または `C:\Program Files (x86)\Intel\oneAPI\` (インテル® oneAPI ベース・ツールキットの一部としてインストールした場合) 以下にあります。

次のいずれかの方法でインテル® Advisor を起動します。

- advixe-gui コマンドを実行します。
- **[スタート]** メニューから、**[すべてのプログラム] > [Intel oneAPI [version]] > [Intel Advisor [version]]** を選択します。

プロジェクトの準備

1. **[File] > [New] > [Project...]** を選択して (または、**[Welcome]** ページで **[New Project...]** をクリックして)、**[Create a Project]** ダイアログボックスを表示します。
2. **[Project name]** フィールドに `roofline` と入力して、サンプル・アプリケーション・プロジェクトの場所を指定し、**[Create Project]** ボタンをクリックして `roofline.advixeproj` ファイルを作成し、**[Project Properties]** ダイアログボックスを開きます。



3. **[Analysis Target]** タブで **[Survey Hotspots Analysis]** タイプが選択されていることを確認します。
4. **[Application]** フィールドの横にある **[Browse...]** をクリックして、`release` サブディレクトリーにあるビルドした `roofline_demo` バイナリーファイルを選択します。
5. **[Survey Trip Count Analysis]** タイプでは、**[Inherit settings from Survey Hotspots Analysis Type]** チェックボックスがオンになっていることを確認します。
6. **[OK]** ボタンをクリックして、**[Project Properties]** ダイアログボックスを閉じます。

インテル® Advisor チュートリアル: Visual Studio* IDE: サンプル・アプリケーションの準備

インテル® Advisor の Visual Studio* IDE プラグインを使用して、`roofline_demo_samples` サンプル・アプリケーションを試す場合は、以下の手順に従ってください。

ソフトウェア・ツールの入手とサンプル・アプリケーションの展開

次のツールが必要です。

- インテル® Advisor のインストール・パッケージとライセンス
- インテル® C++ コンパイラー・クラシック 15.x 以降、またはサポートされるほかのコンパイラー

  インテル® コンパイラーを使用すると、ベクトル化アドバイザーの**サーベイレポート**からより多くの利点が得られます (最大限に活用するには、バージョン 17.x 以降が必要です)。サポートされるほかのコンパイラーについては、「リリースノート」を参照してください。

- `.zip` ファイルの展開ユーティリティー

まだインテル® Advisor やインテル® C++ コンパイラー・クラシックをお持ちでない場合は、https://software.intel.com/content/www/us/en/develop/tools.html (英語) からダウンロードしてください。

`roofline_demo_samples` サンプル・アプリケーションを設定するには、次の操作を行います。

1. サンプルコードとサンプル結果の両方を、システムの書き込み可能なディレクトリーまたは共有フォルダーにダウンロード (英語) します。
2. サンプルコードとサンプル結果の `.zip` ファイルを展開します。

Visual Studio* ソリューションを開く

1. Visual Studio* IDE を起動します。
2. 必要に応じて、**[View] > [Solution Explorer]** を選択します。
3. **[File] > [Open] > [Project/Solution]** を選択します。
4. **[Open Project]** ダイアログボックスで `roofline_demo_samples.sln` ファイルを開きます。

プロジェクトの準備

このチュートリアルで使用するため、サンプル・アプリケーションを準備します。

1. **[Solution Explorer]** のプロジェクト名が **roofline_demo_samples (Intel C++ [version])** であることを確認します。そうでない場合は、**[Solution Explorer]** で **roofline_demo_samples** プロジェクトを右クリックして、**[Intel Compiler] > [Use Intel C++]** を選択します。
2. Visual Studio* のツールバーで **[Solutions Configuration]** ドロップダウンが **Debug** に設定されている場合は、**Release** に変更します。
3. **[Solution Explorer]** で **roofline_demo_samples** を右クリックして、**[Properties]** を選択します。
4. **[Configuration Properties] > [C/C++] > [General]** を選択します。**[Debug Information Format]** が **[Program Database (/ZI)]** に設定されていることを確認します。
5. **[Configuration Properties] > [C/C++] > [Optimization]** を選択します。**[Optimization]** が **[Maximum Optimization (Favor Speed) (/O2)]** に設定されていることを確認します。
6. **[Configuration Properties] > [C/C++] > [Optimization [Intel C++]]** を選択します。**[Parallelization]** が **[No]** に設定されていることを確認します。
7. **[Configuration Properties] > [C/C++] > [Code Generation]** を選択します。**[Enable Enhanced Instruction Set]** が **[Intel(R) Advanced Vector Extensions 2 (/arch:CORE-AVX2)]** に設定されていることを確認します。
8. **[Apply]** ボタンをクリックしてから、**[OK]** ボタンをクリックします。
9. **[Build] > [Clean Solution]** を選択します。
10. **[Build] > [Rebuild Solution]** を選択します。

注
インテル® Advisor のワークフローと解析結果を表示するには、インテル® Advisor ツールバーの  アイコンをクリックするか、**[Tools] > [Intel Advisor [version]] > [Vectorization and Threading Advisor Analysis]** を選択します。

独自のアプリケーションをビルドして、正確で完全なベクトル化アドバイザーの結果を生成するには、以下の設定でリリースモードの最適化されたバイナリーをビルドします。

操作	最適な C/C++ 設定
完全なデバッグ情報を生成する (コンパイラーとリンカー)	Linux* コマンドライン: -g Windows* コマンドライン: • /ZI • /DEBUG Microsoft* Visual Studio* IDE: • **[C/C++] > [General] > [Debug Information Format] > [Program Database (/Zi)]** • **[Linker] > [Debugging] > [Generate Debug Info] > [Yes (/DEBUG)]**

操作	最適な C/C++ 設定
適度な最適化を有効にする	Linux* コマンドライン: -O2 以上 Windows* コマンドライン: /O2 以上 Visual Studio* IDE: **[C/C++] > [Optimization] > [Optimization] > [Maximum Optimization (Favor Speed) (/O2)]** 以上
コンパイラー診断を生成する (インテル® コンパイラー 15.0 では必要、16.0 以降では不要)	Linux* コマンドライン: -qopt-report=5 Windows* コマンドライン: /Qopt-report:5 Visual Studio* IDE: **[C/C++] > [Diagnostics [Intel C++]] > [Optimization Diagnostic Level] > [Level 5 (/Qopt-report:5)]**
ベクトル化を有効にする	Linux* コマンドライン: -vec Windows* コマンドライン: /Qvec
SIMD ディレクティブを有効にする	Linux* コマンドライン: -simd Windows* コマンドライン: /Qsimd
OpenMP* ディレクティブに基づくマルチスレッド・コードの生成を有効にする	Linux* コマンドライン: -qopenmp Windows* コマンドライン: /Qopenmp Visual Studio* IDE: **[C/C++] > [Language [Intel C++]] > [OpenMP Support] > [Generate Parallel Code (/Qopenmp)]**

インテル® Advisor チュートリアル: ルーフライン解析の実行

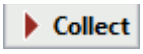
以下のステップを実行します。

- ルーフライン解析の実行
- ルーフライン・グラフの表示/非表示
- ルーフライン・グラフのデータの理解
- ルーフライン・グラフのコントロールの理解

このトピックでは、以下について説明します。

- ルーフライン解析は、サーベイ解析と続けて実行されるトリップカウント & FLOP 解析の組み合わせです。トリップカウント & FLOP 解析の実行には、サーベイ解析の 3 ~ 4 倍の時間がかかる場合があります。
- **ルーフライン**・グラフの各ドットの大きさと色は、各ループ/関数の相対実行時間を表しています。大きな赤いドットは最も多くの時間を費やしており、小さな緑のドットは実行時間が短いことを示します。
- **ルーフライン**・グラフの水平ライン (ルーフライン) は、計算能力の上限を示しており、最適化なしではループ/関数のパフォーマンスをこれ以上高めることはできません。
- **ルーフライン**・グラフの斜めのラインはメモリー帯域幅の上限を示しており、最適化なしではこれ以上のパフォーマンスは期待できません。
- 最上部のルーフラインはマシンの最大能力を示すため、ドットはこれを超えることはありません。また、すべてのループがマシンの最大能力を利用できるわけではありません。
- パフォーマンスを最大限に向上させる最良の候補は、最上部の達成可能なルーフラインから最も離れた大きな赤いドットです。
- **ルーフライン**・グラフには、外観を設定したり、興味のあるデータに注目するための各種コントロールがあります。

ルーフライン解析の実行

[Vectorization Workflow] ペインで **[Run Roofline]** の下にある 

コントロールをクリックして、ターゲット・アプリケーションを 2 回実行します。

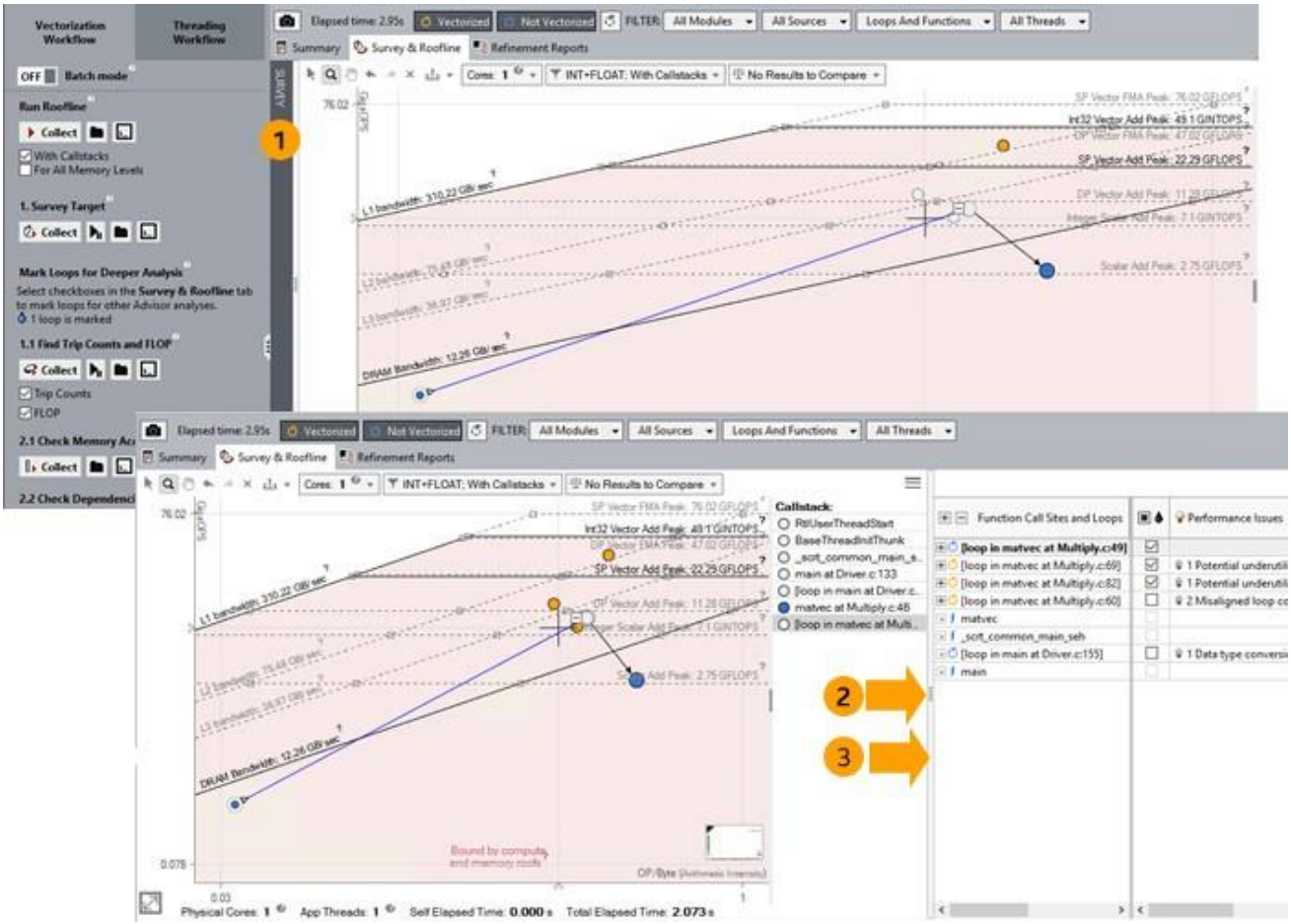
- サーベイ解析でマシンのハードウェア制限を測定し、ループ/関数のタイミング情報を収集します。
- トリップカウント & FLOP 解析で FLOP データを収集します。この収集には、サーベイ解析の 3 ~ 4 倍の時間がかかります。

完了すると、インテル® Advisor は **ルーフライン**・グラフを表示します。

注
Visual Studio* IDE に **[Workflow]** が表示されない場合: インテル® Advisor ツールバーの  アイコンをクリックします (表示には数秒かかる場合があります)。

ルーフライン・グラフの表示/非表示

ルーフライン・グラフを表示/非表示にするコントロールがいくつかあります。



1 クリックして、**ルーフライン**・グラフと**サーベイレポート**の表示を切り替えます。
2 クリックして、**ルーフライン**・グラフと**サーベイレポート**を並べて表示/個別に表示します。
3 ドラッグして、**ルーフライン**・グラフと**サーベイレポート**のサイズを調整します。

ヒント
このチュートリアルの以降の説明では、**ルーフライン**・グラフと**サーベイレポート**を並べて表示します。

ルーフライン・グラフのデータの理解

ルーフライン・グラフは、マシンの**達成可能な最大パフォーマンス**に対して、アプリケーションの**実際のパフォーマンス**と**演算強度**を視覚的に示します。

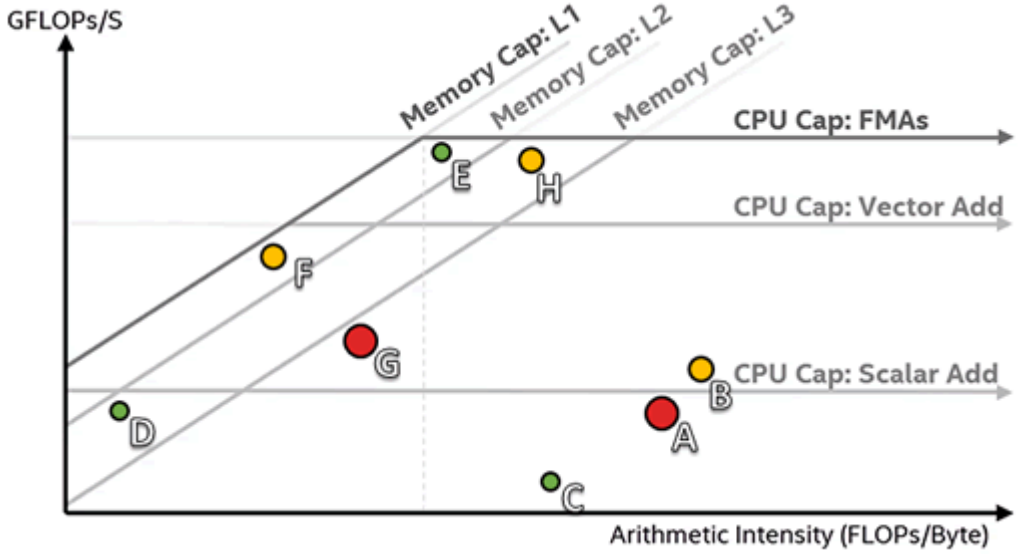
- 演算強度 (x 軸) - ループ/関数のアルゴリズムを基に、CPU/GPU とメモリー間で転送された 1 バイトあたりの浮動小数点操作数 (FLOPs) または整数操作数 (INTOPs) で測定されます。
- パフォーマンス (y 軸) - 1 秒あたりの 10 億浮動小数点演算数 (GFLOPS) または 10 整数演算数 (GINTOPS) で測定されます。

一般に、以下のことが言えます。

- **ルーフライン**・グラフの各ドットの大きさと色は、各ループ/関数の相対実行時間を表しています。大きな赤いドットは最も多くの時間を費やしているため、最適化の最良の候補です。小さな緑のドットは実行時間が短いため、最適化の労力が無駄になるかもしれません。

- **ルーフライン**·グラフの斜めのラインはメモリー帯域幅の上限を示しており、最適化なしではこれ以上のパフォーマンスを達成することはできません。例えば、**L1 Bandwidth** ルーフラインは、ループが**常に** L1 キャッシュにヒットする場合にある演算強度で実行できる最大作業量を示します。ループは、データセットが L1 キャッシュを頻繁にミスする場合、L1 キャッシュの速度の恩恵を受けられず、代わりに低速な L2 キャッシュにより制限されます。L1 キャッシュを頻繁にミスして L2 キャッシュにヒットするループのドットは、**L2 Bandwidth** ルーフラインの下に表示されます。
- **ルーフライン**·グラフの水平ライン (ルーフライン) は、計算能力の上限を示しており、最適化なしではループ/関数のパフォーマンスをこれ以上向上することはできません。例えば、**Scalar Add Peak** は、この状況下でスカラーループが実行可能な加算命令の最大数を示します。**Vector Add Peak** は、この状況下でベクトルループが実行可能な加算命令の最大数を示します。そのため、ベクトル化されていないループのドットは、**Scalar Add Peak** ルーフラインの下に表示されます。
- 最上部のルーフラインはマシンの最大能力を示すため、ドットはこれを超えることはありません。また、すべてのループがマシンの最大能力を利用できるわけではありません。
- ドットと最上部の達成可能なルーフラインの間の距離が大きいほど、パフォーマンス向上の可能性が高くなります。

以下の**ルーフライン**·グラフでは、ループ A と G (大きな赤いドット)、そして B (ルーフから離れている黄色のドット) が最適化の最良の候補です。ループ C、D、E (小さな緑のドット) と H (黄色のドット) は、パフォーマンス向上の余地があまりないか、パフォーマンスに大きな影響を与えるには小さすぎるため候補にはなりません。

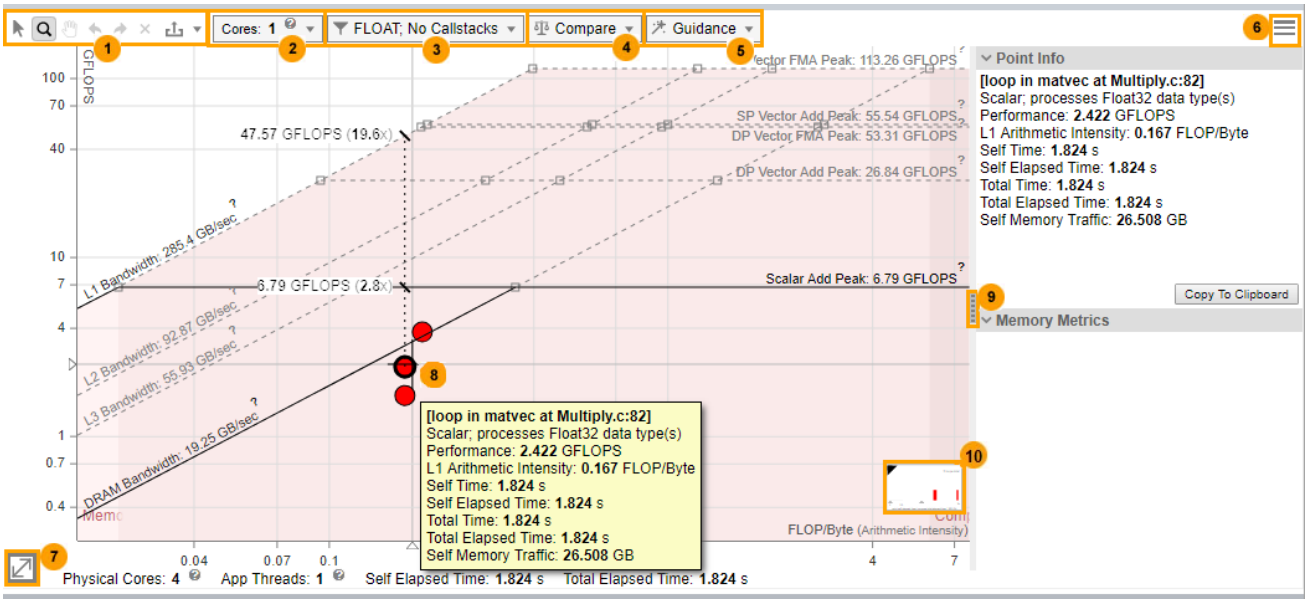


注
ルーフライン·グラフと**サーベイレポート**は同期されています。**ルーフライン**·グラフのドットをクリックすると、**サーベイレポート**で対応するデータ行がハイライト表示され、**サーベイレポート**でデータ行をシングルクリックすると、ループに浮動小数点操作が含まれていれば**ルーフライン**·グラフで対応するドットが点滅します。浮動小数点操作を含まないループは**ルーフライン**·グラフに表示されません。

ルーフライン·グラフで各ルーフライン (ライン)、ピーク (長方形)、ループ (ドット) にマウスのカーソルを合わせると、各要素の説明が表示されます。

ルーフライン・グラフのコントロールの理解

以下を含む、いくつかのコントロールを利用して**ルーフライン**・グラフの重要なデータに注目できます。



1
- **Select Loops by Mouse Rect**: マウスで長方形を描いて、その範囲内にある 1 つ以上のループ/関数を選択します。
- **Zoom by Mouse Rect**: マウスで長方形を描いて、その範囲を拡大/縮小します。マウスホールで拡大/縮小することもできます。
- **Move View By Mouse**: マウスでグラフを前後左右に移動します。
- **Undo/Redo**: 直前のズーム操作を取り消し/やり直します。
- **Cancel Zoom**: デフォルトの倍率にリセットします。
- **Export as x**: インテル® Advisor のビューアーがなくても表示可能な、ダイナミックでインタラクティブな HTML または SVG ファイル形式でグラフをエクスポートします。ドロップダウンで出力ファイル形式を切り替えます。

2 **[Cores]** ドロップダウン・ツールバーを使用して、以下を行うことができます。

- ルーフラインを調整して、ホストマシン上でのコードの実用的なパフォーマンスの上限を確認します。
- シングルスレッド・アプリケーション (または、ランクあたり 1 スレッドの MPI アプリケーションのように、シングルスレッドで実行するように設定されたマルチスレッド・アプリケーション) のルーフを構築します (インテル® Advisor のフィルターを使用して**ルーフライン**・グラフに表示されるループを制御できますが、**ルーフライン**・グラフでは **[Threads]** フィルターをサポートしていません)。

適切な CPU コア数を選択して、ルーフの値を増減できます。

- **1** - コードがシングルスレッドの場合
- **スレッド数と同じまたはほぼ同じコア数** - コードのスレッド数が利用可能な CPU コア数よりも少ない場合

- **最大コア数** - コードのスレッド数が利用可能な CPU コア数よりも多い場合

デフォルトでは、コア数はアプリケーションで使用されるスレッド数 (偶数値) に設定されています。

コードをマルチソケット PC で実行する場合、次のオプションが表示されます。

- アプリケーションがメモリーを 1 つのソケットにバインドする場合、**[Bind cores to 1 socket]** を選択します (デフォルト)。例えば、ソケットごとに 1 ランクの MPI アプリケーションでは、このオプションを使用します。

  注
  1 つのソケットで利用可能な最大コア数を超える CPU コア数を選択する場合、このオプションは無効になることがあります。

- アプリケーションがメモリーをすべてのソケットにバインドする場合、**[Spread cores between all n sockets]** を選択します。例えば、MPI 以外のアプリケーションではこのオプションを選択します。

3
- 浮動小数点操作 (FLOP)、整数操作 (INT)、ミックス (FLOP + INT) の表示を切り替えます。
- **コールスタック付きでルーフラインを収集した場合: ルーフライン**・グラフにコールスタック付きのルーフラインを表示できます。

4 比較のため、インテル® Advisor のほかの結果やアーカイブされていないスナップショットから**ルーフライン**・グラフのデータを表示します。

ドロップダウン・ツールバーを使用して、次の操作を行うことができます。

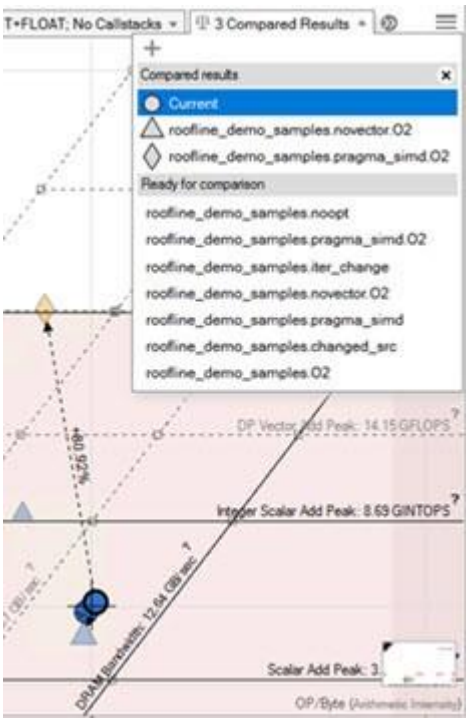
- 結果/スナップショットをロードして、対応するファイル名を **[Compared Results]** 領域に表示します。
- 選択されている結果/スナップショットをクリアして、対応するファイル名を **[Ready for comparison]** 領域に移動します。

  注
  結果/スナップショットをリロードするには、**[Ready for comparison]** 領域のファイル名をクリックします。

- 比較結果をファイルに保存します。

  注
  比較ファイルをアップロードしても、ループ/関数の関係を示す矢印のラインは表示されません。

現在の結果でループ/関数のドットをクリックすると、そのドットとロードされた結果/スナップショットにある対応するループ/関数のドットの関係 (矢印のライン) が表示されます。



5 ルーフライン・グラフにビジュアル・インジケーターを追加して、パフォーマンスの制限や、ループ/関数がメモリー依存か、計算依存か、あるいは両方かなど、データを解釈しやすくします。

ドロップダウン・ツールバーを使用して、次の操作を行うことができます。

- **[Display roof rulers]** チェックボックスをオンにして、ループ/関数から最も近い、最上部のルーフラインへ垂直のラインを表示します。ルーラーを表示するには、ループ/関数にホバーします。ラインと各ルーフが交差する場所では、そのループ/関数のハードウェア・パフォーマンスの上限がラベルで表示されます。
- **すべてのメモリーレベルのルーフラインを収集した場合: [Show memory level relationships]** チェックボックスをオンにすると、表示されているメモリーレベルとルーフ、および選択したループ/関数のドットの関係が視覚的に強調されます。
- **[Show Roofline boundaries]** チェックボックスをオンにして、ループ/関数がメモリー依存か、計算依存か、あるいは両方かを分かりやすくするため、ルーフラインのゾーンを色分けします。

ガイダンスオプションを選択するとプレビュー画像が更新され、変更内容をルーフライン・グラフの外観に反映した状態を確認できます。変更を適用するには **[Apply]** をクリックし、ルーフライン・グラフの外観をデフォルトの状態に戻すには **[Default]** をクリックします。

ループ/関数のドットがハイライト表示されたら、ループ/関数をもう一度ダブルクリックするか、ループ/関数を選択した状態で**スペース**キーまたは **Enter** キーを押して、選択したループ/関数のドットにルーフライン・グラフをズームできます。オリジナルのルーフライン・グラフ表示に戻るまでこのアクションを繰り返します。

ラベル付けされたドットを非表示にするには、別のループ/関数を選択するか、ルーフライン・グラフの空白の領域をダブルクリックします。

6
- **Roofline View Settings:** デフォルトの倍率設定を調整します。
  - **ルーフライン**･グラフごとに最適な倍率
  - すべての**ルーフライン**･グラフに適した倍率
- **Roofs Settings:** ルーフライン (ライン) の表示/非表示と外観を変更します。
  - マルチスレッドではなく、シングルスレッドのベンチマーク結果に基づいてルーフ値を計算できるようにします。
  - **[Visible]** チェックボックスをクリックして、ルーフラインを表示/非表示にします。
  - **[Selected]** チェックボックスをクリックして、ルーフラインの外観 (実線または破線) を変更します。
  - **[Value]** 列でルーフ値を手動で微調整して、コードのハードウェア制限を設定できます。
- **Loop Weight Representation:** ループ/関数の重み (ドット) の外観を変更します。
  - **Point Weight Calculation:** ループ/関数の重み計算の**基本値**を変更します。
  - **Point Weight Ranges**: ループ/関数のドットの**サイズ**、**色**、重みの**範囲**を変更します。**[+]** ボタンをクリックするとループの重み範囲が 2 分割され、**[-]** ボタンをクリックするとループの重み範囲が下の範囲とマージされます。
  - **Point Colorization:** 重み範囲やタイプ (ベクトルまたはスカラー) でループ/関数のドットを色分けします。セルフ時間のないループの色を変更することもできます。

ルーフの設定やドットの重み表現の設定を JSON ファイルに保存したり、カスタム設定をロードできます。

7 数値を使用して拡大/縮小できます。

8 ループ/関数のドットをクリックして、次の操作を行うことができます。

- 黒枠を表示します。
- ループ/関数のメトリックを表示します。
- ほかのウィンドウタブで対応するデータを表示します。

ループ/関数のドットか、**ルーフライン**･グラフの空白の領域を右クリックして、以下のような操作を行うことができます。

- フィルターアウト (ドットを一時的に非表示) やフィルターイン (ほかのすべてのドットを一時的に非表示) により**ルーフライン**･グラフを簡素化したり、フィルターをクリア (すべてのドットを表示) できます。
- クリップボードにデータをコピーします。

9 メトリックペインを表示/非表示にします。

- **[Point Info]** ペインで基本的なパフォーマンス･メトリックを確認します。
- **すべてのメモリーレベルのルーフラインを収集した場合: [Memory Metrics]** ペインで、ループ/関数のキャッシュ利用効率と、ループ/関数が依存しているメモリーレベルを確認します。

10 各ループの重み表現カテゴリーのループの数と割合を表示します。

このチュートリアルの以降の説明では、時間とハードウェア依存性を考慮して、事前に収集された解析結果を使用します。

インテル® Advisor チュートリアル: メモリー帯域幅のボトルネックへの対応

以下のステップを実行します。

- 結果のスナップショットを開く
- 最も興味のあるルーフライン・グラフのデータに注目
- ルーフライン・グラフのデータの解釈

このトピックでは、以下について説明します。

- メモリー帯域幅のボトルネックは、一般にキャッシュの最適化により解決できます。
- **ルーフライン**・グラフの解釈をサポートするため、インテル® Advisor のほかのビューのデータを確認します。

注
時間とハードウェア依存性を考慮して、ここでは事前に収集された解析結果を使用します。

結果のスナップショットを開く

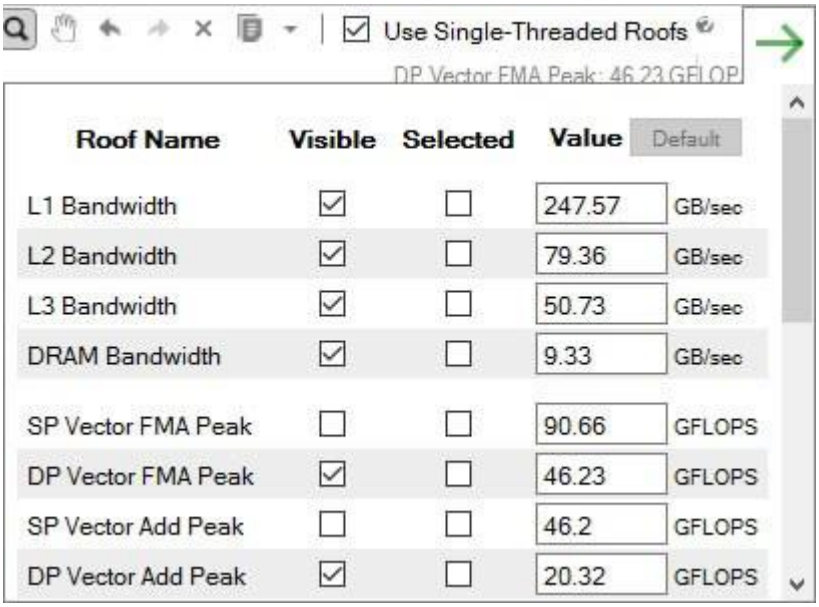
次のいずれかの操作を行います。

- スタンドアロン GUI: **[File] > [Open] > [Result]** から `Result1.advixeexpz` 結果を選択します。
- Visual Studio* IDE: **[File] > [Open]** から `Result1.advixeexpz` 結果を選択します。

最も興味のあるルーフライン・グラフのデータに注目

1. 表示の切り替えを使用して、**ルーフライン**・グラフと**サーベイレポート**を並べて表示します。
2. インテル® Advisor ツールバーの **[Loops And Functions]** フィルター・ドロップダウンから **[Loops]** を選択します。

3. **ルーフライン**・グラフで次の操作を行います。
    - **[Use Single-Threaded Loops]** チェックボックスをオンにします。
    - コントロールをクリックして、すべての *SP...* ループの **[Visibility]** チェックボックスをオンにします (このサンプルコードの変数はすべて倍精度であるため、単精度のルーフラインを非表示にします)。

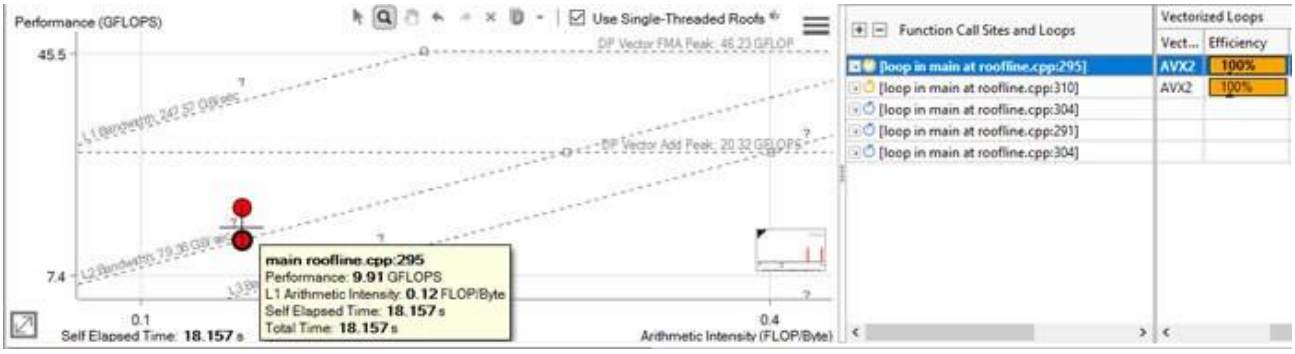


[Point Colorization] セクションで **[Colors of Point Weight Ranges]** を選択して、ランタイム別にドットを色分けします (赤、黄、緑)。

をクリックして変更を保存します。

- コントロールをクリックします。x 軸のフィールドで既存の値を **Backspace** キーで消去し、0.1 と 0.4 を入力します。y 軸のフィールドで既存の値を **Backspace** キーで消去し、7.4 と 45.5 を入力します。 ボタンをクリックして変更を保存します。

ルーフライン・グラフのデータの解釈



この**ルーフライン**・グラフの下のドットは、`roofline.cpp:295` の `main` にあるループを表しており、(画面には表示されていない) **Scalar Add Peak** ルーフラインと **L2 Bandwidth** ルーフラインの上にあります。

なぜこの位置にドットがあるのでしょうか?

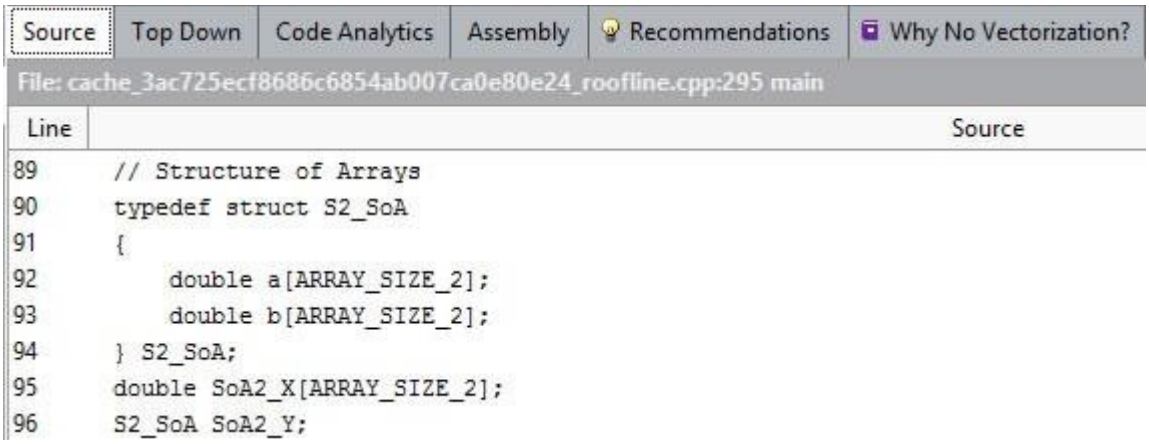
ループのパフォーマンスが L2 キャッシュに関連したメモリー帯域幅のボトルネックにより制限されている可能性が考えられます。

これを検証するため、以下の操作を行います。

1. **サーベイレポート**を確認します。
    - `roofline.cpp:295` の `main` にあるループの **Vectorized Loops/Efficiency** 値が 100% です。

      ベクトル化効率が 100% であるため、このドットは (スクリーンには表示されていない) **Scalar Add Peak** ルーフラインの上にあります。

    - `roofline.cpp:295` の `main` にあるループのデータ行をクリックして、**[Source]** タブに関連するソースコードを表示します。
2. **[Source]** タブでソースコードの 89 - 96 行目にスクロールして、関連するデータ構造体の定義を確認すると、配列構造体 (SOA) であることが分かります。



SOA はベクトル化効率の良いデータレイアウトですが、このサンプルコードではチュートリアルのデータセットが L1 キャッシュに収まらず、L2 キャッシュからのロードが頻繁に発生します (この原因については、動画「インテル® Advisor 2017 のルーフライン解析」(英語) をご覧ください)。

`roofline.cpp:295` の `main` にあるループは、実際に L2 キャッシュに関連したメモリー帯域幅のボトルネックによりパフォーマンスが制限されるため、**L2 Bandwidth** ルーフラインの上にあります。

メモリー帯域幅のボトルネックを解消する方法として、キャッシュ利用を最適化するためコードを再構成することが考えられます。

`roofline.cpp:310` の `main` にあるループはキャッシュ利用が最適化されているため、このループのドット (前出の**ルーフライン**・グラフの上のドット) は **L2 Bandwidth** ルーフラインの上にあります。

1. **サーベイレポート**で、`roofline.cpp:310` の `main` にあるループのデータ行をクリックします。
2. **[Source]** タブでソースコードの 97 - 101 行目にスクロールして、関連するデータ構造体の定義を確認すると、配列構造体配列 (AOSOA) であることが分かります。`roofline.cpp:310` の `main` にあるループが AOSOA データレイアウトの場合、このサンプルコードではチュートリアルのワークロードが 2 つのステップ分割され、それぞれのステップのデータセットは L1 キャッシュに収まります。

インテル® Advisor チュートリアル: 計算能力のボトルネックへの対応

以下のステップを実行します。

- 結果のスナップショットを開く
- 最も興味のあるルーフライン・グラフのデータに注目
- ルーフライン・グラフのデータの解釈

このトピックでは、以下について説明します。

- 演算強度 (**ルーフライン**・グラフの x 軸) = アクセスされるバイトあたりの浮動小数点操作数。すべてのアルゴリズムに演算強度があります。理論的には、このメトリックはアルゴリズム自体の特性であるため、最適化により変化することはありません。つまり、**ルーフライン**・グラフ上のドットは、パフォーマンスの変化に応じて上下には移動しますが、左右に移動することはめったにありません。
- ループを最適化しただけでは、対応するドットを次のルーフラインに移動できません。ループが最適化を上手く利用している必要があります。非効率なベクトル化や孤立した FMA (Fused Multiply Add) を最適化しただけでは十分ではありません。
- 適切な状況下では、データレイアウトとメモリーアクセスを最適化することで、計算能力とメモリー帯域幅の両方の制限を解決できます。
- **[Recommendations]** タブで「how-can-I-fix-this-issue?」にあるコード固有の推奨事項を利用できます。

注
時間とハードウェア依存性を考慮して、ここでは事前に収集された解析結果を使用します。

結果のスナップショットを開く

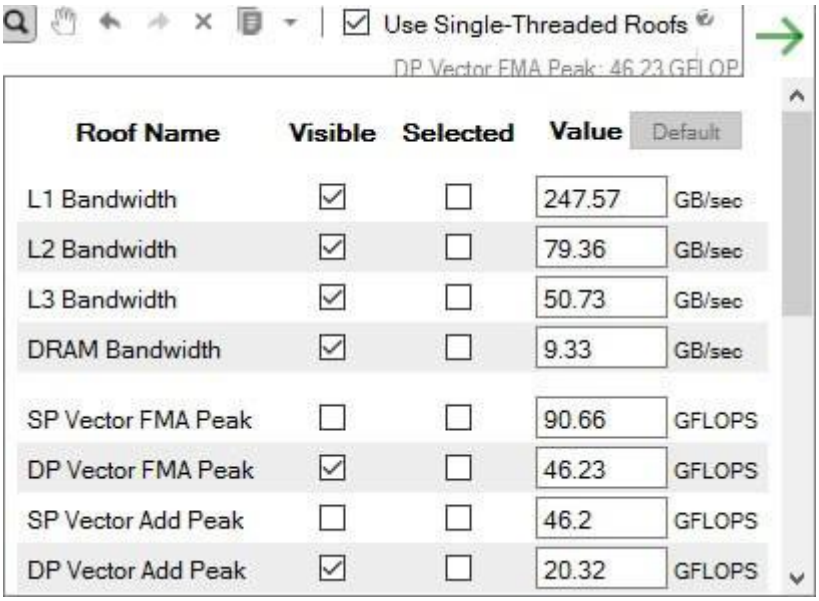
次のいずれかの操作を行います。

- スタンドアロン GUI: **[File] > [Open] > [Result]** から `Result2.advixeexpz` 結果を選択します。
- Visual Studio* IDE: **[File] > [Open]** から `Result2.advixeexpz` 結果を選択します。

最も興味のあるルーフライン・グラフのデータに注目

1. 表示の切り替えを使用して、**ルーフライン**・グラフと**サーベイレポート**を並べて表示します。
2. インテル® Advisor ツールバーの **[Loops And Functions]** フィルター・ドロップダウンから **[Loops]** を選択します。

3. **ルーフライン**・グラフで次の操作を行います。
   - **[Use Single-Threaded Loops]** チェックボックスをオンにします。
   - コントロールをクリックして、すべての *SP...* ループの **[Visibility]** チェックボックスをオンにします (このサンプルコードの変数はすべて倍精度であるため、単精度のルーフラインを非表示にします)。

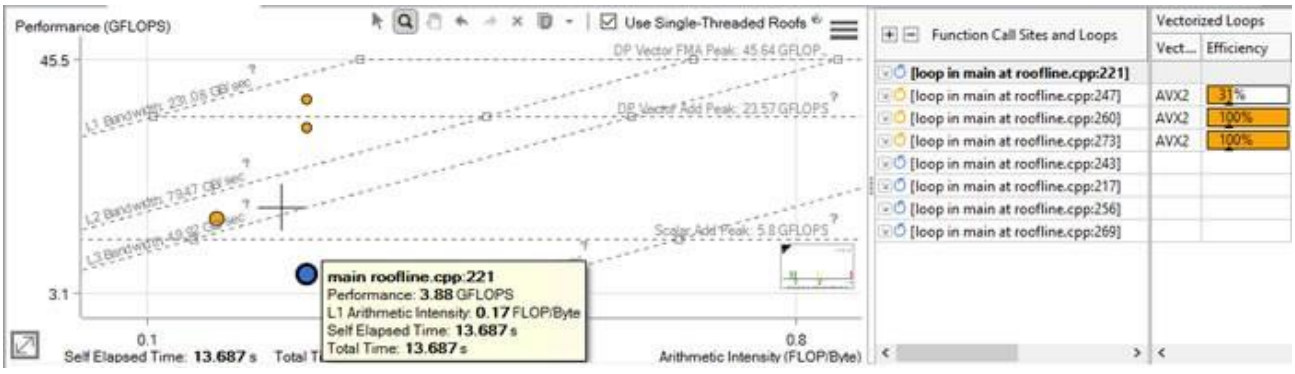


   [Point Colorization] セクションで **[Vectiorized/Scalar]** を選択して、ランタイム (赤、黄、緑) の代わりに、スカラー (青)/ベクトル (オレンジ) で色分けします。

   をクリックして変更を保存します。

   - コントロールをクリックします。x 軸のフィールドで既存の値を Backspace キーで消去し、0.1 と 0.8 を入力します。y 軸のフィールドで既存の値を Backspace キーで消去し、3.1 と 45.5 を入力します。 ボタンをクリックして変更を保存します。

ルーフライン・グラフのデータの解釈



ルーフライン・グラフで、`roofline.cpp:221` の `main` にあるループを表す青いドットは、**Vector Add Peak** ルーフラインと **Scalar Add Peak** ルーフラインの下にあります。

その原因は、ドットの色が示すように、ループがベクトル化されていないためであると考えられます。これは、**サーベイレポート**の最初の列にあるスカラーの状態でも確認できます (青いアイコン = スカラー、オレンジのアイコン = ベクトル)。このチュートリアルでは、ディレクティブを使用してループがベクトル化されないようにしています。インテル® Advisor には、ループがベクトル化されなかった理由を診断するさまざまなツールがあり、実際に問題があった場合はこれらのツールを使用できます。

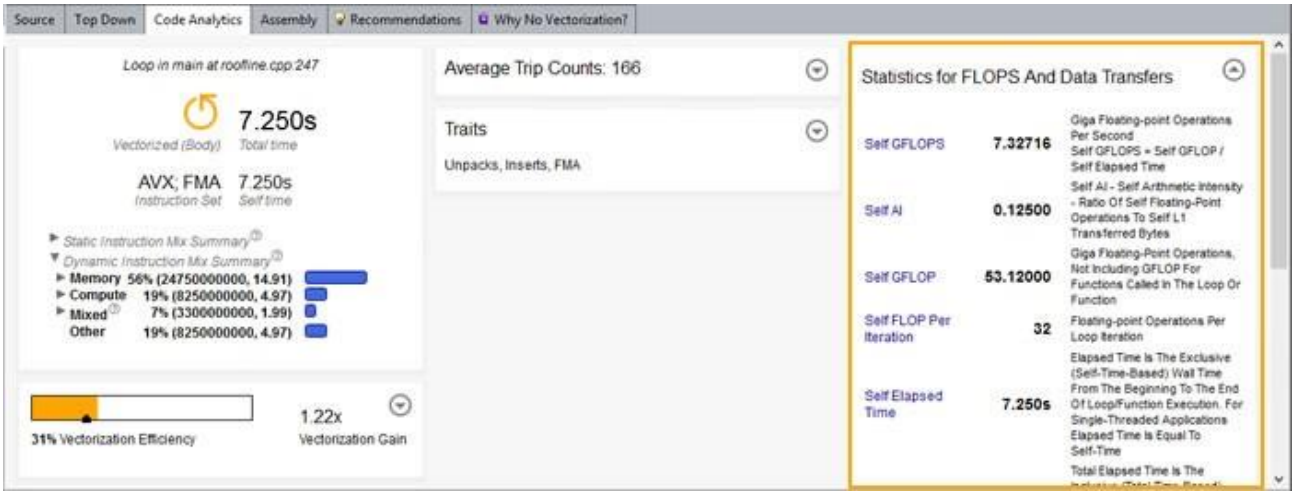
`roofline.cpp:247` の `main` にあるループは、`roofline.cpp:221` の `main` にあるループのベクトル化されたバージョンです。

ルーフライン・グラフでは、このループを表す一番下のオレンジのドットの位置が、**演算強度**軸上で左へ移動しています。これは、理論的にはありえないことです。演算強度は転送されたバイトあたりの浮動小数点操作数を示すもので、すべてのアルゴリズムには演算強度があり、このメトリックはアルゴリズム自体の特性であるため、最適化により変化しません。つまり、**ルーフライン**・グラフ上のドットは、パフォーマンスが向上すると上方向に移動しますが、通常は左右に移動することはありません。

コンパイラーの最適化により、`roofline.cpp:247` の `main` にあるループの演算強度が変わっています。

何が起きたのでしょうか?

[Code Analytics] タブで、青いドットとオレンジのドットの両方のループの **[GFLOPS]** ドロップダウンをチェックします。



両方のループの統計を並べて確認すると、ベクトル化されたループの**反復ごとのセルフ FLOP** に違いが見られます。ループのベクトル長は 4 であるため、8*4=32 でこれは理にかなっています。合計と反復ごとの**データ転送**値が変わっています。256 という値はベクトル化によって説明できません。ベクトル化により変わったのであれば、48*4=192 となり、256 にはなりません。

メトリック	スカラーループ (青いドット)	ベクトル化されたループ (一番下のオレンジのドット)
反復ごとのセルフ FLOP	8	32
データ転送: 合計ギガバイト	318.720	424.960
データ転送: ループ反復ごとのバイト数	48	256

この統計から、コンパイラーによってメモリーアクセスが変更されたことが分かります。**[Code Analytics]** タブからも分かるように、新しいアンパック命令と挿入命令がメモリー計算に影響を与えている可能性があります。

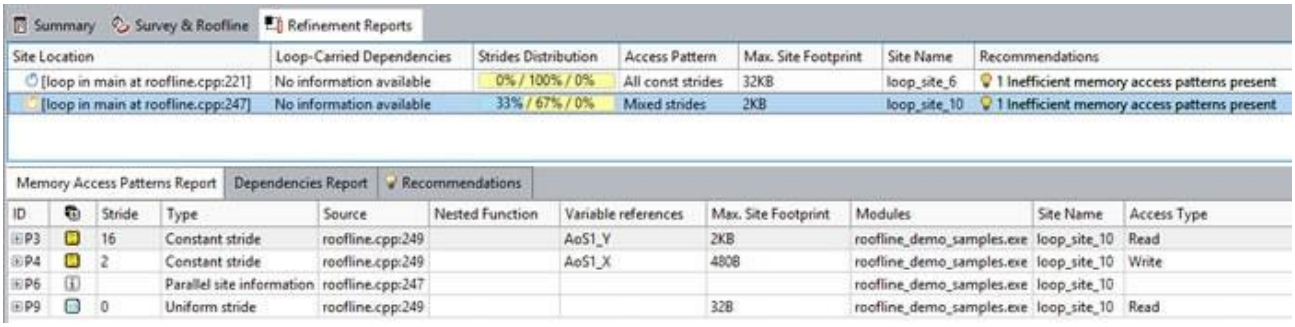
`roofline.cpp:247` の `main` にあるループに関してもう 1 つ気になることは、**ルーフライン**・グラフで、ループを表す一番下のオレンジのドットが **Scalar Add Peak** ルーフラインのすぐ上にあることです。

原因として、ループが効率良くベクトル化されていない可能性が考えられます。

これを検証するため、以下の操作を行います。

サーベイレポートで、**[Vectorized Loops/Efficiency]** の値を確認すると 31% です。ベクトル化が十分ではないため、ループは **Vector Add Peak** ルーフラインに近づくことができません。ループを**効率良く**ベクトル化する必要があります。

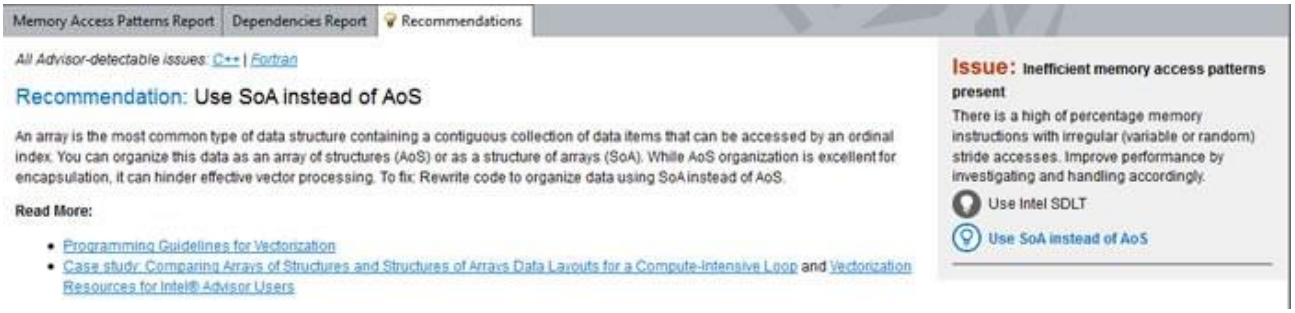
[Efficiency] 値が非常に低い原因として、非効率なメモリーアクセスにより、VPU/SIMD リソースが最大限に活用されていない可能性が考えられます。これは、構造体配列 (AOS) データレイアウトでよくある問題です。**メモリー・アクセス・パターン・レポート**を確認します。



メモリーアクセスの問題が確認されました。ほとんどのメモリーアクセスが均一ストライドではありません (**[Code Analytics]** タブの挿入操作からもこれを確認できます)。これは、ベクトル化の非効率性を助長します。

ベクトル化の効率を改善するため、次の操作を行います。

構造体配列 (AOS) の代わりに、配列構造体 (SOA) データレイアウトを使用するようにコードを再構成します。これは、最適化のアドバイスとして、**[Recommendations]** タブにも記載されています。



`roofline.cpp:260` の `main` にあるループ (真ん中のオレンジのドット) にこの変更を適用しました。

このドットは、**Vector Add Peak** ルーフラインのすぐ下にあります。**サーベイレポート** で、**[Vectorize Loops/Efficiency]** の値を確認すると 100% です。

このドットは L3 キャッシュによって制限されることがなく、L2 キャッシュによる制限もごくわずかであるため、メモリー帯域幅ルーフラインもスキップしています。

このため、SOA データレイアウトに変更することで、計算能力とメモリー帯域幅の両方のボトルネックが解決されました。

最後に、`roofline.cpp:273` の `main` にあるループ (一番上のオレンジのドット) を見てみましょう。ドットが **Vector Add Peak** ルーフラインの上にある理由を考えてみてください。

ヒント
[Assembly] タブで、一番上と真ん中のオレンジのドットで表される 2 つのループの命令を比較してみてください。

インテル® Advisor チュートリアル: 実際のボトルネックの特定

以下のステップを実行します。

- 結果のスナップショットを開く
- 最も興味のあるルーフライン・グラフのデータに注目
- ルーフライン・グラフのデータの解釈

このトピックでは、以下について説明します。

- ドットの上にある最初のルーフラインが必ずしもボトルネックであるとは限りません。ドットの上のすべてのルーフラインが原因となりえます。
- ドットの下のルーフラインがボトルネックになることもありますが、ドットとルーフラインの距離が離れているほど、そのルーフラインがボトルネックになる可能性は低いと考えることができます。
- ドットの上の最初のルーフラインが論理的にボトルネックとは考えられない場合、インテル® Advisor のその他の機能、アプリケーションに関する知識を駆使して、ボトルネックが見つかるまで上方向に順にルーフラインを調査します。
- **ルーフライン**・グラフは、データを入力したら解答が得られるようなユーティリティーではありませんが、コードを最適化するための正しい方向性を示してくれます。

結果のスナップショットを開く

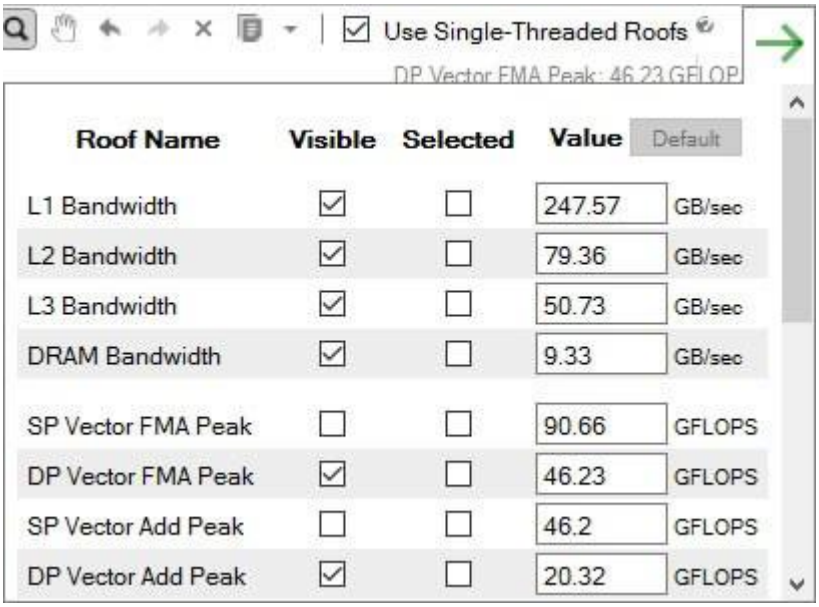
次のいずれかの操作を行います。

- スタンドアロン GUI: **[File] > [Open] > [Result]** から `Result3.advixeexpz` 結果を選択します。
- Visual Studio* IDE: **[File] > [Open]** から `Result3.advixeexpz` 結果を選択します。

最も興味のあるルーフライン・グラフのデータに注目

1. 表示の切り替えを使用して、**ルーフライン**・グラフと**サーベイレポート**を並べて表示します。
2. インテル® Advisor ツールバーの **[Loops And Functions]** フィルター・ドロップダウンから **[Loops]** を選択します。

3. **ルーフライン**・グラフで次の操作を行います。
   - **[Use Single-Threaded Loops]** チェックボックスをオンにします。
   - ☰ コントロールをクリックして、すべての *SP...* ループの **[Visibility]** チェックボックスをオンにします (このサンプルコードの変数はすべて倍精度であるため、単精度のルーフラインを非表示にします)。

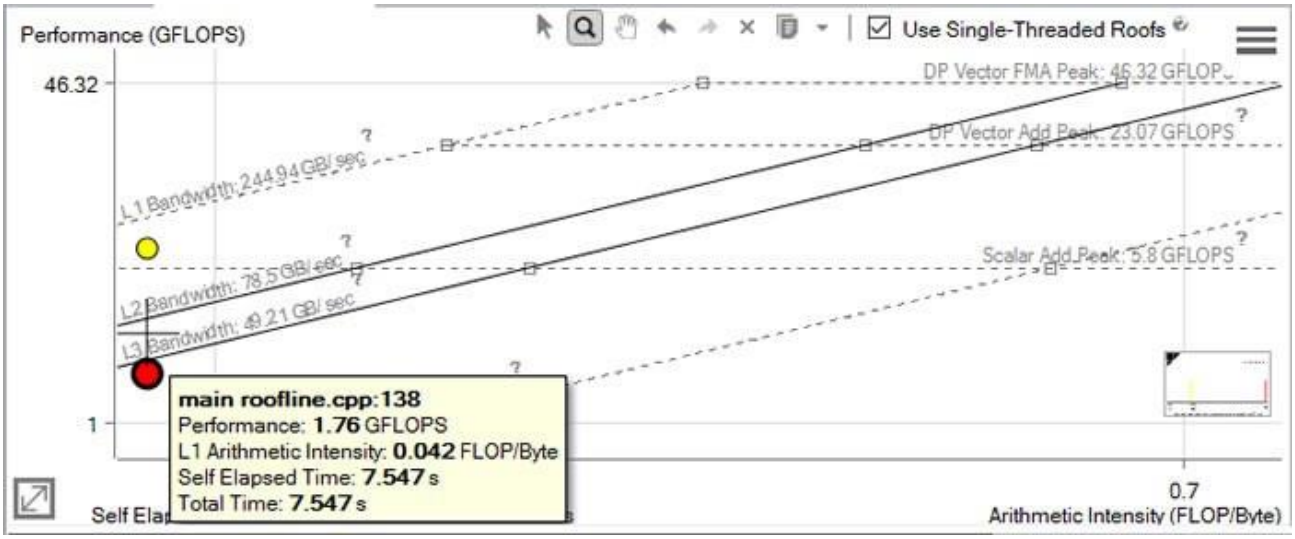


[Point Colorization] セクションで **[Colors of Point Weight Ranges]** を選択して、ランタイム別にドットを色分けします (赤、黄、緑)。

 をクリックして変更を保存します。

-  コントロールをクリックします。x 軸のフィールドで既存の値を **Backspace** キーで消去し、0.05 と 0.7 を入力します。y 軸のフィールドで既存の値を **Backspace** キーで消去し、1.0 と 14.8 を入力します。  ボタンをクリックして変更を保存します。

ルーフライン・グラフのデータの解釈



`roofline.cpp:138` の main にあるループのドット (赤いドット) は、**L3 Bandwidth** ルーフラインの下にあるため、メモリー帯域幅のボトルネックである可能性があります。

しかし、このサンプルコードではデータセットが L1 キャッシュに収まることが分かっています。次の **L2 Bandwidth** ルーフラインも原因ではなさそうです。

その次は **Scalar Add Peak** ルーフラインなので、計算能力のボトルネックの可能性があります。

サーベイレポートを使用して、ループがスカラー (青いアイコン) であることを簡単に確認できます。

ループをベクトル化して、メモリーを最適化しないとどうなるでしょうか? 試してみると、`roofline.cpp:151` の `main` にあるループ (黄色のドット) は、**Scalar Add Peak** ルーフラインの上の **L1 Bandwidth** ルーフラインの近くになりました。

つまり、メモリー帯域幅ではなく、計算能力がボトルネックです。

インテル® Advisor チュートリアル: まとめ

ルーフライン解析は、マシンで**達成可能な最大パフォーマンス**に対して、アプリケーションの**実際のパフォーマンス**と**演算強度**を視覚的に示すオプションの解析です。

ルーフライン・グラフは、次の質問に対する答えを提供します。

- 現在のハードウェア・リソースで達成可能な最大パフォーマンスは?
- アプリケーションは現在のハードウェア・リソースで最適に動作するか?
- そうでない場合、最適化の最良の候補は?
- メモリー帯域幅や計算能力が各最適化候補のパフォーマンスを制限しているか?

ルーフライン解析はキャッシュを考慮しており、DDR メモリーのトラフィックだけでなく、すべてのメモリー・サブシステムのトラフィックを計測します。シングルスレッド・コードとマルチスレッド・コードの両方に対応しています。

このチュートリアルでは、ベクトル化アドバイザーと C++ サンプル・アプリケーション (`roofline_demo_samples`) を使用して、以下の操作を行う方法を紹介しました。

- ルーフライン解析の実行
- 最も興味のある**ルーフライン**・グラフのデータに注目
- **ルーフライン**・グラフのデータの解釈
- **ルーフライン**・グラフのデータを基に最適化の方針を決定

ステップ	要約

ステップ 1: チュートリアルの準備
 	
インテル® Advisor のスタンドアロン GUI を使用する場合: インテル® コンパイラーでリリースモードのターゲット・アプリケーションをビルドし、ターゲットの解析結果を保持する新しいインテル® Advisor プロジェクトを作成して設定します。
 	
Visual Studio* IDE を使用する場合: ターゲット・ソリューションを開いて、インテル® コンパイラーでリリースモードのソリューションをビルドします。
	• ターゲットは、インテル® Advisor が解析可能な実行形式のファイルです。
	• 正確で完全なベクトル化アドバイザーの結果を生成するようにアプリケーションをビルドするには、以下の設定でリリースモードの最適化されたバイナリーをビルドします。
	  ○ /ZI
	  ○ /DEBUG
	  ○ /Qopt-report:5
	  ○ /O2 以上
	  ○ /Qvec
	  ○ /Qsimd
	  ○ /Qopenmp

ステップ	要約
ステップ 2: ルーフライン解析の実行 ルーフライン解析を実行して、**ルーフライン**・グラフのデータとコントロールを理解します。	• ルーフライン解析は、サーベイ解析と続けて実行されるトリップカウント & FLOP 解析の組み合わせです。トリップカウント & FLOP 解析の実行には、サーベイ解析の 3 ~ 4 倍の時間がかかる場合があります。 • **ルーフライン**・グラフの各ドットの大きさと色は、各ループ/関数の相対実行時間を表しています。大きな赤いドットは最も多くの時間を費やしており、小さな緑のドットは実行時間が短いことを示しています。 • **ルーフライン**・グラフの水平ライン (ルーフライン) は、計算能力の上限を示しており、最適化なしではループ/関数のパフォーマンスをこれ以上高めることはできません。 • **ルーフライン**・グラフの斜めのラインはメモリー帯域幅の上限を示しており、最適化なしではこれ以上のパフォーマンスは期待できません。 • 最上部のルーフラインはマシンの最大能力を示すため、ドットはこれを超えることはありません。また、すべてのループがマシンの最大能力を利用できるわけではありません。 • パフォーマンスを最大限に向上させる最良の候補は、最上部の達成可能なルーフラインから最も離れた大きな赤いドットです。 • **ルーフライン**・グラフには、外観を設定したり、興味のあるデータに注目する各種コントロールがあります。
ステップ 3: メモリー帯域幅のボトルネックへの対応 結果のスナップショットを開いて、最も興味のある**ルーフライン**・グラフのデータに注目して、データを解釈します。	• メモリー帯域幅のボトルネックは、一般にキャッシュの最適化により解決できます。 • **ルーフライン**・グラフの解釈をサポートするため、インテル® Advisor のほかのビューのデータを確認します。

ステップ	要約
ステップ 4: 計算能力のボトルネックへの対応 結果のスナップショットを開いて、最も興味のある**ルーフライン**・グラフのデータに注目して、データを解釈します。	• 演算強度 (**ルーフライン**・グラフの x 軸) = アクセスされるバイトあたりの浮動小数点操作数。すべてのアルゴリズムに演算強度があります。理論的には、このメトリックはアルゴリズム自体の特性であるため、最適化により変化しません。つまり、**ルーフライン**・グラフ上のドットは、パフォーマンスの変化に応じて上下には移動しますが、左右に移動することはめったにありません。 • ループを最適化しただけでは、対応するドットを次のルーフラインに移動できません。ループが最適化を**上手く**利用している必要があります。非効率なベクトル化や孤立した FMA (Fused Multiply Add) を最適化しただけでは十分ではありません。 • 適切な状況下では、データレイアウトとメモリーアクセスを最適化することで、計算能力とメモリー帯域幅の両方の制限を解決できます。 • **[Recommendations]** タブで「how-can-I-fix-this-issue?」にあるコード固有の推奨事項を利用できます。
ステップ 5: 実際のボトルネックの特定 結果のスナップショットを開いて、最も興味のある**ルーフライン**・グラフのデータに注目して、データを解釈します。	• ドットの上にある最初のルーフラインが必ずしもボトルネックであるとは限りません。ドットの上のすべてのルーフラインが原因となりえます。 • ドットの下のルーフラインがボトルネックになることもありますが、ドットとルーフラインの距離が離れているほど、そのルーフラインがボトルネックになる可能性は低いと考えることができます。 • ドットの上の最初のルーフラインが論理的にボトルネックとは考えられない場合、常識やインテル® Advisor のその他の機能、アプリケーションに関する知識を駆使して、ボトルネックが見つかるまで上方向に順にルーフラインを調査します。 • **ルーフライン**・グラフは、データを入力したら解答が得られるユーティリティーではありませんが、コードを最適化するための正しい方向性を示してくれます。

法務上の注意書き

インテル® テクノロジーの機能と利点はシステム構成によって異なり、対応するハードウェアやソフトウェア、またはサービスの有効化が必要となる場合があります。

絶対的なセキュリティーを提供できる製品またはコンポーネントはありません。

実際の費用と結果は異なる場合があります。

* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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も許諾するものではありません。

本資料で説明されている製品およびサービスには、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。

インテルは、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適格性、特定目的への適合性、および非侵害性の黙示の保証、ならびに履行の過程、取引の過程、または取引での使用から生じるあらゆる保証を含みますが、これらに限定されるわけではありません。