インテル® コンパイラーによる最適化やベクトル化についてインテル® Advisor によるループ解析

エクセルソフト株式会社

概要

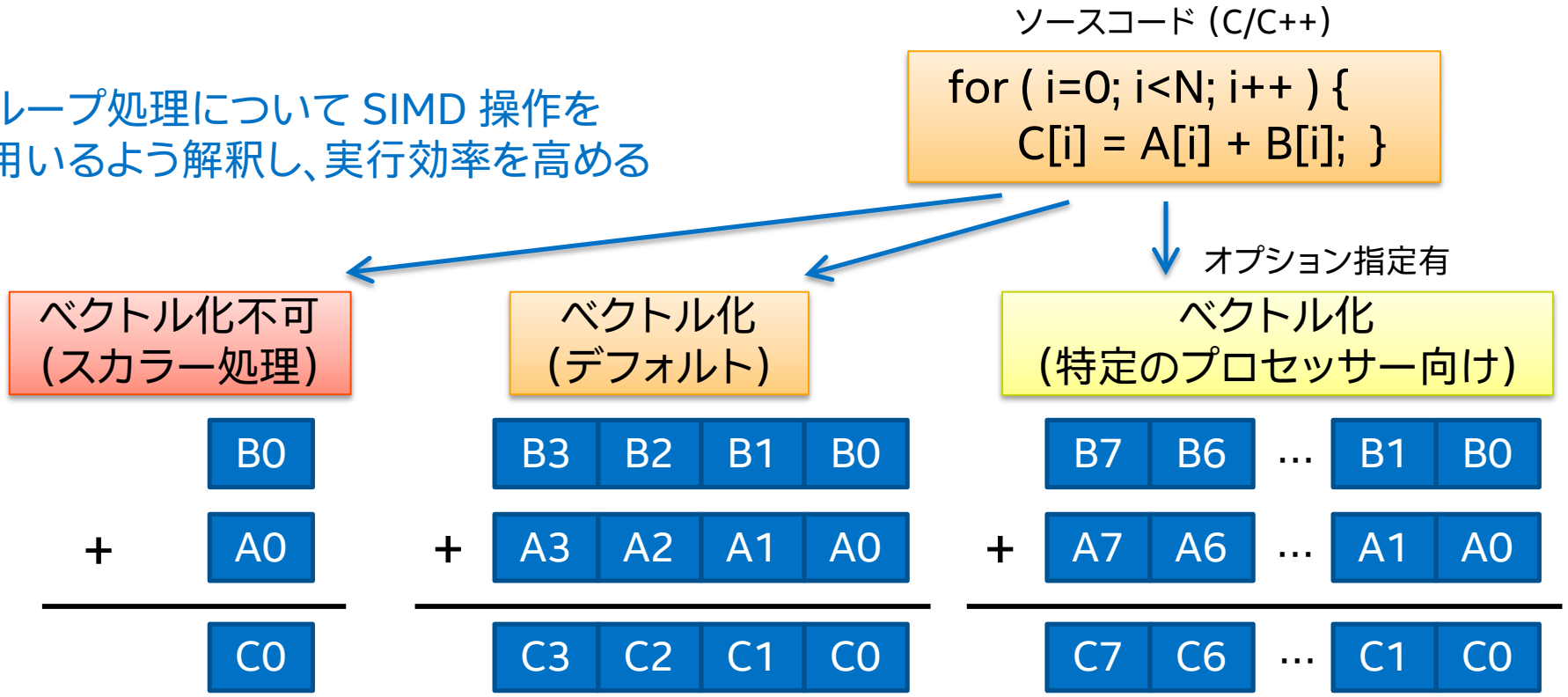
ベクトル化と命令セット

ベクトル化を支援するインテルコンパイラーの機能

インテル Advisor のベクトル化アドバイザーによる最適化

ベクトル化

ループ処理について SIMD 操作を
用いるよう解釈し、実行効率を高める



SIMD : Single Instruction Multiple Data

インテル® コンパイラーの自動ベクトル化機能

デフォルトで有効(-O2 以上の最適化オプション)

- /O1 ではベクトル化が行われない
- ループが多用されるアプリケーションでは、/O3 (-O3) を試す

```
icc  source.c
icpc  source.cpp
ifort  source.f
ifort  source.f90
```

≒価

```
icc  -O2  -msse2  source.c
icpc  -O2  -msse2  source.cpp
ifort  -O2  -msse2  source.f
ifort  -O2  -msse2  source.f90
```

コンパイルまたは実行に失敗する場合には、/Od (-O0) で
コンパイラーの最適化の影響を外し、問題を切り分ける

スカラー演算コード vs. SIMD 演算コード

```
for( i=0; i<MAX; i++ )
     c[i] = (a[i] + b[i]) * c[i];
```

※ MAX = 1024 とする

インテル® コンパイラー

自動ベクトル化

```
..B1.2:
      movsd    (%rsi,%rax,8), %xmm0
      addsd    (%rdx,%rax,8), %xmm0
      mulsd    (%rdi,%rax,8), %xmm0
      movsd    %xmm0, (%rdi,%rax,8)
      incq     %rax
      cmpq     $1024, %rax
      jl       ..B1.2
```

1024 回のループ

```
..B1.2:
      vmovupd   (%rsi,%rax,8), %ymm0
      vaddpd    (%rdx,%rax,8), %ymm0, %ymm1
      vmulpd    (%rdi,%rax,8), %ymm1, %ymm2
      vmovupd   %ymm2, (%rdi,%rax,8)
      addq      $4, %rax
      cmpq      $1024, %rax
      jb        ..B1.2
```

4 要素ずつ
256 回のループ

addsd : Scalar Double-fp Add

vaddpd : Packed Double-fp Add

インテルコンパイラーのベクトル化オプション

使用する SIMD 命令セットの指定

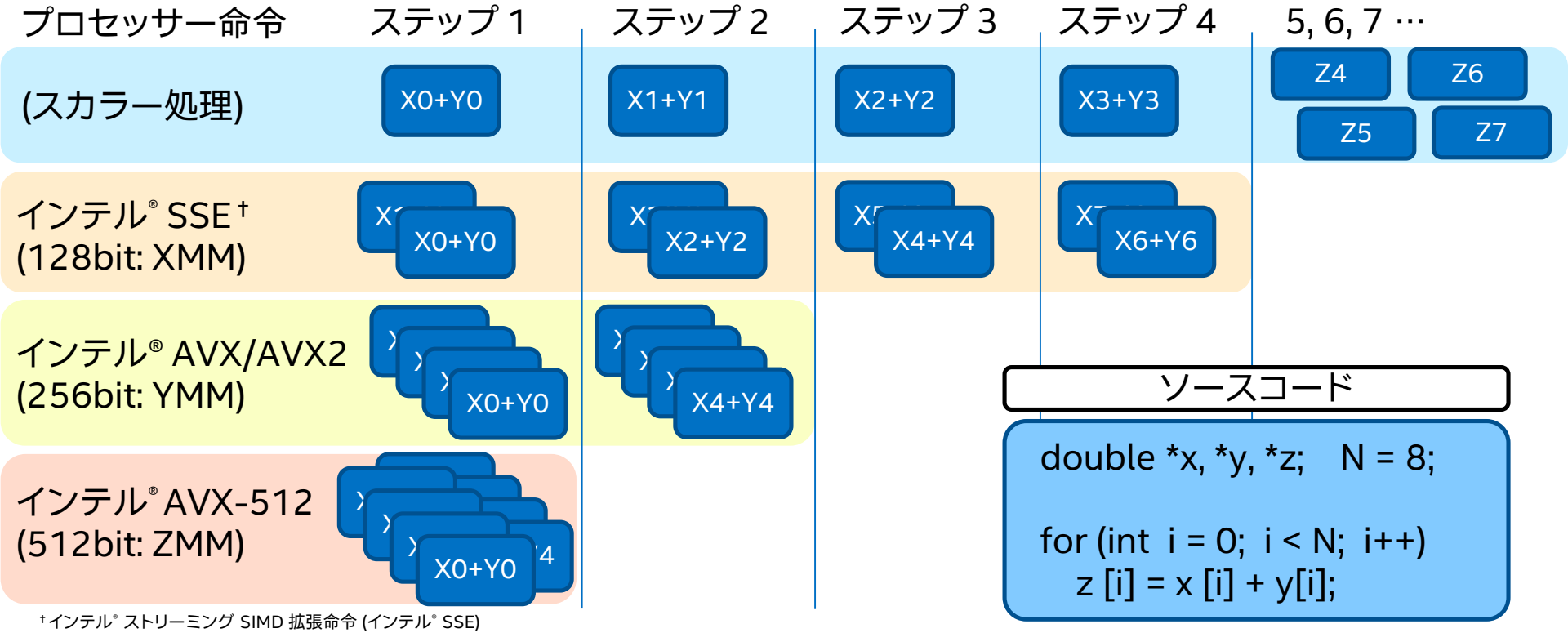
- /Qx<SIMD 命令セット, …>
- /Qax<SIMD 命令セット, …>
- /QxHOST

<SIMD 命令セット> に指定可能なキーワード:

SSE2, SSE3, SSSE3, SSE4.1, SSE4.2,	(128 bit)
ATOM_SSSE3, ATOM_SSE4.2,	(128 bit)
AVX, CORE-AVX-I, CORE-AVX2,	(256 bit)
MIC-AVX512, CORE-AVX512, COMMON-AVX512	(512 bit)

命令セットはパフォーマンス、動作に影響する

利用可能なレジスタ幅の拡張により演算効率が向上する



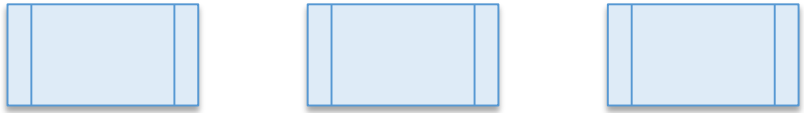
†インテル® ストリーミング SIMD 拡張命令 (インテル® SSE)

命令セットはパフォーマンス、動作に影響する

すべてのソースファイルに同じオプションを指定することを推奨

…しかし困難な場合がある …

オブジェクト・ファイルが提供される、事情によりオプションが制限される



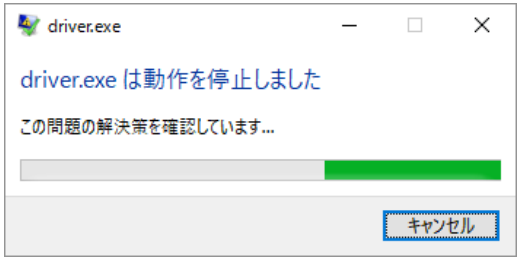
main.c　　func1.c　　func2.obj (または特定のオプションが指示される)

> icl main.c func1.c func2.obj /QaxCORE-AVX2 /Fetest_code.exe
　または
> icl func2.c /QxCORE-AVX512 /c
> icl main.c fuinc1.c func2.obj /QaxCORE-AVX2 /Fetest_code.exe

実行すると?

命令セットはパフォーマンス、動作に影響する

問題となる: ターゲットの実行環境が最新のプロセッサーではない、インテル® AVX/AVX2 からインテル® AVX-512 への移行中のコードが含まれる…




```
[xlsoftkk@localhost work]$ ./a.out
Illegal instruction (コアダンプ)
[xlsoftkk@localhost work]$
```

必ずしも例外が発生するとは限らない……

ベクトル化できなければ
インテル® AVX-512 命令は
生成されない

使用する命令セットは全てのソースファイルに
同じオプションを指定することを推奨

プロセッサー・ディスパッチの影響

ランタイムチェックによるオーバーヘッドの増加

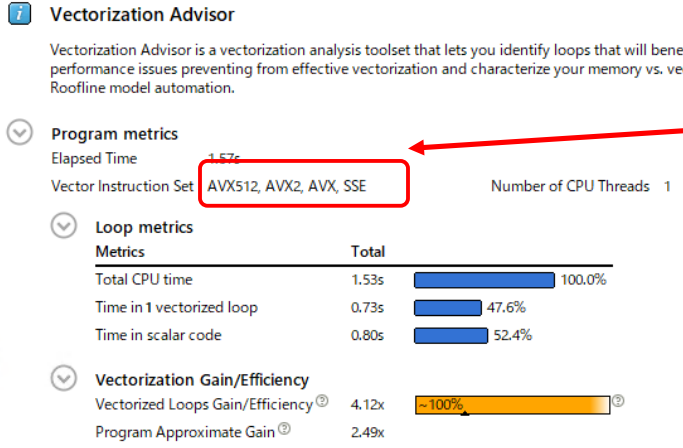
```
C:¥Users¥kiyo¥Desktop¥hakata¥code¥matmul>icl multiply.c /c /QaxCORE-AVX512 /QxCO
RE-AVX2 /Qopenmp /nologo
multiply.c

C:¥Users¥kiyo¥Desktop¥hakata¥code¥matmul>icl driver.c multiply.obj /QaxCORE-AVX2
 /nologo
driver.c

C:¥Users¥kiyo¥Desktop¥hakata¥code¥matmul>driver

ROW:101 COL: 101
Execution time is 3.269 seconds
GigaFlops = 6.241052
Sum of result = 195853.999899
```

> icl multiply.c /c /QaxCORE-AVX512 /QxCORE-AVX2
> icl driver.c multiply.obj /QaxCORE-AVX2 /Fedriver.exe

```
C:¥Users¥kiyo¥Desktop¥hakata¥code¥matmul>icl multiply.c /c /QxCORE-AVX2 /Qopenmp
 /nologo
multiply.c

C:¥Users¥kiyo¥Desktop¥hakata¥code¥matmul>icl driver.c multiply.obj /QaxCORE-AVX2
 /nologo
driver.c

C:¥Users¥kiyo¥Desktop¥hakata¥code¥matmul>driver

ROW:101 COL: 101
Execution time is 2.872 seconds
GigaFlops = 7.103760
Sum of result = 195853.999899
```

> icl multiply.c /c /QxCORE-AVX2
> icl driver.c multiply.obj /QaxCORE-AVX2 /Fedriver.exe

使用されている SIMD 命令セットを確認する



ベクトル命令セットに
インテル® AVX が適用されている

サーベイレポートで
詳細を確認

ROOFLINE

Function Call Sites and Loops		Performance Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops				Instruction Set Analysis			Advanced
							Vector ISA	Efficiency	Gain...	VL (...	Traits	Data T...	Num..	
[loop in matvec at multiply.c:15]			0.34 ...	0.34 ...	Vectorized (Bo...		AVX512	~75%	6.03x	8	FMA	Float32; F	5; 8	Unrolled by 4
[loop in matvec at multiply.c:15]			0.317s	0.317s	Vectorized (Bo...		AVX512			8	FMA	Float32; F	8	Unrolled by 4
[loop in matvec at multiply.c:15]			0.030s	0.030s	Vectorized (Re...		AVX512			8	FMA	Float64; I	5	
[loop in matvec at multiply.c:13]			0.863s	1.210s	Scalar	inner loop was alre...					Permutes	Float32...	12	
[loop in main at driver.c:100]			0.000s	1.210s	Scalar	loop with function ...						Float64	2	
[loop in matvec at multiply.c:13]			0.000s	1.210s	Scalar	inner loop was alre...					FMA	Float64...	12	

コンパイラーの最適化レポート

ベクトル化の成否について、コンパイラーから情報を得る

- コンパイラーオプション

  /Qopt-report<N>（N = 0 ~ 5）

  – テキストファイル *.optrpt へ内容を出力

※ レポートにおいてベクトル化されないとされる部分が、アプリケーションの
ホットスポット(多くの CPU 時間を利用する点)であるかは、インテル® Advisor で確認する

参考：iSUS - 新しい最適化レポートを使用してインテル® コンパイラーをさらに活用する
http://www.isus.jp/article/performance-special/get-most-out-of-intel-compiler-with-new-opt-reports/

最適化レポート

```
レポート: ループの入れ子、ベクトル、自動並列化の最適化 [loop, vec, par]
LOOP BEGIN at C:¥code¥matmul¥multiply.c(13,2)
  リマーク #15541: 外部 loop は自動ベクトル化されませんでした: SIMD ディレクティブの使用を検討してください。

  LOOP BEGIN at C:¥code¥matmul¥multiply.c(15,3)
    リマーク #15344: loop はベクトル化されませんでした: ベクトル依存関係がベクトル化を妨げています。
    最初の依存関係を以下に示します。詳細については、レベル 5 のレポートを使用してください。
    リマーク #15346: ベクトル依存関係: FLOW の依存関係が b[i] (16:4) と b[i] (16:4) の間に仮定されました。
    リマーク #25439: 剰余ありアンロール - 2
  LOOP END

  LOOP BEGIN at C:¥code¥matmul¥multiply.c(15,3)
  <Remainder>
  LOOP END
LOOP END
```

最適化レポートのオプション

/Qopt-report-phase:<フェーズ[,フェーズ,...]>
フェーズ = loop, par, vec, openmp, ipo, pgo, cg, offload, tcollect, all

/Qopt-report-file:<stdout | stderr | ファイル名>

/Qopt-report-annotate:<fmt>
fmt = html, text

/Qopt-report-routine:<関数名[,関数名]>

/Qopt-report-filter: “ソースファイル名, 行番号-行番号”

レポートレベルの指定 (ベクトル化)

/Qopt-report:*N*

N には、取得するレポートの詳細レベルを指定
N が省略された場合は *N* = 2 がデフォルト

- レベル 0: ベクトル化レポートなし
- レベル 1: ベクトル化が行われた場合をレポート
- レベル 2: レベル 1 に加え、ベクトル化されなかった場所と簡単な診断をレポート
- レベル 3: ループベクトル化の診断を追加
- レベル 4: データのアライメントなど詳細情報を追加
- レベル 5: 依存性情報を追加

ベクトル化を支援するコンパイラー機能

ベクトル化に影響する 6 つの要因

ループ伝搬依存

```
DO I = 1, N
   A(I+1) = A(I) + B(I)
ENDDO
```

関数呼び出し

```
for (i = 1; i < nx; i++) {
  x = x0 + i * h;

     sumx = sumx + func(x, y, xp);
}
```

ポインター・エイリアシング

```
void scale(int *a, int *b){
   for (int i = 0; i < 1000; i++)
         b[i] = z * a[i];
}
```

不明なループカウント

```
struct _x { int d; int bound; };

void doit(int *a, struct _x *x)
{
  for(int i = 0; i < x->bound; i++)
    a[i] = 0;
}
```

間接メモリーアクセス

```
 for (i=0; i<N; i++)
      A[B[i]] = C[i]*D[i]
```

外部ループ

```
for(i = 0; i <= MAX; i++) {
  for(j = 0; j <= MAX; j++) {
    D[i][j] += 1;
  }
}
```

ベクトル化を支援するコンパイラーの機能(1)

IPO: プロシージャ間の最適化

プロシージャ間の最適化(IPO)は自動処理でコンパイラーがソースファイルを跨ぐコード解析を行い最適化が効果的かどうか判断します

```
main(){
…
for ( i=1; i<nx; i++ ) {
   x = x0 + i*h;
   sumx = sumx + func(x,y,xp,yp)
}
…
}
```

Source1.c

```
float func(float x, float y, float xp, float yp) {
   float denom;

   denom = (x-xp)*(x-xp) + (y-yp)*(y-yp);
   denom = 1.0f/sqrtf(denom);

   return denom;   }
```

Source2.c

プロファイル情報によりインライン展開や
その他の最適化の適用率を向上

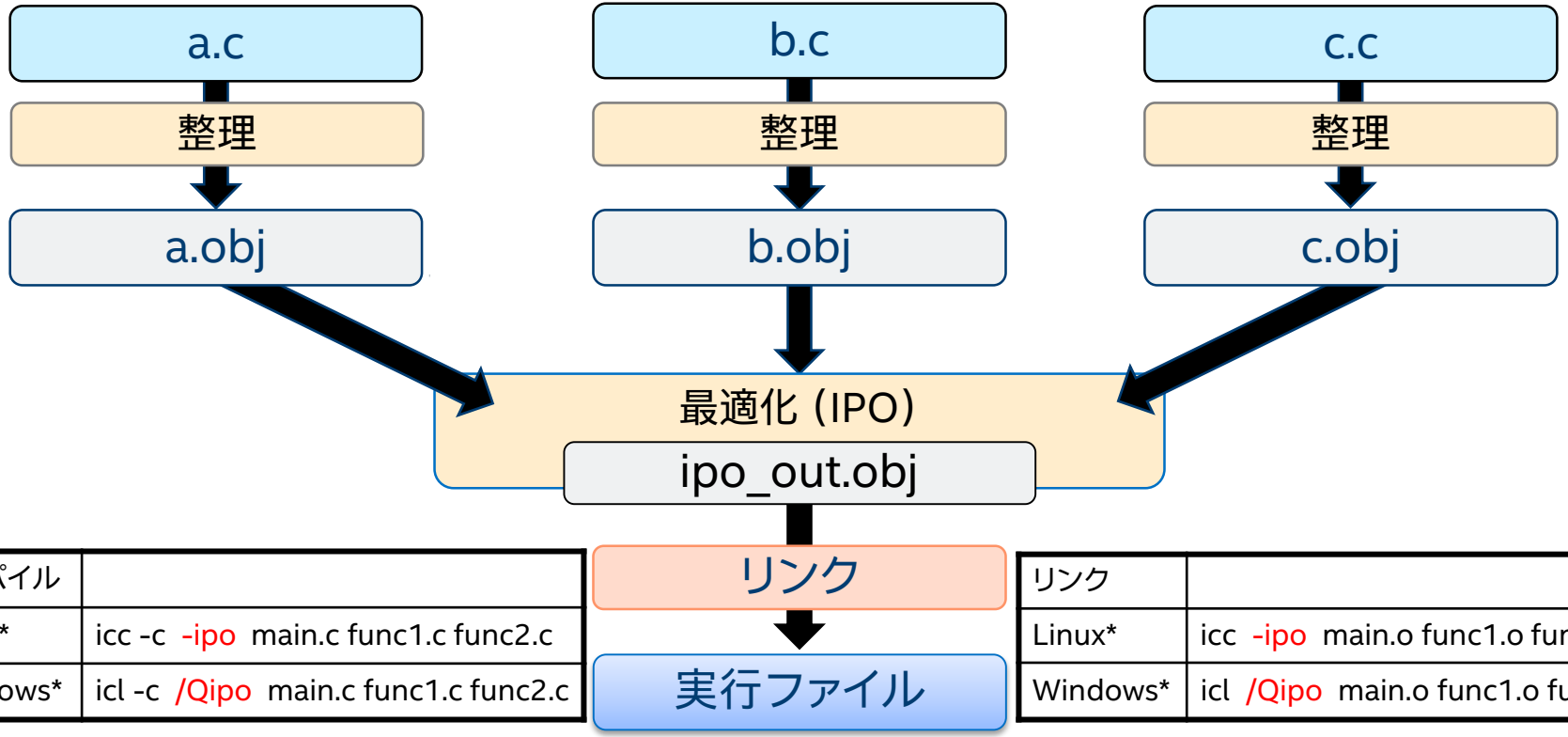
IPO による最適化

アドレスの解析
配列次元のパディング
エイリアス解析
配列の自動転置
メモリープールの自動生成
C++ クラス階層解析
共通ブロック変数の統合
共通ブロックの分割
定数の伝播
不要な呼び出しの検出
不要な仮引数の排除
不要な関数の排除
仮引数のアライメント解析
前方代入

間接呼び出し変換
インライン展開
mod/ref 解析
不要な呼び出しの部分的に排除
レジスターに引数を渡して呼び出しとレジスターの使用を最適化
ポインター解析
ルーチンのキー属性の伝播
専用化
スタックフレームのアライメント
構造体分割とフィールドの並べ替え
シンボル・テーブル・データの促進
未参照変数の削除
プログラム全体の解析

IPO のマルチステップによる最適化

擬似オブジェクトの生成



コンパイル	
Linux*	icc -c -ipo main.c func1.c func2.c
Windows*	icl -c /Qipo main.c func1.c func2.c

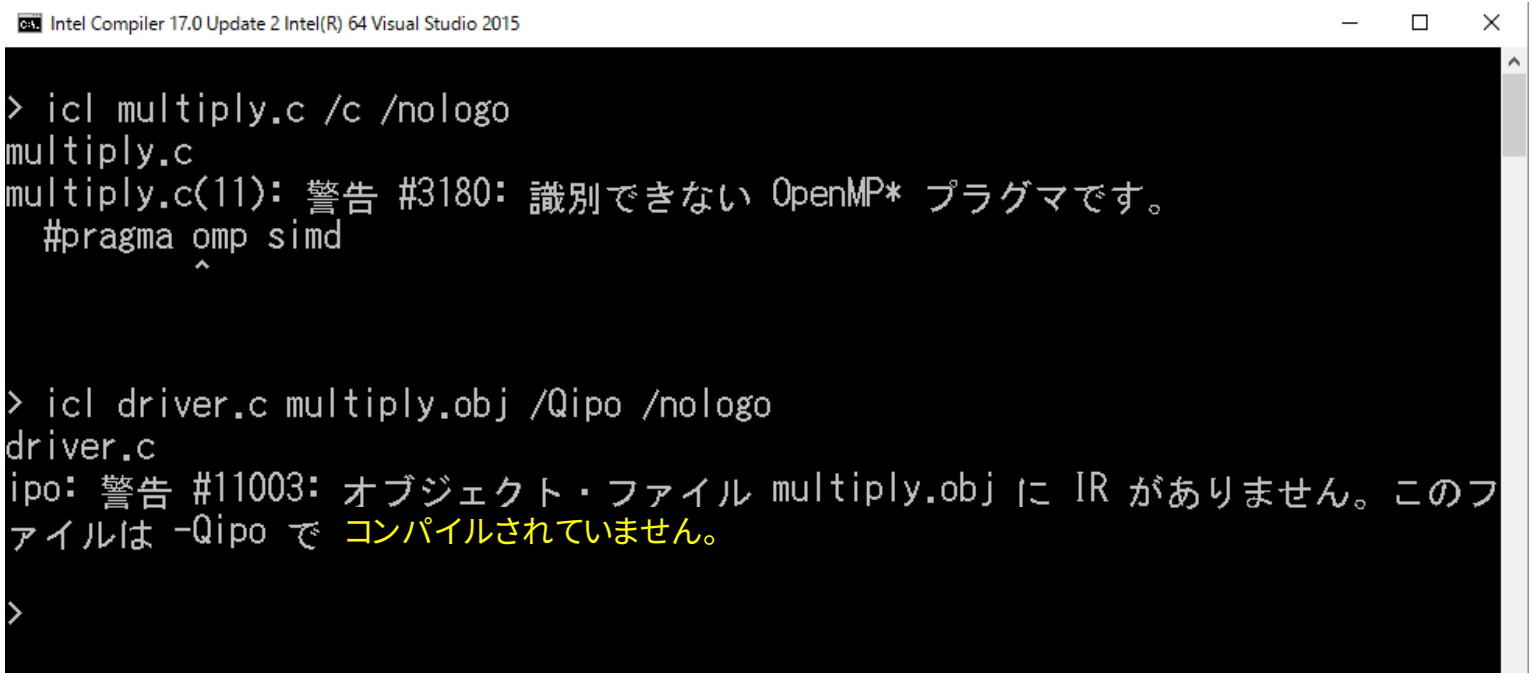
リンク	
Linux*	icc -ipo main.o func1.o func2.o
Windows*	icl /Qipo main.o func1.o func2.o

/Qipo 使用時の注意事項

1. IPO を使用して各ソースファイルがコンパイルされるたびに、コンパイラーはソースコードの中間表現 (IR) を擬似オブジェクト・ファイルに格納する。擬似オブジェクト・ファイルには、通常のオブジェクト・ファイルの代わりに IR が含まれる
2. コンパイラーはリンカーの直前に起動され、コンパイラーは、すべての擬似オブジェクト・ファイルを対象に IPO を実行する (.o や .obj にオブジェクトは含まれない)
   ipo を適用しないオブジェクトと、ipo を適用したオブジェクトをリンクする場合は、インテルのリンクツールを使用

ライブラリーを作成する場合やリンク時は、lib (ar) や link (ld) の代わりに専用ツール、xilib (xiar) と xilink (xild) を使用する

IPO を適用しないオブジェクトと IPOを適用したオブジェクト



ベクトル化を支援するコンパイラーの機能(2)

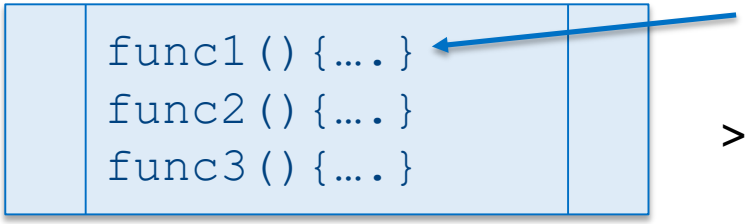
言語機能 (1)

言語機能	説明
__declspec(align(*n*))	変数を *n* バイト境界にアライメントするようにコンパイラーに指示する。変数のアドレスは *address* mod n=0
__assume_aligned(*a*,*n*)	配列 *a* が *n* バイト境界にアライメントされていると見なすようにコンパイラーに指示する。アライメント情報が取得できなかった場合に使用する
#pragma ivdep	ベクトル依存性が存在していると推定されてもそれを無視するようにコンパイラーに指示
#pragma novector	ループをベクトル化しないように指定

ベクトル化を支援するコンパイラーの機能(2)

言語機能 (2)

ソースコードに複数の関数が含まれている場合、アルゴリズムやメモリー参照の方法により、特定の SIMD 命令セットで最適な性能を発揮するような状況も発生することがある



source_code.c

インテル® AVX2 で最高の性能を発揮!!

> icl source_code.c /QxCORE-AVX512

#pragma [intel] optimization_parameter target_arch=CORE-AVX2
を関数宣言の直前に追加すると、対象の関数だけ命令セットを変更可能

ベクトル化を支援するコンパイラーの機能(3)

ポインタエイリアスの可能性を制御

- 回避方法(1): ポインタの restrict 化
  - C99 で規定(コンパイラーオプション -std=c99)
  - C++ でもコンパイラー独自拡張として利用可能(-restrict)
- 回避方法(2): コンパイラーオプション
  - -fargument-noalias
    仮引数がエイリアスしないことを仮定
  - Fortran では、言語仕様により
    デフォルトで有効
    (-assume dummy_aliases で無効化)

```
1  void addvec(int num, float * restrict a, float *b)
2  {
3      int i;
4
5      for(i=0; i<num; i++){
6          a[i] = a[i] + b[i];
7      }
8  }
```

ベクトル化を支援するコンパイラーの機能(4)

-no-vec オプション

- ベクトル化によりアプリケーションがどれくらい恩恵を得られているか簡単に調査
- /Qvec- (-no-vec) オプションが指定されるとコンパイラーは、SIMD 命令を使用するが、ベクトル化せずにスカラー操作を行うコードを生成
- /Qvec- (-no-vec) ありと、なしのバイナリーを作成してパフォーマンスを比較

ベクトル化を支援するコンパイラーの機能(5)

浮動小数点数値演算に対する最適化の制御 - /fp:<keyword>

/Od (最適化無効)

1.79975711734**40164**e+12

オプション無し (デフォルト : SSE2, /fp:fast)

1.79975711734**39973**e+12

/fp:precise

1.79975711734**40164**e+12

/QxCORE-AVX2

1.79975711734**40073**e+12

/QxCORE-AVX2 /fp:precise

1.79975711734**40227**e+12

/QxCORE-AVX2 /fp:consistent

1.79975711734**40164**e+12

```
// #include <random>

   mt19937 engine(777);
   uniform_real_distribution<double>
         dist(-1.0e10, 1.0e10);

   vector<double> x(100000);
   for( size_t i=0; i<x.size(); i++ ) {
      x[i] = dist(engine);
   }

   double sum = 0.0;
   for( size_t i=0; i<x.size(); i++ ) {
      sum = sum + x[i];
   }
   printf("%.16e¥n", sum);
```

浮動小数点数演算の不確実性

異なる条件では異なる結果が生じることがある

- コンパイラー(ベンダー)の違い
- コンパイラーバージョンの違い
- (SIMD)命令セットの違い
- 並列実行数の違い(MPI の場合は、ノード内並列数の違いも含む)
- CPU(マイクロアーキテクチャー世代)の違い

参考情報

インテル® コンパイラーの浮動小数点演算における結果の一貫性(英語)
https://software.intel.com/en-us/articles/consistency-of-floating-point-results-using-the-intel-compiler
(参考訳)http://www.isus.jp/article/compileroptimization/consistency-of-floating-point-results/

The Parallel Universe - Issue 21 より「インテル® MPI ライブラリーの条件付き再現性」
http://www.xlsoft.com/jp/products/intel/tech/guide.html?tab=3

インテル® Advisor

ベクトル化によるパフォーマンスとベクトル化状況を把握する

効果的な最適化の方針を見つける

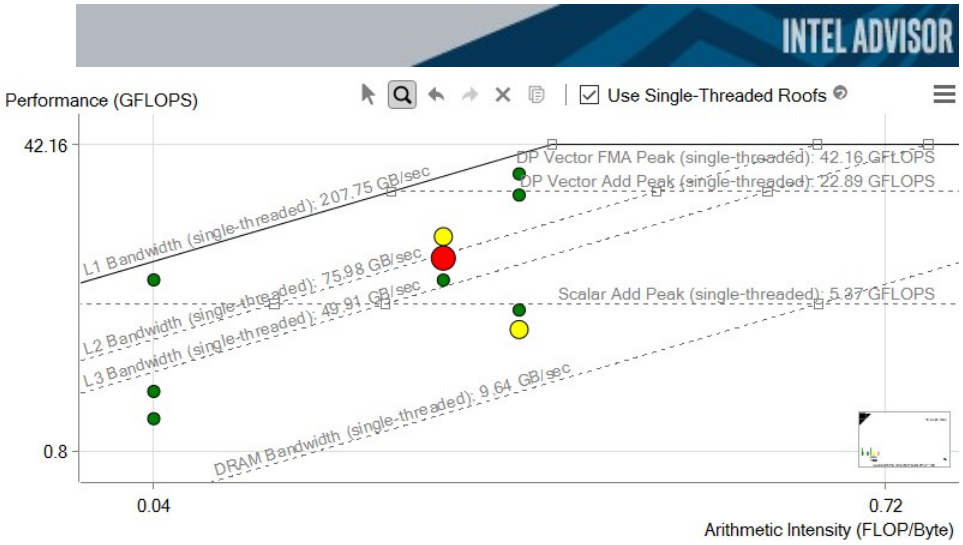
インテル® Advisor: キャッシュを意識したルーフライン解析

ルーフライン・パフォーマンスの詳細
パフォーマンスの問題があるループをハイライト

それぞれのループのパフォーマンスの「ヘッドルーム」を表示

- 改善可能なループは?
- 改善の価値があるループは?

ボトルネックの原因となる可能性を表示して最適化の次のステップを提案

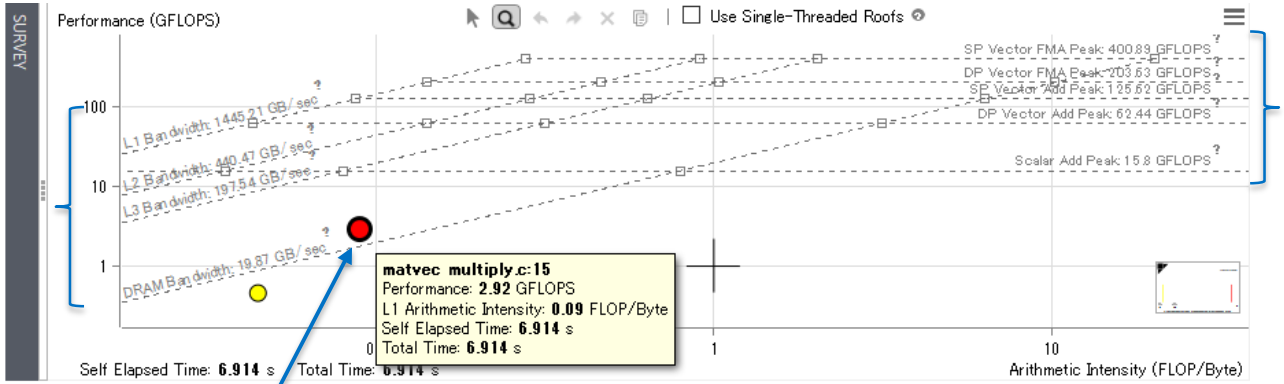


$$AI = \frac{flops}{DRAM\ からの転送サイズ\ (byte)}$$

プログラムの性能を知る方法

インテル® Advisor の Roofline (ルーフライン) 解析を使用して、デフォルトコードのパフォーマンスを理解する: icl driver.c multiply.c /O2 /Zi

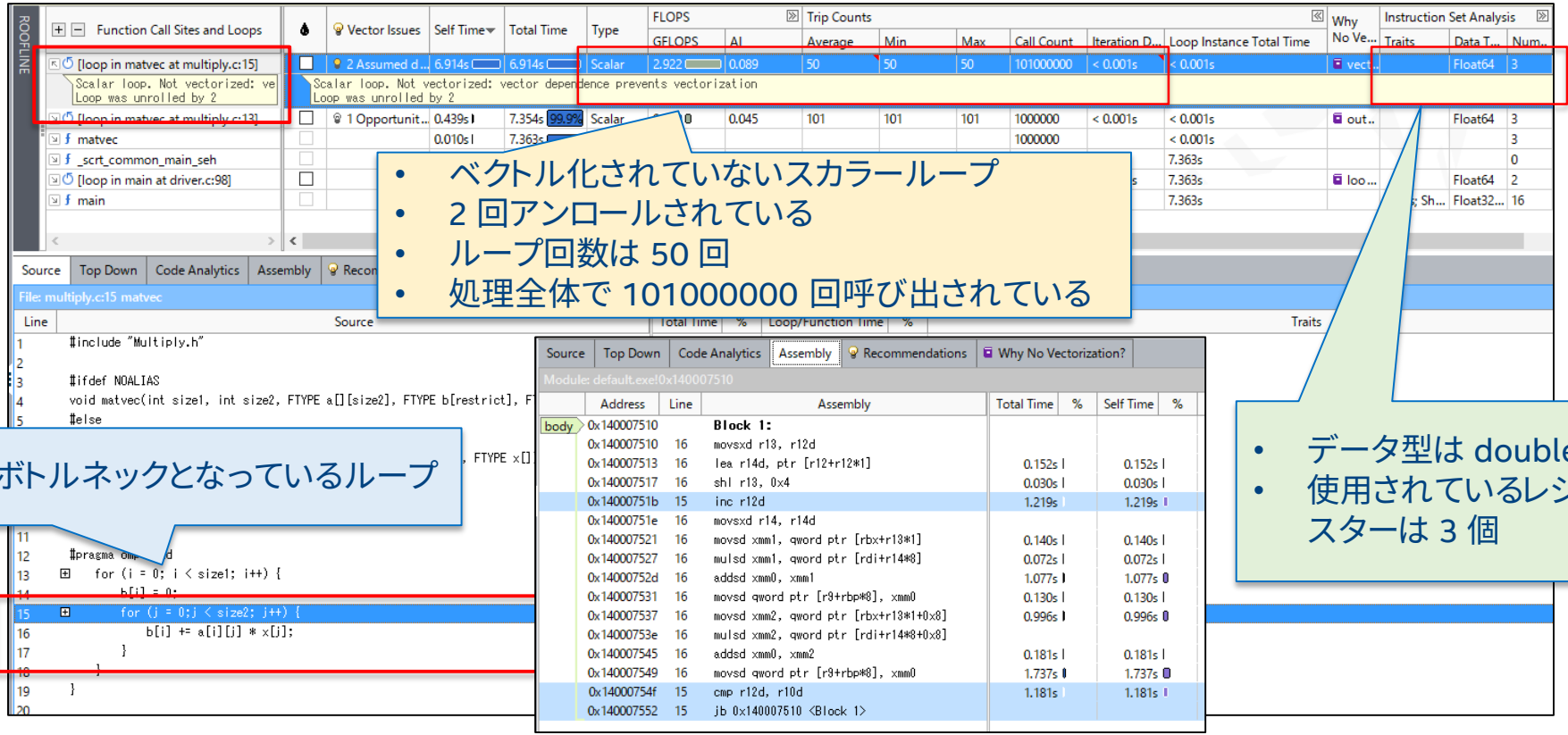
キャッシュとメモリーの潜在的なピーク性能



命令セットの潜在的なピーク性能

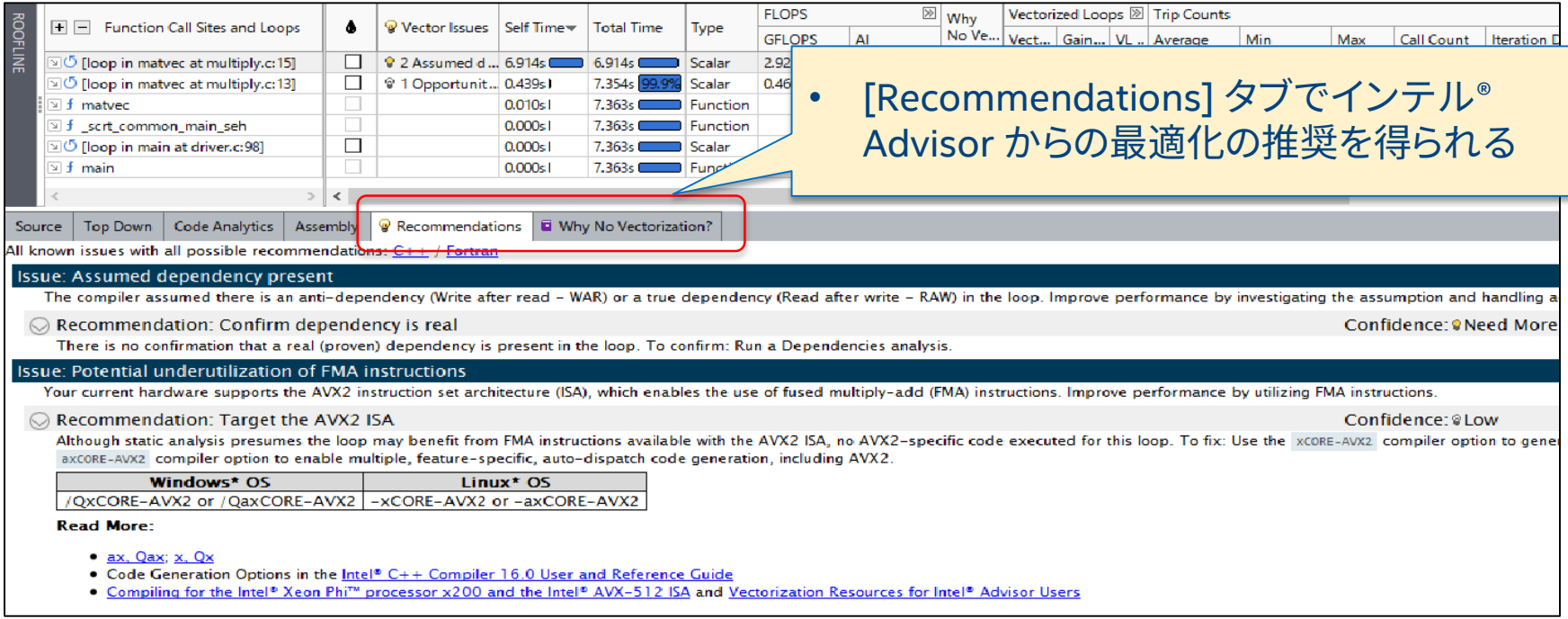
この段階でアプリケーションは、メモリー集約型であり SIMD 命令のロード/ストアデータはキャッシュに収まっていないか、活用していない

サーベイ解析で詳しく調査



32

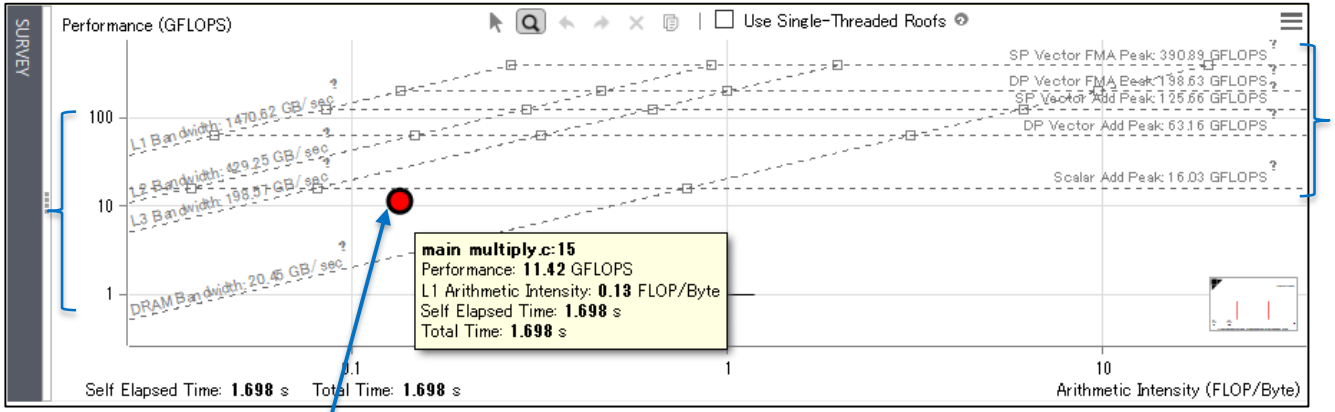
インテル® Advisor による推奨



/Qipo でデフォルトのベクトル化を適用した結果

>icl driver.c multiply.c /Qipo /Zi /O2

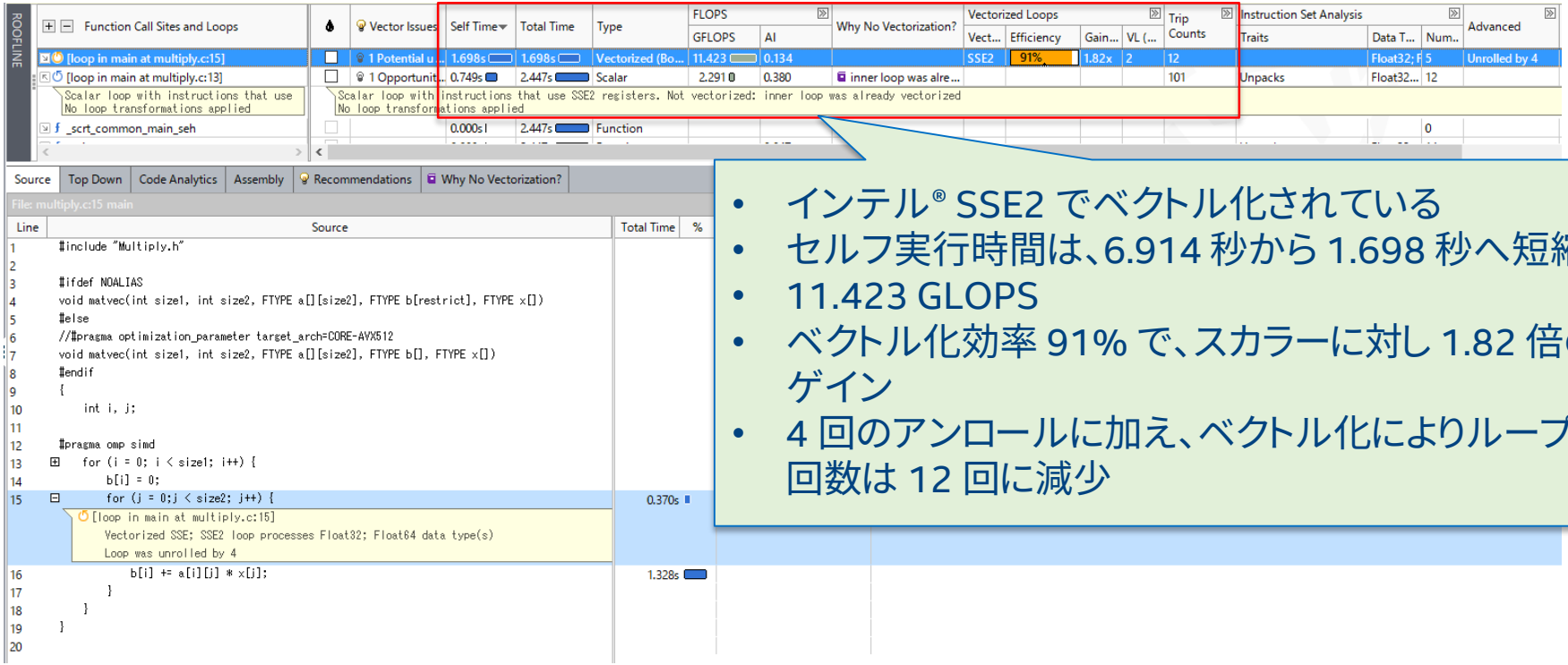
キャッシュとメモリーの潜在的なピーク性能



命令セットの潜在的なピーク性能

この段階でアプリケーションの計算性能はかなり向上、L3 キャッシュを効率良く利用しつつある

サーベイ解析で調査 : /Qipo 適用後

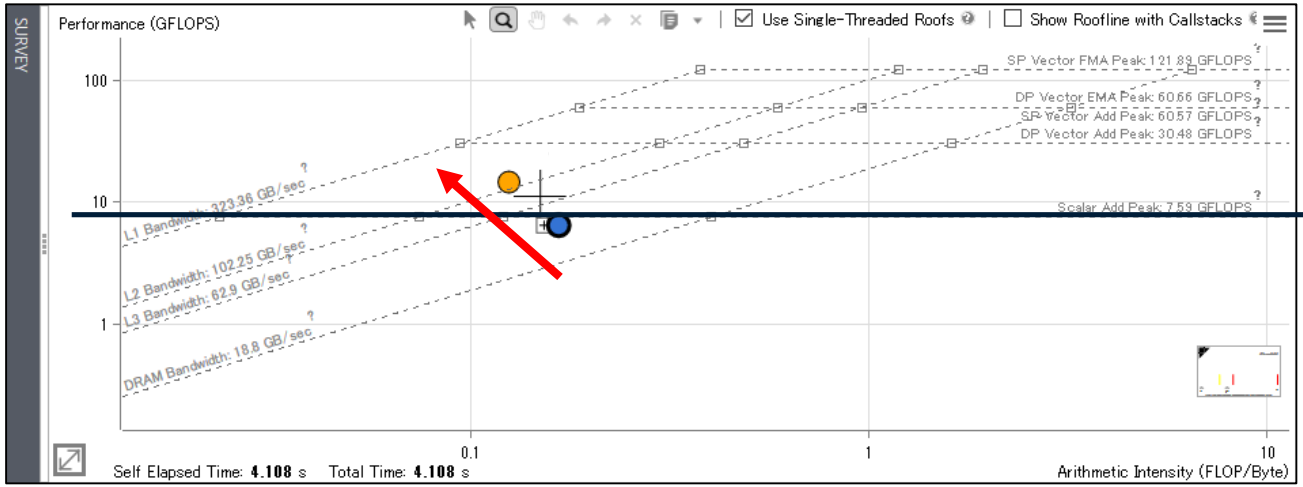


- インテル® SSE2 でベクトル化されている
- セルフ実行時間は、6.914 秒から 1.698 秒へ短縮
- 11.423 GLOPS
- ベクトル化効率 91% で、スカラーに対し 1.82 倍のゲイン
- 4 回のアンロールに加え、ベクトル化によりループ回数は 12 回に減少

最適化による特性変化の可能性を考慮する

ボトルネックの特定

デフォルトのコードをルーフライン解析を使用したポイントと、
同じ処理をベクトル化したポイントを表示 : icl main.c /Zi /O2 /QxCORE-AVX2

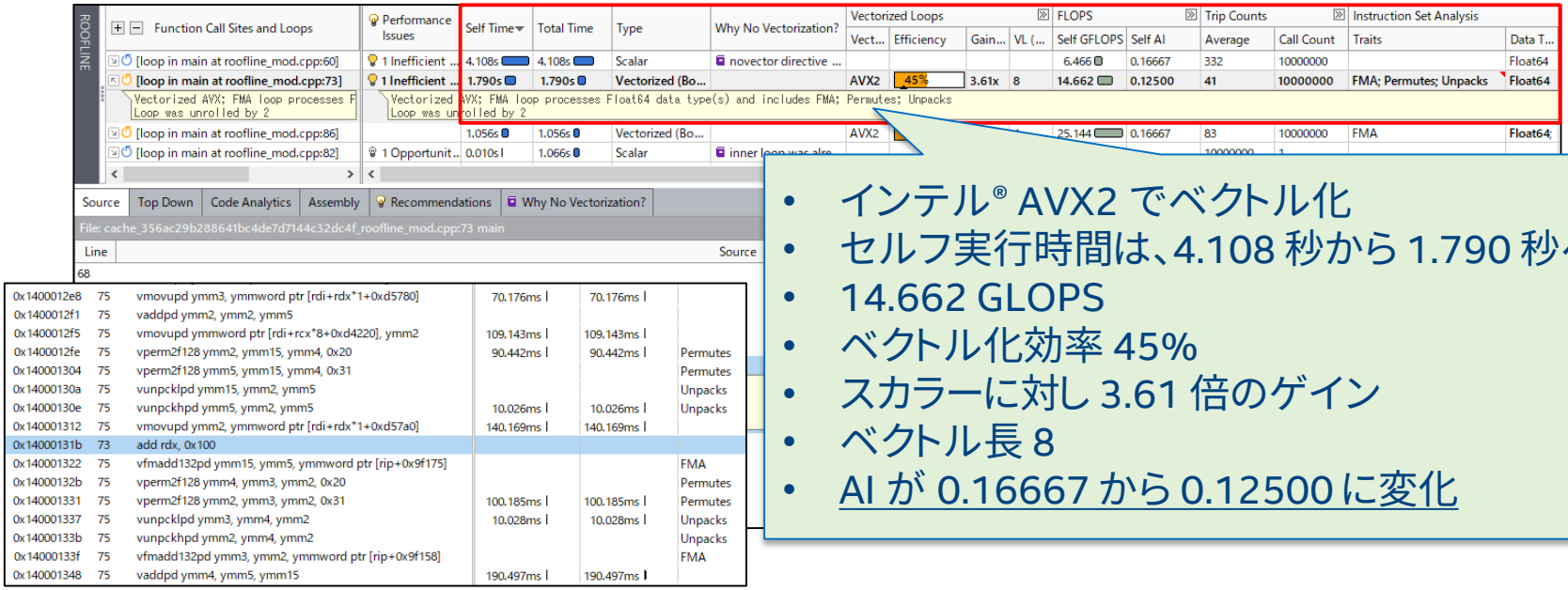


スカラー演算のルーフ

ベクトル化よりスカラー演算のルーフを超えたが、同時に計算強度が低下している

サーベイ解析の調査：ベクトル化の適用

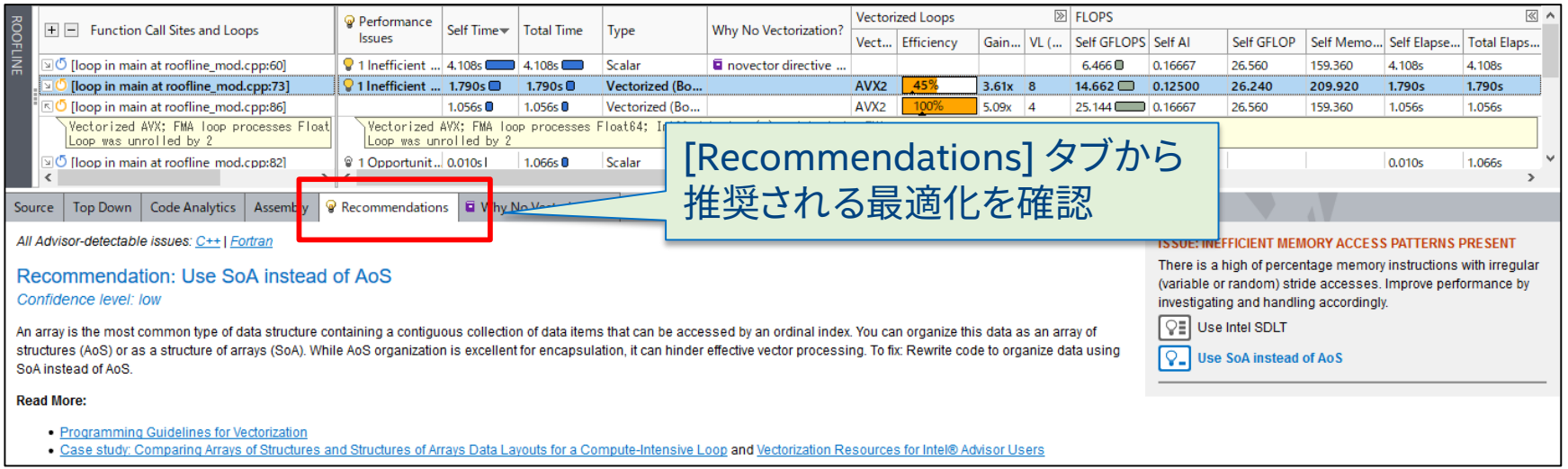
ユニットストライドの調査



- インテル® AVX2 でベクトル化
- セルフ実行時間は、4.108 秒から 1.790 秒へ短縮
- 14.662 GLOPS
- ベクトル化効率 45%
- スカラーに対し 3.61 倍のゲイン
- ベクトル長 8
- AI が 0.16667 から 0.12500 に変化

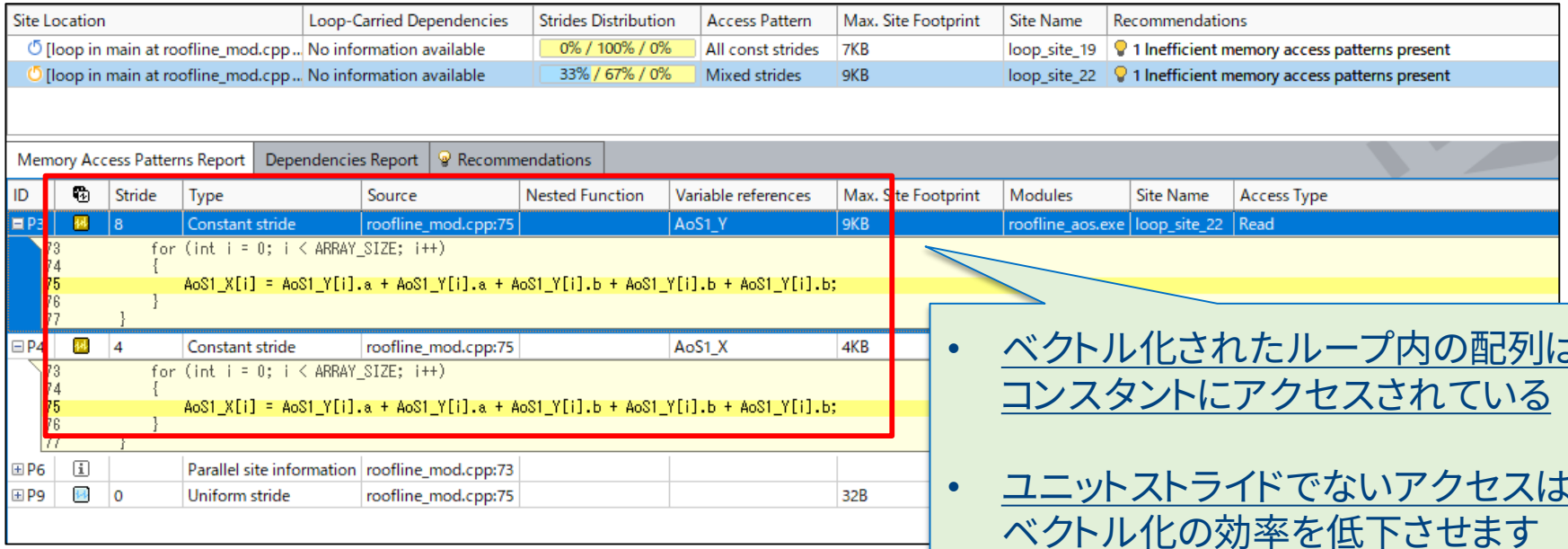
インテル® Advisor の推奨を確認

AoS を SoA にレイアウト変換する最適化を推奨



メモリーアクセスパターン解析の実行

ユニットストライドの調査



Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Max. Site Footprint	Site Name	Recommendations
[loop in main at roofline_mod.cpp..	No information available	0% / 100% / 0%	All const strides	7KB	loop_site_19	1 Inefficient memory access patterns present
[loop in main at roofline_mod.cpp..	No information available	33% / 67% / 0%	Mixed strides	9KB	loop_site_22	1 Inefficient memory access patterns present

Memory Access Patterns Report | Dependencies Report | Recommendations

ID	Stride	Type	Source	Nested Function	Variable references	Max. Site Footprint	Modules	Site Name	Access Type
P3	8	Constant stride	roofline_mod.cpp:75		AoS1_Y	9KB	roofline_aos.exe	loop_site_22	Read

```
73          for (int i = 0; i < ARRAY_SIZE; i++)
74          {
75              AoS1_X[i] = AoS1_Y[i].a + AoS1_Y[i].a + AoS1_Y[i].b + AoS1_Y[i].b + AoS1_Y[i].b;
76          }
77      }
```

P4	4	Constant stride	roofline_mod.cpp:75		AoS1_X	4KB

```
73          for (int i = 0; i < ARRAY_SIZE; i++)
74          {
75              AoS1_X[i] = AoS1_Y[i].a + AoS1_Y[i].a + AoS1_Y[i].b + AoS1_Y[i].b + AoS1_Y[i].b;
76          }
77      }
```

P6		Parallel site information	roofline_mod.cpp:73			
P9	0	Uniform stride	roofline_mod.cpp:75			32B

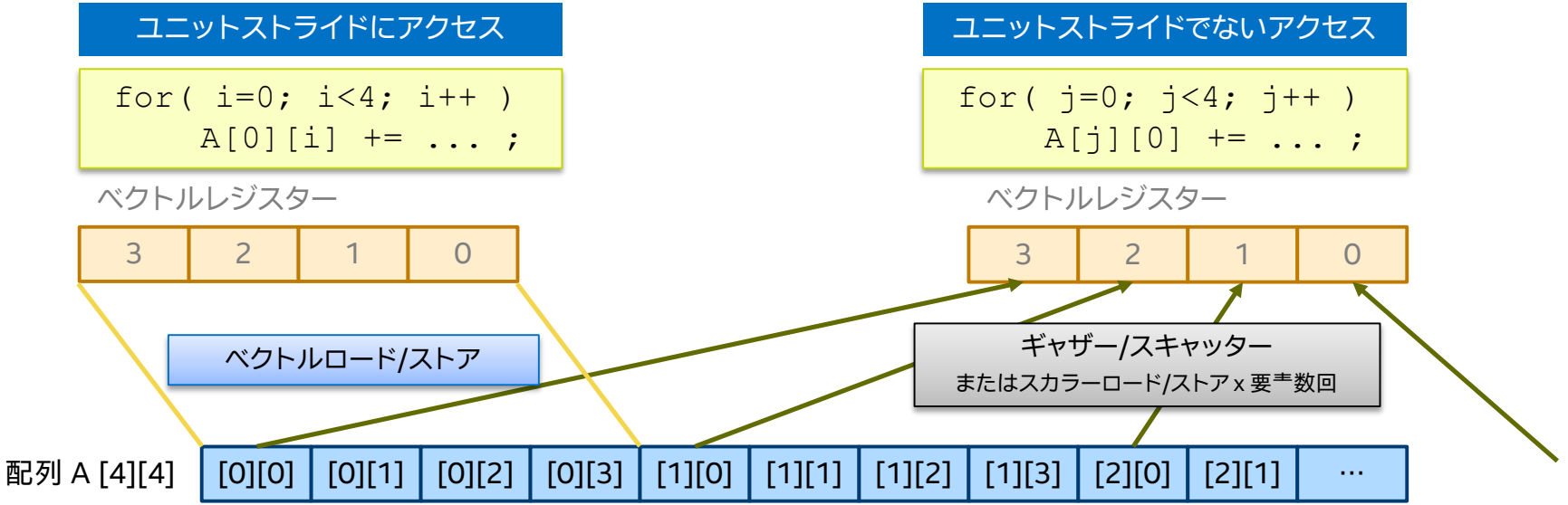
- ベクトル化されたループ内の配列はコンスタントにアクセスされている
- ユニットストライドでないアクセスはベクトル化の効率を低下させます

データ配置、アクセスパターンによる効率の変化

SIMD 命令は、連 売したデータを扱うため、隣り合わない要素のアクセスは非効率

- ユニットストライドでないデータアクセスは
  性能低下の原因になる

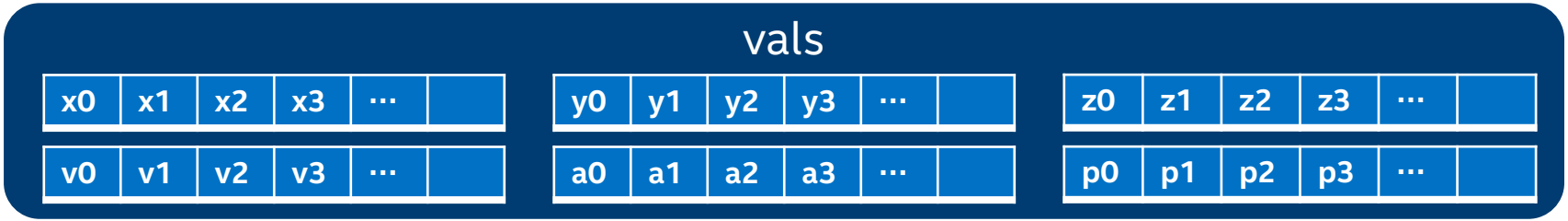
※ Fortran は配列の左の添え字、
C/C++ は配列の右の添え字がユニットストライド



AoS(構造体の配列)と SoA(配列の構造体)

構造体の配列(AoS)

配列の構造体(SoA)

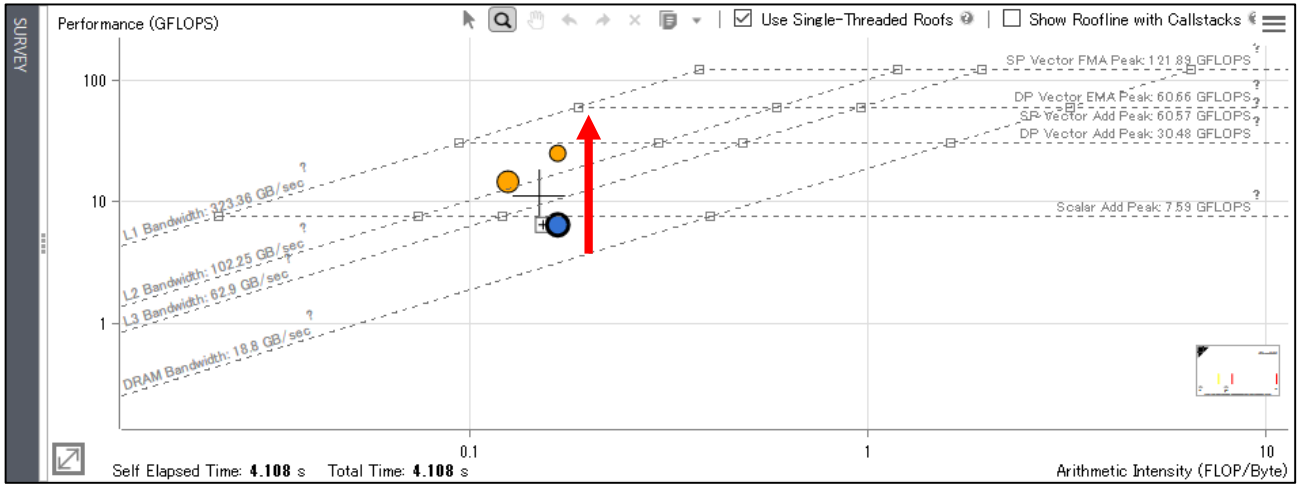


構造体の各要素に連続してアクセスすることが多い場合、SoA の方が効率的

AoS から SoA にレイアウト変換を行う最適化

```
// Array of Structures
typedef struct S1_AoS
{
    double a;
    double b;
} S1_AoS;
double AoS1_X[ARRAY_SIZE];
S1_AoS AoS1_Y[ARRAY_SIZE];
```

```
// Structure of Arrays
typedef struct S1_SoA
{
    double a[ARRAY_SIZE];
    double b[ARRAY_SIZE];
} S1_SoA;
double SoA1_X[ARRAY_SIZE];
S1_SoA SoA1_Y;
```



AoS1_Y 構造体を SoA に変更したことにより、SIMD 命令のロード/ストアが最適化され
パフォーマンスが上に向上している

まとめ

インテル®コンパイラーの自動ベクトル化機能と SIMD 命令セットを利用します

ベクトル化に影響する要因を意識します

インテル® Advisor で現状の把握とコード解析を行い改善の糸口を発見します

効率のよくベクトル化されたコードは、マルチスレッド化による
相乗効果を高め、さらにハードウェアの性能を
引き出すことが可能です