インテル® Parallel Studio XE Professional Edition

インテル® Advisor ベクトル化アドバイザー 入門ガイド

Version 1.0

内容

1. インテル® Advisor の利用 2
2. ワークフロー 3
STEP1. 必要条件の設定 4
STEP2. インテル® Advisor の起動 5
STEP3. プロジェクトの作成と設定 7
STEP4. ベクトル化に関する情報を取得する 9
STEP5. ループ処理の詳細を取得する 11
ループ処理を詳細解析の対象にする 12
STEP6. メモリー・アクセス・パターンを解析する 13
STEP7. データ間の依存性を解析する 15
3. 補足情報 17
ループ処理ごとの FLOPS を確認する 17
ルーフライン・グラフを確認する 18
4. 更新履歴と商標 19

1. インテル® Advisor の利用

インテル® Advisor 製品は、並列プログラミングを行う上で重要なベクトル化とスレッド化を支援するためのアドバイザー・ツールです。インテル® Advisor のベクトル化アドバイザーは、アプリケーションに実装されているループ処理、および関数を調査して、主に下記の情報を提供します。

- **ループごとにベクトル化の可否と消費した時間**
- **使用された SIMD 拡張命令とベクトル化効率 (ベクトル化されている場合)**
- **ベクトル化されていない理由と、解決に向けた推奨事項の提案 (ベクトル化されていない場合)**
- **ループの繰り返し回数と呼び出し回数、1 ループ処理するために消費する時間**
- **コンパイラーの最適化情報 (ループアンロール、ブロッキング、アラインメント、ベクトル幅の拡張)**
- **データ構造によるメモリー・アクセス・パターン**
- **データ間の依存関係**
- **FLOPS 情報**
- **ルーフライン・グラフ**

さらにこれらの情報をソースコードおよびアセンブリー・コードに対応付けることで、ベクトル化に関する情報をコードレベルで可視化します。

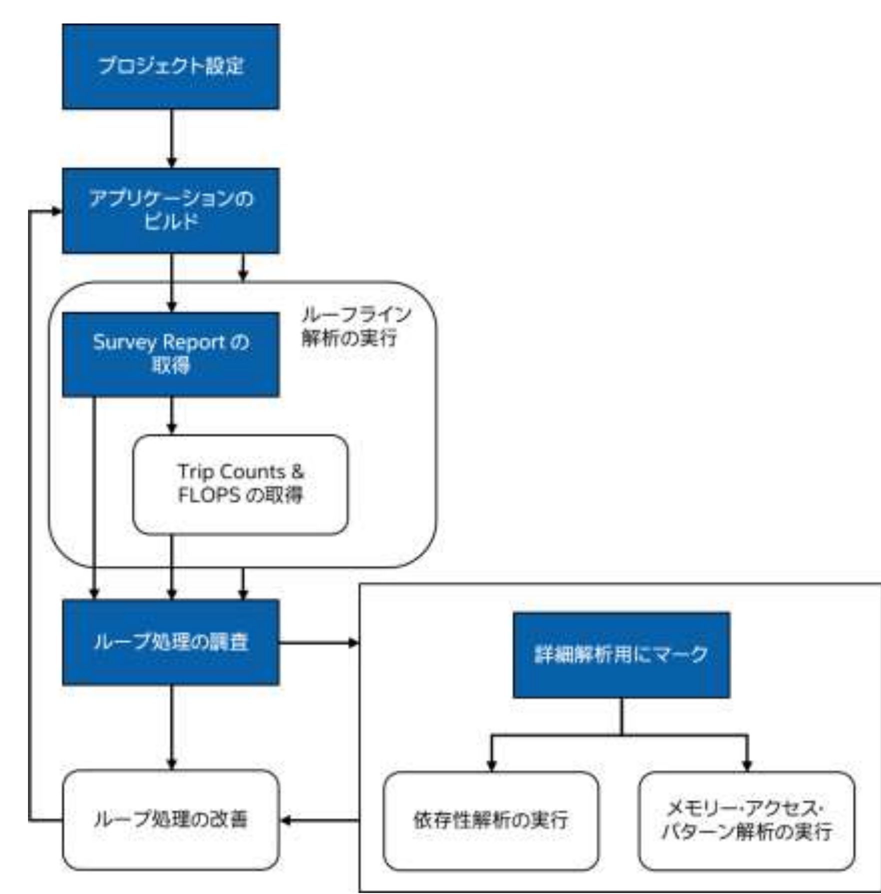
本ガイドはインテル® Advisor のベクトル化アドバイザーに付属する C++ 言語のサンプルプログラムを使用して、ベクトル化アドバイザーから必要な情報を取得するための使用方法と、各画面の見方をチュートリアル形式で説明します。本チュートリアルでは以下のステップを実行します。

STEP1. 必要条件の設定
STEP2. インテル® Advisor の起動
STEP3. プロジェクトの作成と設定
STEP4. ベクトル化に関する情報を取得する
STEP5. ループ情報の詳細を取得する
STEP6. メモリー・アクセス・パターンを取得する
STEP7. データ間の依存性を解析する

本チュートリアルはインテル® Advisor の使用方法の説明を目的としており、ベクトル化の改善手法に関して詳しく説明しません。チュートリアルの前に、ベクトル化の概要と改善例についてベクトル化によるパフォーマンスの向上をご確認ください。

2. ワークフロー

インテル® Advisor のベクトル化アドバイザーは下記左側のワークフローに従ってアプリケーションを調査します。右図の順番に解析をかけていき、必要な情報を取得してアプリケーションのパフォーマンスの改善につなげます。(白いボックスの操作は必須ではありません)



インテル® Advisor が提供するワークフロー

STEP1. 必要条件の設定

1. **サンプルコードを解凍して展開します。**

インストール・フォルダーから、下記のサンプルコードを含む zip ファイルを展開して任意の場所に配置します。

[サンプルコード]
<インストール・フォルダー>\Advisor\samples\en\C++\Vector_Tutorial_Introduction.zip

サンプルコードの一式から example.cpp を使用します。

[example.cpp]
<展開先フォルダー>\Vector_Tutorial_Introduction\Vectorization_Advisor\example.cpp

2. **サンプルコードをコンパイルして実行ファイルを作成します。**

解析対象のプログラムは下記のオプションを使用して、各種最適化情報の追加と、最適化を行いベクトル化されるようにビルドします。

必要な操作	Windows*	Linux*
デバッグ情報	/Zi	-g
最適化レベル	/O2 以上	-O2 以上
自動ベクトル化有効	/Qvec (デフォルト)	-vec (デフォルト)
OpenMP* ディレクティブ有効	/Qopenmp	-qopenmp
simd ディレクティブ有効	/Qsimd	-simd

インテル® Advisor は Windows* と Linux* 向けに同様の GUI 環境を提供します。本チュートリアルでは、Windows* 上で下記のコマンドでコンパイルしたプログラムを使用します。

[Windows*] icl /Zi /O2 /QxHOST /Qopenmp /debug:inline-debug-info example.cpp /Fevec.exe
[Linux*] icc -g -O2 -xHOST -qopenmp -debug inline-debug-info example.cpp -o vec.out

サンプルコードには Visual Studio* 向けにプロジェクト・ファイル一式が含まれています。インテル® Advisor を Visual Studio* 上に統合した環境がある場合は、Visual Studio* を起動してサンプルコードのプロジェクトを開き、ビルドしてください。「**STEP3. プロジェクトの作成と設定**」をスキップすることができます。

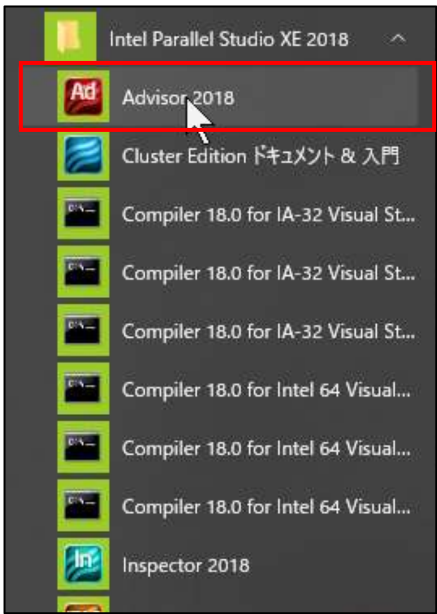
STEP2. インテル® Advisor の起動

OS と開発環境にあわせて 3 パターンの方法で起動することができます。

- ✓ **スタートメニューからインテル® Advisor GUI を起動する (Windows* のみ)**
- ✓ **Visual Studio* 上からインテル® Advisor GUI を起動する (Windows* のみ)**
- ✓ **コンソールコマンドからインテル® Advisor GUI を起動する (Linux* 向け)**

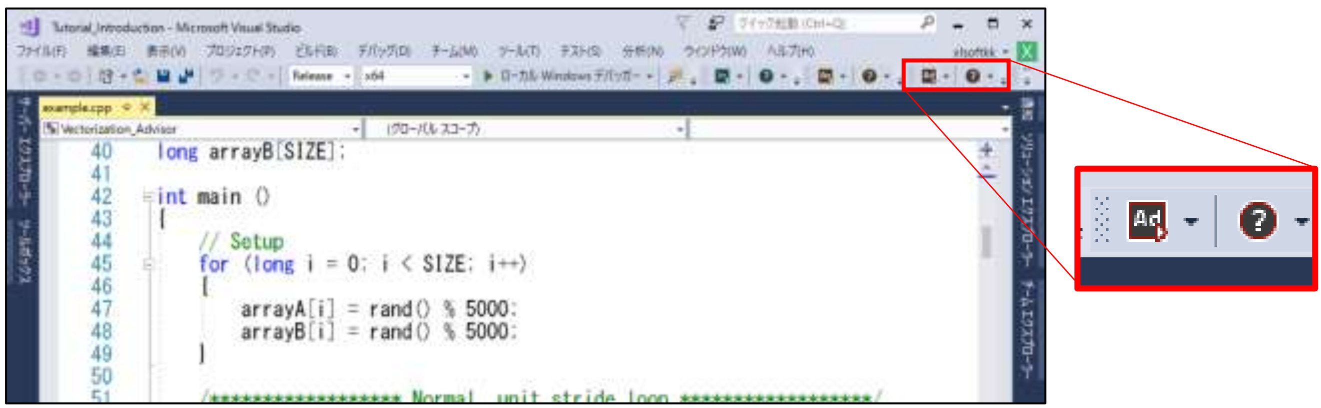
■ スタートメニューからインテル® Advisor GUI を起動する (Windows* のみ)

[Intel Parallel Studio XE 2018] > [Advisor 2018] を開く、もしくは Windows* 検索で "advisor" を入力して [Advisor 2018] を開きます。



■ Visual Studio* からインテル® Advisor GUI を起動する (Windows* のみ)

上部メニューの を開きます。



■ コンソールコマンドからインテル® Advisor GUI を起動する (Linux* 向け)

advixe-vars.sh を実行して GUI を起動するために必要な環境変数を設定します。

$source <インストール・ディレクトリー>/advisor/bin/advixe-vars.sh

インテル® Advisor GUI を起動します。

$advixe-gui

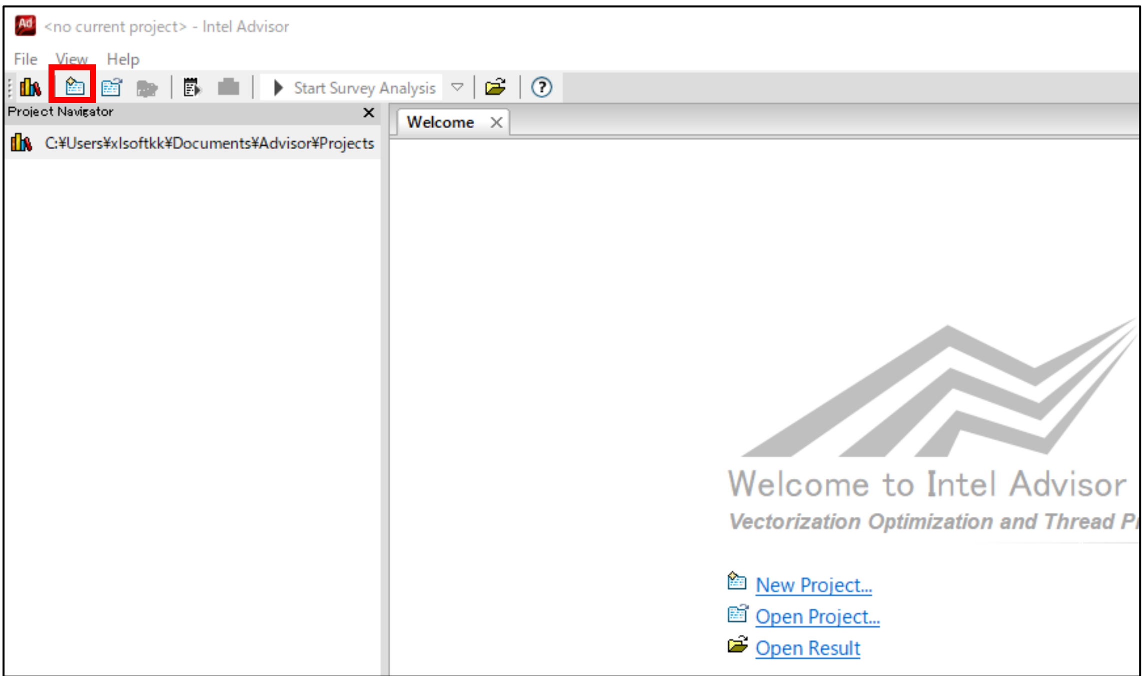
デフォルトのインストール設定では下記のパスに配置されます。

/opt/intel/advisor/bin/advixe-vars.sh

STEP3. プロジェクトの作成と設定

インテル® Advisor は 1 つのアプリケーションに対して、1 つのプロジェクトを作成して管理します。設定画面では解析対象の実行ファイルと、コンパイル時に生成されるシンボルファイル (オブジェクト・ファイル)、ソースファイルを必ず指定します。

1. インテル® Advisor のプロジェクトを作成するために、 を選択します。

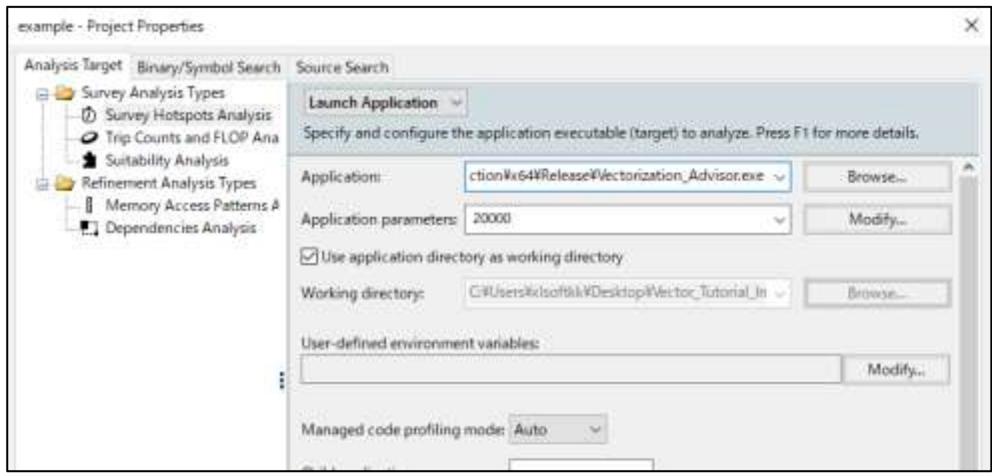


2. [Project name] にプロジェクト名を入力して [Create Project] を選択します。

   本ガイドでは vec_intro をプロジェクト名に指定します。プロジェクト名には任意の名前を入力することが可能です。

3. インテル® Advisor のプロジェクト・プロパティー画面から必要な設定を追加します。

■ **インテル® Advisor GUI (Windows*、Linux* 共通)**

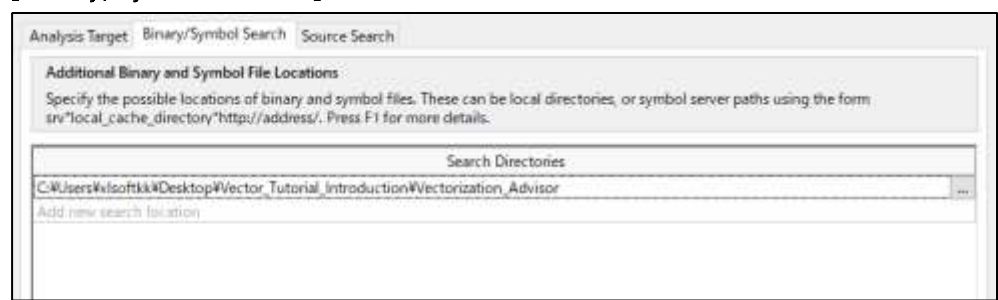


(1) [Application:] に実行ファイルを指定します。サンプルコードの vec.exe または vec.out を指定します。

MPI アプリケーションの解析は GUI から実行できません。MPI アプリケーションの解析方法は「インテル® Advisor コマンドラインと MPI」を参照してください。

(2) [Binary/Symbol Search] タブに移動してシンボルファイルの配置先を指定します。

[Binary/Symbol Search] タブでは以下の画面が確認できます。

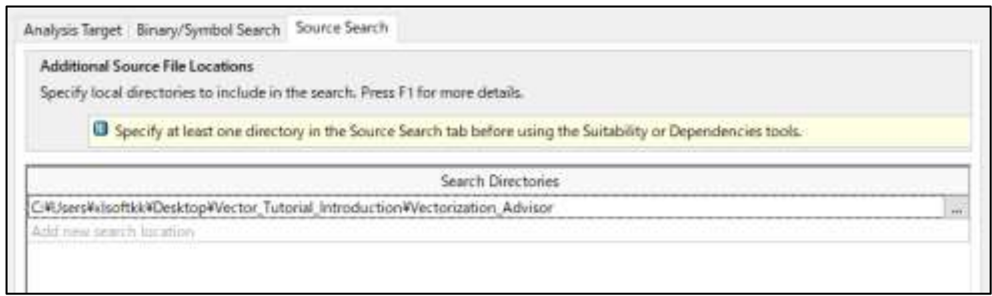


サンプルコードの example.pdb (Windows*)/vec.out (Linux*) が配置されているフォルダーを [...] から指定します。

[example.pdb] <展開先フォルダー>\Vector_Tutorial_Introduction\Vectorization_Advisor\

(3) [Source Search] タブに移動してソースファイルの配置先を指定します。

[Source Search] タブでは以下の画面が表示されます。



サンプルコードの example.cpp が配置されているフォルダーを指定します。

[example.cpp] <展開先フォルダー>\Vector_Tutorial_Introduction\Vectorization_Advisor\

(4) [OK] を選択してプロジェクトの設定を完了します。

■ Visual Studio* 統合環境 (Windows* のみ)

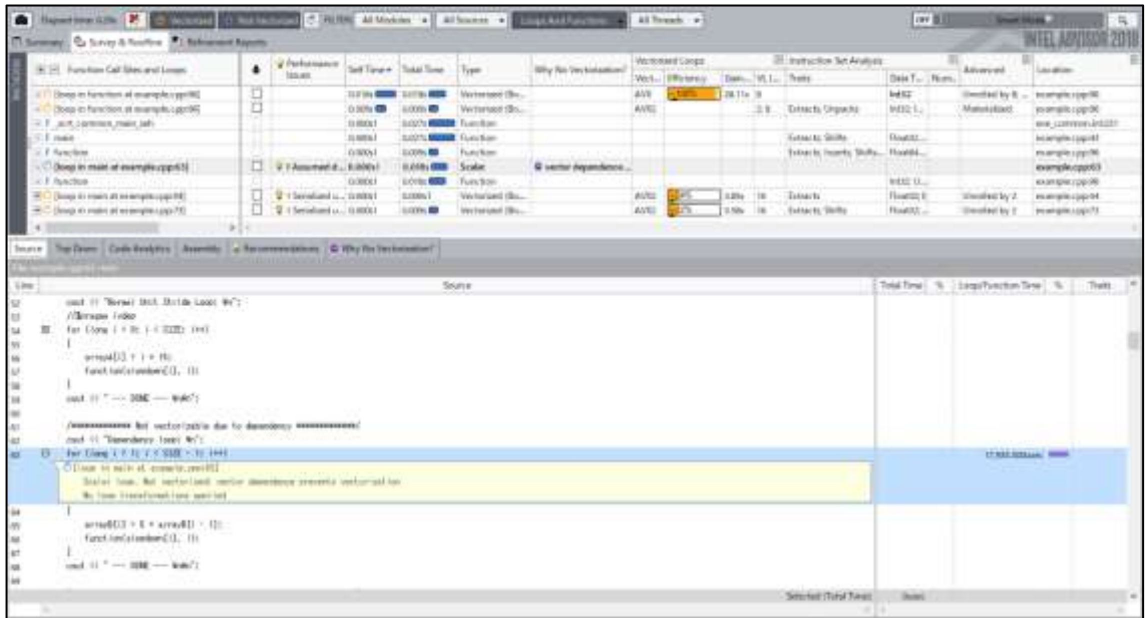
Visual Studio* のプロジェクト設定を継承するため、基本的にインテル® Advisor のプロジェクトを別途作成、設定する必要はありません。変更したい場合は、[プロジェクト] > [Intel Advisor 2018 Project Properties...] を選択します。インテル® Advisor GUI と同じプロジェクト設定の画面を確認することができます。

STEP4. ベクトル化に関する情報を取得する

1. Survey Target の Collect を選択して、解析対象のプログラムの解析を開始します。

解析を開始すると、解析対象のプログラムが実行され、インテル® Advisor がサンプリングを行います。プログラムの終了後、インテル® Advisor は収集した情報をファイナライズして、Survey Report を表示します。

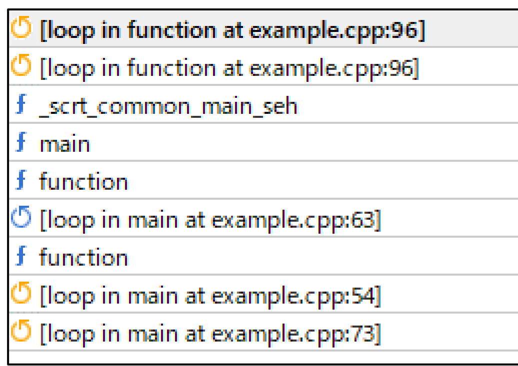
2. [Survey & Roofline] タブの情報を確認します。



[Survey Target] の実行により、主に下記の情報を確認することが可能です。

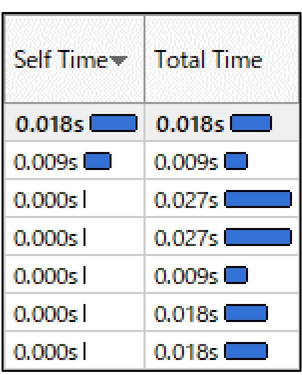
ループ処理のベクトル化の可否

アプリケーションで実行されたループ処理を Vectorized Not Vectorized で色分けして表示します。f は関数を示します。



ループ処理で消費した時間

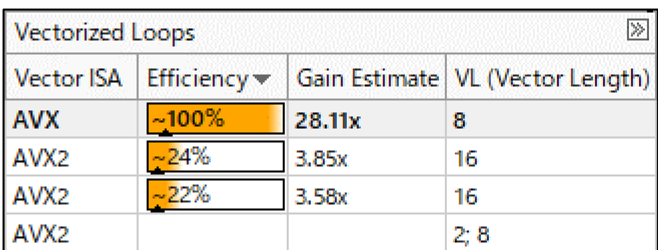
ループ処理内部の計算処理によって消費された時間を [Self Time] として計上します。
ループ処理を抜けるまでにかかった時間を [Total Time] として計上します。



Self Time	Total Time
0.018s	**0.018s**
0.009s	0.009s
0.000s	0.027s
0.000s	0.027s
0.000s	0.009s
0.000s	0.018s
0.000s	0.018s

使用された SIMD 拡張命令とベクトル化効率

スカラー処理と比較したスピードアップと、実行効率を 0% ~ 100% で表示します

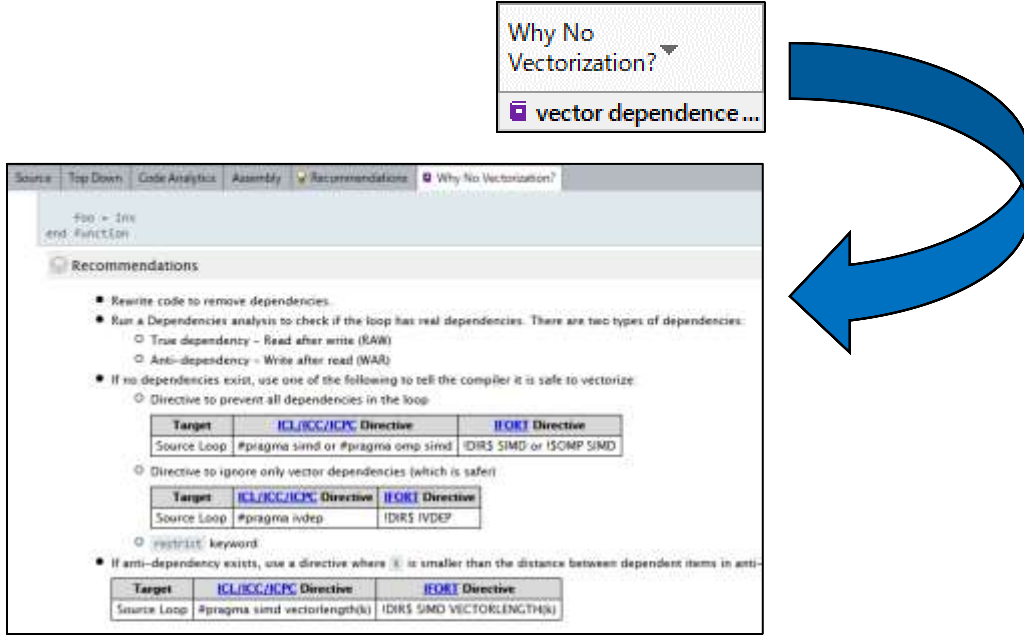


Vectorized Loops

Vector ISA	Efficiency	Gain Estimate	VL (Vector Length)
AVX	**~100%**	**28.11x**	**8**
AVX2	~24%	3.85x	16
AVX2	~22%	3.58x	16
AVX2			2; 8

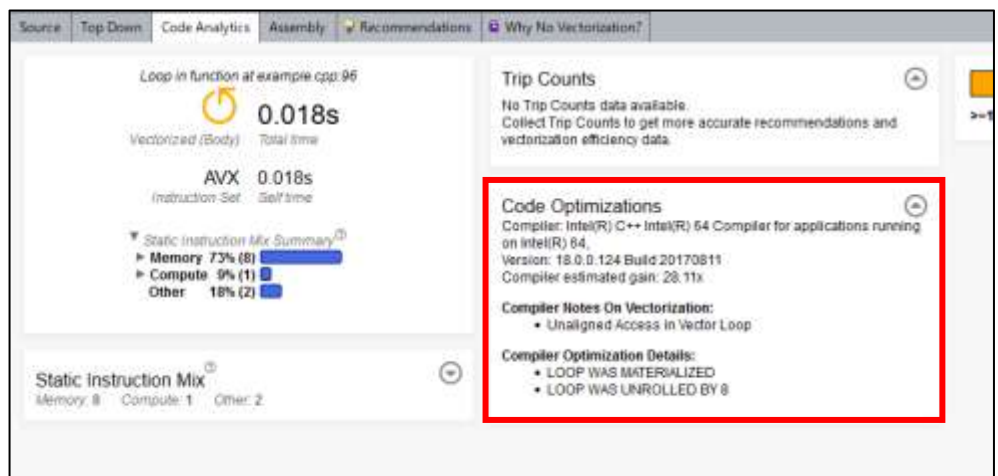
ベクトル化されていない理由と、解決に向けた推奨事項の提案

[Why No Vectorization?] の項目に表示されている を選択すると、ベクトル化されていない理由と、改善につなげるためのヒントを提示します。



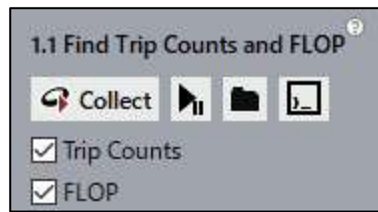
コンパイラーの最適化情報

[Code Analytics] タブを選択すると、ループ処理に対してコンパイラーが適用した最適化情報と注意事項を確認できます。



STEP5. ループ処理の詳細を取得する

1. [Find Trip Counts and FLOP] 解析の [Trip Counts] と [FLOP] をチェックします。



2. [Find Trip Counts and FLOP] の Collect を選択して解析を開始します。

   Survey Target 同様にプログラムが起動します。インテル® Advisor のファイナライズが完了するまで待機します。

3. [Survey & Roofline] タブの情報を確認します。

   [Find Trip Counts and FLOP] の解析により主に下記の情報を確認できます。

ループ処理の繰り返し回数、呼び出し回数

[Trip Counts] 項目にループ処理の繰り返し回数や、呼び出し回数、1 ループあたりに消費する時間を表示します。

Trip Counts					
Average	Min	Max	Call Count	Iteration Duration	Loop Instance Total Time
4	**4**	**4**	**140149**	**< 0.001s**	**< 0.001s**
256	256	256	28125	< 0.001s	< 0.001s
			1		0.027s
			1		0.027s
			28125		< 0.001s
74998	74998	74998	1	< 0.001s	0.018s
			140149		< 0.001s
65151	65151	65151	1		
2343; 3	2343; 3	2343; 3	1; 1	< 0.001s	
1171; 7	1171; 7	1171; 7	1; 1	< 0.001s	

ループ処理を詳細解析の対象にする

「メモリー・アクセス・パターンを解析する」、「データ間の依存性を解析する」は解析にかかるオーバーヘッドが大きく、複雑なアプリケーションではすべてのループ処理を対象にすべきではありません。解析機能に応じて、詳細解析 (Deeper Analysis) のフラグを設定して、適切なループ処理から必要な情報を取得することを推奨します。

解析によって得られる情報は異なるため、詳細解析対象となりえる典型的なループ処理として下記のパターンが考えられます。

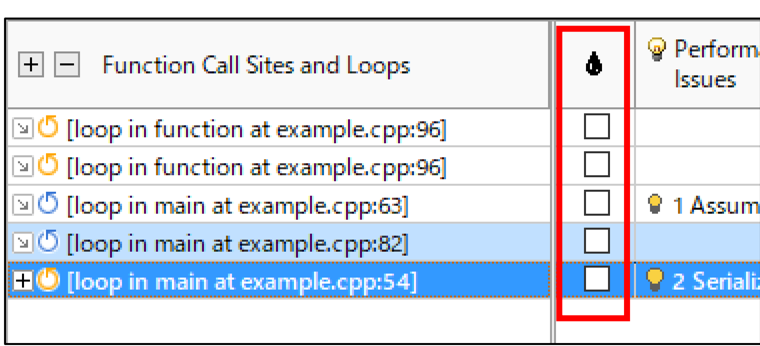
- **共通**
  Self Time が大きく、呼び出し回数が多い

本チュートリアルは解析結果の紹介を目的としており、プログラムの実行時間が短いため、Self Time と呼び出し回数は考慮していません。

- **メモリー・アクセス・パターン解析:**
  ベクトル化されているが効率性 (Efficiency) が低い
  ベクトル化されていない理由に、効率に関するメッセージが表示されている

- **依存性解析**
  ベクトル化されていないループ処理であり、ベクトル化されていない理由として依存性が含まれる場合があるとのメッセージが表示されている

詳細解析の対象とするためには、[Survey & Roofline] タブに表示される 項目をチェックします。



STEP6. メモリー・アクセス・パターンを解析する

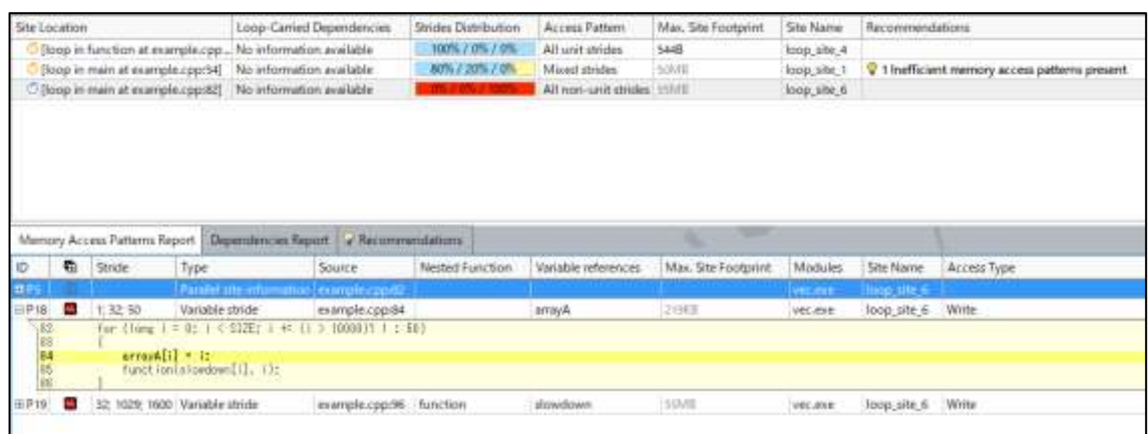
1. example.cpp:54、example.cpp:82、および example.cpp:96 の 項目をチェックします。example.cpp:96 は効率良くベクトル化されているループ処理の例として確認します。

メモリー・アクセス・パターンの解析は詳細解析の対象になるため、[Survey & Roofline] タブの画面から解析対象のループ処理をマークします。詳細解析の対象は「ループ処理を詳細解析の対象にする」を確認してください。

2. [Check Memory Access Patterns] の Collect を選択して解析を開始します。

インテル® Advisor のファイナライズが完了して画面が更新されるまで待機します。

3. [Refinement Reports] を確認します。example.cpp:82 を選択してください。



[Check Memory Access Patterns] 解析では主に下記の情報を確認することが可能です。

変数ごとのメモリー・アクセス・パターン

[Memory Access Patterns Report] 項目にはループ処理で使用されている変数が、ユニットストライドなアクセスをしているか、非ユニットストライドなアクセスを行っているかどうか、アクセスパターンを確認することができます。

	Stride	Type	Source	Nested Function	Variable references	Max. Site Footprint	Modules
		Parallel site information	example.cpp:82				vec.exe
	1; 32; 50	Variable stride	example.cpp:84		arrayA	219KB	vec.exe
	32; 1029; 1600	Variable stride	example.cpp:96	function	slowdown	55MB	vec.exe

```
for (long i = 0; i < SIZE; i += (i > 10000)? 1 : 50)
{
    arrayA[i] = i;
    function(slowdown[i], i);
}
```

ユニットストライドに関する情報は「ランタイム・パフォーマンスの理解」や、「コンパイラー最適化入門: 第 4 回 自動ベクトル化はどんな時に行われるか」を確認してください。

表示されるアクセスパターンの種類は以下の 5 種類です。一般的に表の下側のアクセスパターンはベクトル化の効率を著しく低下させます。効率の良いベクトル化は **Unit stride** なアクセスが理想的です。

アクセスパターン	意味
Uniform stride 0	ループに関係なく同じメモリーアドレスを参照している。
Unit stride (stride 1)	1 ループあたり 1 要素分隣接する要素にメモリーアクセスを行っている。
Constant stride (stride N)	1 ループあたり N 要素分ジャンプしてメモリーアクセスを行っている。
Irregular stride	ループごとに異なる要素分をジャンプしてメモリーアクセスを行っている。不規則なメモリーアクセス。
Gather (irregular) stride	**Irregular stride** をコンパイラーが特殊な命令を使用してベクトル化している。効率の悪いベクトル化。

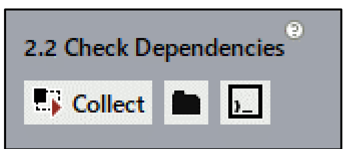
STEP7. データ間の依存性を解析する

1. example.cpp:63、example.cpp:82、および example.cpp 96 の 項目をチェックします。example.cpp:96 はベクトル化可能なループ処理の例として確認します。

Function Call Sites and Loops		Performance Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops Vector ISA	Efficiency	Gain Es
[loop in function at example.cpp:96]	☑		0.031s	0.031s	Vectorized (Bo...		AVX	100%	28.11x
[loop in function at example.cpp:96]	☐		0.010s	0.010s	Vectorized (Bo...		AVX2		
_scrt_common_main_seh			0.000s	0.042s	Function				
main			0.000s	0.042s	Function				
function			0.000s	0.010s	Function				
[loop in main at example.cpp:63]	☑	1 Assumed d...	0.000s	0.016s	Scalar	vector dep...			
function			0.000s	0.031s	Function				
[loop in main at example.cpp:82]	☑		0.000s	0.016s	Scalar	loop contr...			
[loop in main at example.cpp:54]	☐	2 Serialized u...	0.000s	0.010s	Vectorized (Bo...		AVX2	[illegible]	3.85x

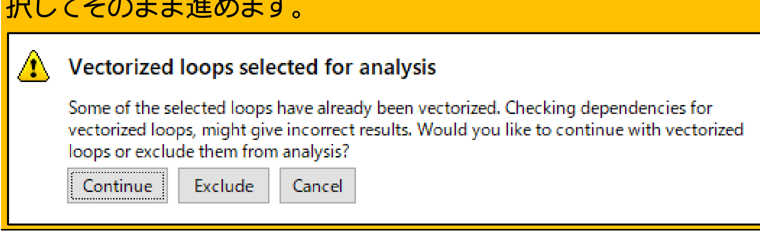
データ間の依存性の解析は詳細解析の対象になるため、[Survey & Roofline] タブの画面から解析対象のループ処理をマークします。詳細解析の対象とする処理は「ループ処理を詳細解析の対象にする」を確認してください。

2. [Check Dependencies] の Collect を選択して解析を開始します。

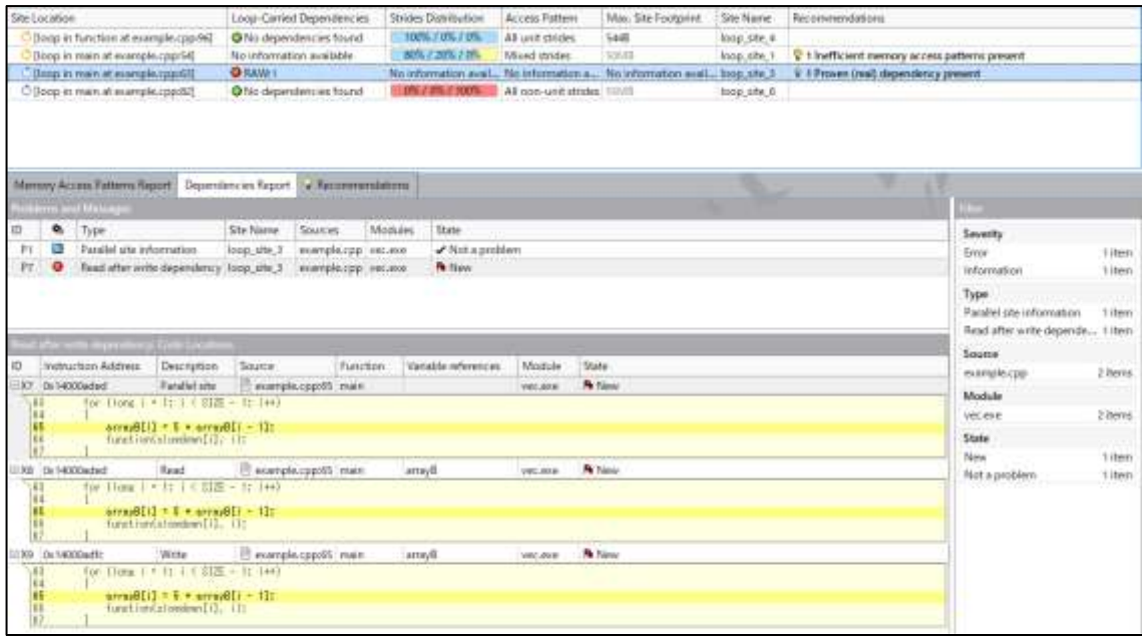


インテル® Advisor のファイナライズが完了して画面が更新されるまで待機します。

ベクトル化されたループを選択しているため、警告が表示されることがあります。表示された場合は [Continue] を選択してそのまま進めます。



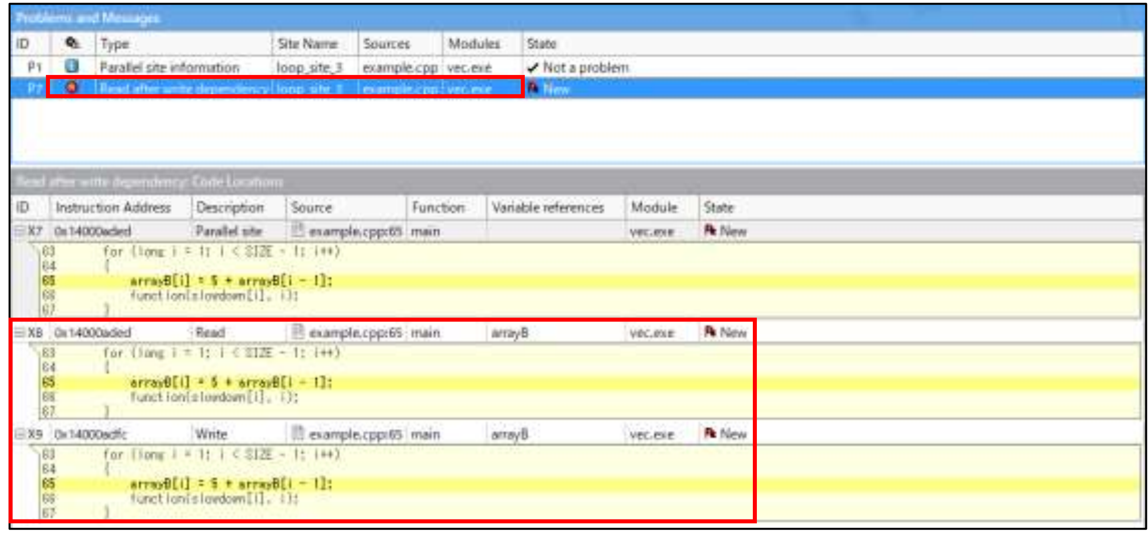
3. [Refinement Reports] を確認します。example.cpp:63 を選択してください。



[Check Dependencies] 解析では主に下記の情報を確認することができます。

依存関係を持つ変数

[Dependencies Report] タブの内容には、変数間に存在する依存関係を表示します。この例では arrayB のアクセスに依存関係が存在していることを表示しています。



データ間の依存関係に関する情報は、「ループをベクトル化するための条件」や、「インテル® C++ コンパイラーのベクトル化ガイド」の「4.2 データ依存」項目を確認してください。

ベクトル化を適用した場合、ベクトル化の対象となる演算処理の実行順序は固定されません。一般的に、ある値を計算する時に使用するデータは計算前に決定されている必要があるため、前のループで演算した値を利用する処理は基本的にベクトル化できません。依存関係を持つ処理をベクトル化するためにはアルゴリズム・レベルでの変更が必要となるケースがあります。

3. 補足情報

チュートリアルで紹介していない機能について記載します。補足情報に記載している画像の情報は、異なるサンプルコード Vector_Tutorial_Vectorization_and_Data_Size.zip をチュートリアルと同じ手順にて解析を行うと確認することができます。

ループ処理ごとの FLOPS を確認する

「STEP5. ループ処理の詳細を取得する」に記載したように、[Find Trip Counts and FLOP] の [FLOP] をチェックし、 Collect を選択して解析を開始すると、解析結果にループごとの FLOPS を確認することができます。

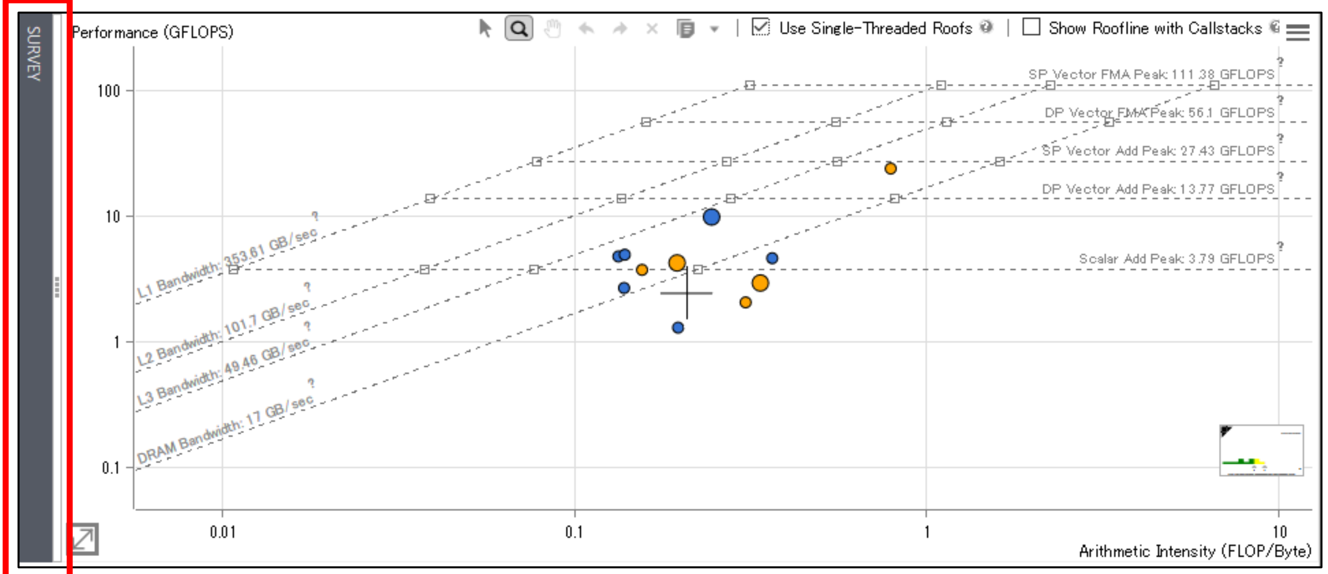
FLOPS

Self GFLOPS	Self AI	Self GFLOP	Self Memo...	Self Elapse...	Total Elaps...
24.011	0.78947	1.500	1.900	0.062s	0.672s
9.894	0.24456	5.244	21.444	0.530s	0.530s
4.959	0.13859	1.627	11.740	0.328s	0.469s
4.802	0.13291	2.100	15.800	0.437s	0.437s
4.650	0.36364	0.800	2.200	0.172s	0.969s
4.266	0.19512	2.400	12.300	0.563s	0.563s
3.757	0.15517	1.350	8.700	0.359s	0.359s
2.954	0.33645	1.800	5.350	0.609s	0.609s
2.695	0.13793	0.800	5.800	0.297s	0.297s
2.072	0.30556	0.550	1.800	0.266s	1.187s
1.304	0.19643	0.550	2.800	0.422s	1.311s

ルーフライン・グラフを確認する

[Run Roofline] の Collect を選択して解析を開始します。

[Run Roofline] は Survey Target 解析と Find Trip Counts and FLOP 解析を行うバッチ処理です。そのため、プログラムが 2 回起動します。ルーフライン・グラフは必要な情報が揃えば、自動的に作成されるため、Survey Target 解析と Find Trip Counts and FLOP 解析を手動で別々に実行しても作成されます。



[Survey & Roofline] タブからルーフライン・グラフを確認することができます。画像赤枠の [SURVEY] を選択すると [Survey Target] 解析で取得した情報に切り替えることができます。

ルーフライン・グラフに関する情報は、「インテル® Advisor のルーフライン」、「コールスタックを利用したルーフライン」を確認してください。

4. 更新履歴と商標

バージョン 1.0	初版。 製品バージョン 2018 UP1 使用

Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。

* 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。