

# インテル® ソフトウェア開発製品 によるソースコードの近代化

エクセルソフト株式会社  
黒澤 一平

# ソースコードの近代化

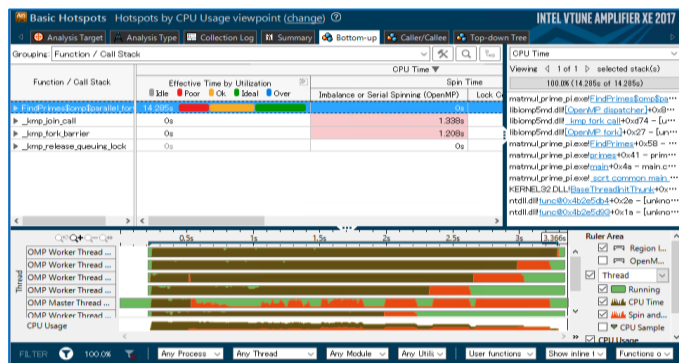


- インテル® Xeon Phi™ プロセッサや、将来のインテル® Xeon® プロセッサ上での実行に向けた準備と適用

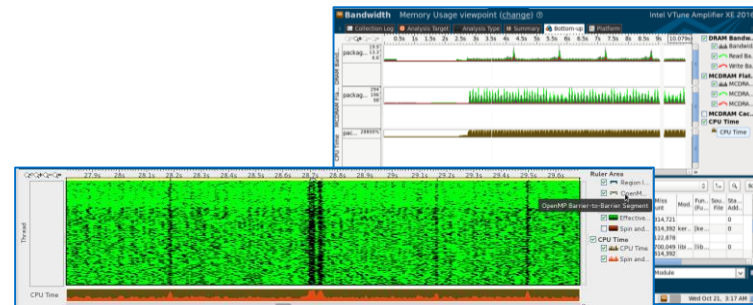
UNLEASH YOUR CODE'S POTENTIAL  
BRING YOUR FUTURE TO LIFE

# インテル® ソフトウェア製品

名称	概要
インテル® Composer XE for Fortran and C++	Fortran, C/C++ 言語に対応した最適化コンパイラー
インテル® VTune™ Amplifier XE	CPU 内部の、処理の効率性やさまざまな情報を取得
インテル® Advisor	ベクトル化、マルチスレッド化の解析を行い、最適化アドバイスを提供
インテル® Trace Analyzer & Collector	MPI アプリケーションの動作状況やボトルネックを解析
インテル® Performance Snapshot	アプリケーション全体の性能をシンプルに解析



インテル® VTune™ Amplifier XE



インテル® Xeon Phi™ プロセッサ  
(開発コード名: Knights Landing) 対応

# 最新ツールを使用する利点

- より高度な最適化を行えるようになります。
- 初心者でも上級者に近い最適化を行えるようになります。
- 時間を大幅に短縮することができます。
- ツールからアドバイスを得ることができます。
- 数年先まで有効なコードを作ることができるようになります。



# ソフトウェア開発者が考慮すべき並列性

## ベクトル化

コアの命令セットを利用

- 1 コアごとの性能向上
  - 複数のデータ要素を同時に処理 (SIMD)
- 

## スレッド並列化

複数コアを利用

- 1 プロセッサの性能向上
  - 複数タスクの同時実行
- 

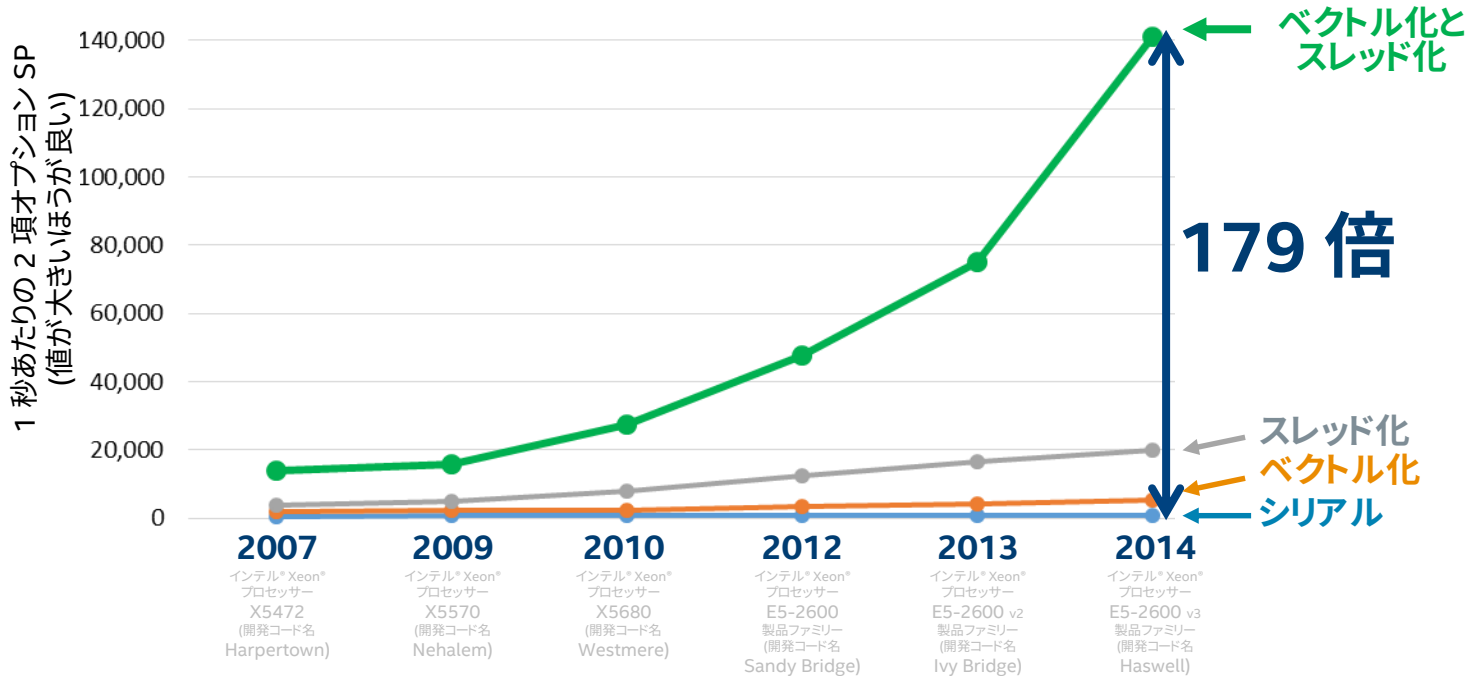
## MPI 並列化

複数マシンを利用

- 複数のマシンを使用
- 複数プロセスの同時実行

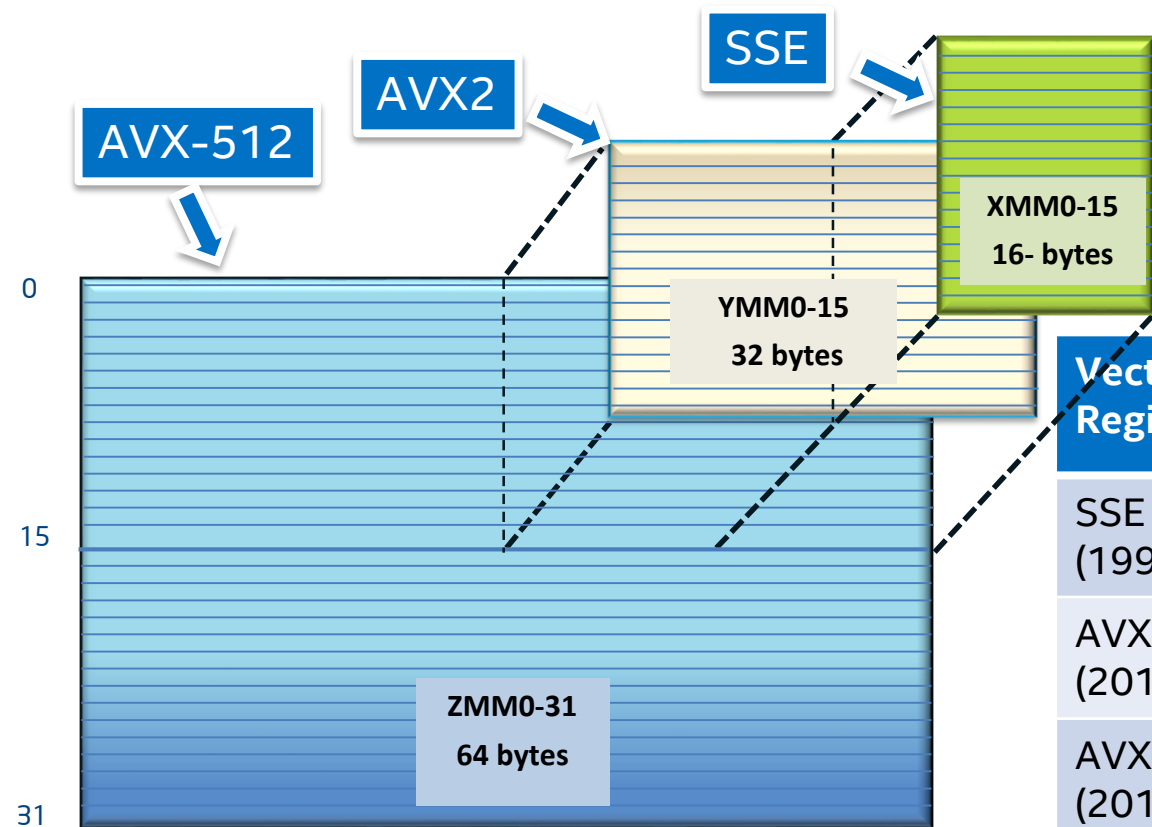
# ベクトル化とマルチスレッド化、最適化の効果

## マルチスレッド化 + ベクトル化はより良い効果が得られる



性能に関するテストに使用されるソフトウェアとワークロードは、性能がIntel® マイクロプロセッサ用に最適化されていることがあります。SYSmark® や MobileMark® などの性能テストは、特定のコンピューターシステム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。詳細については、<http://www.intel.com/performance/> (英語) を参照してください。

# AVX-512



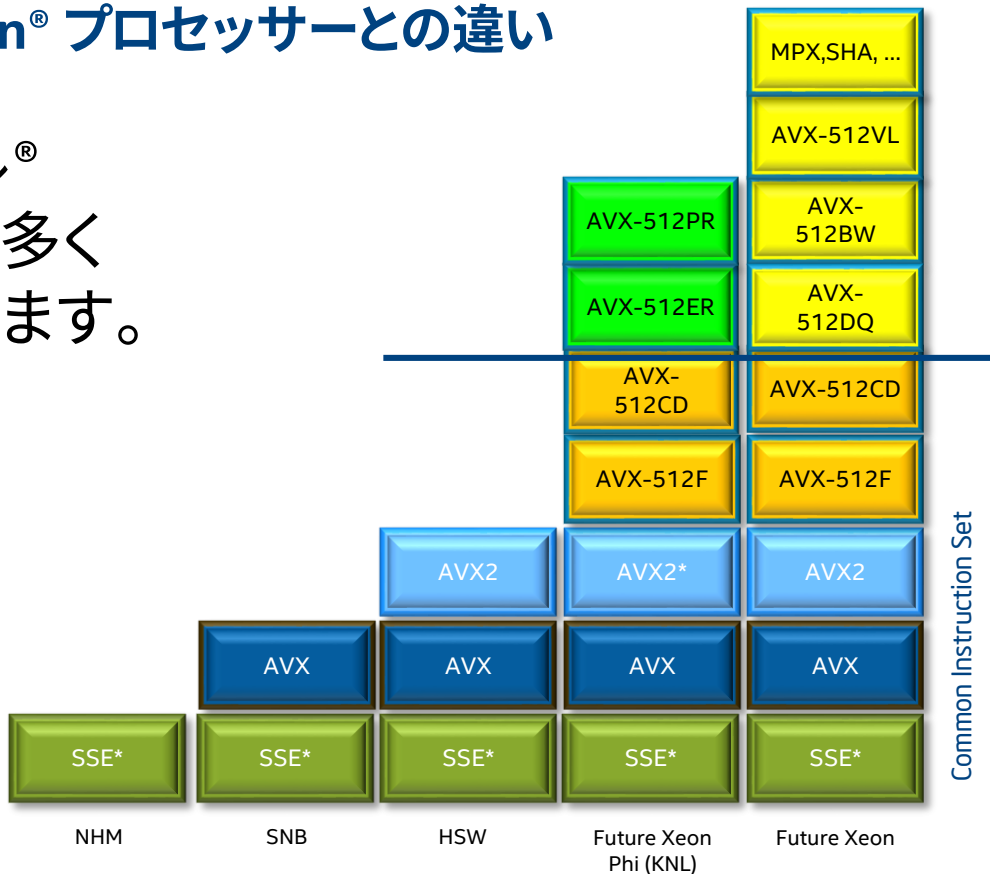
Vector Registers	IA32 (32bit)	Intel64 (64bit)
SSE (1999)	8 x 128bit	16 x 128bit
AVX and AVX-2 (2011 / 2013)	8 x 256bit	16 x 256bit
AVX-512 (2014 – KNL)	8 x 512bit	32 x 512bit

# AVX-512

## KNLと将来のインテル® Xeon® プロセッサーとの違い

- KNLと将来のインテル® Xeon® プロセッサーは多くの互換命令を有しています。

コンパイラオプション	ターゲット
<b>-xmic-avx512</b>	KNLのみ
<b>-xcore-avx512</b>	将来の XEONのみ
<b>-xcommon-avx512</b>	KNLおよび将来の XEON
<b>-mmic</b>	KNC用





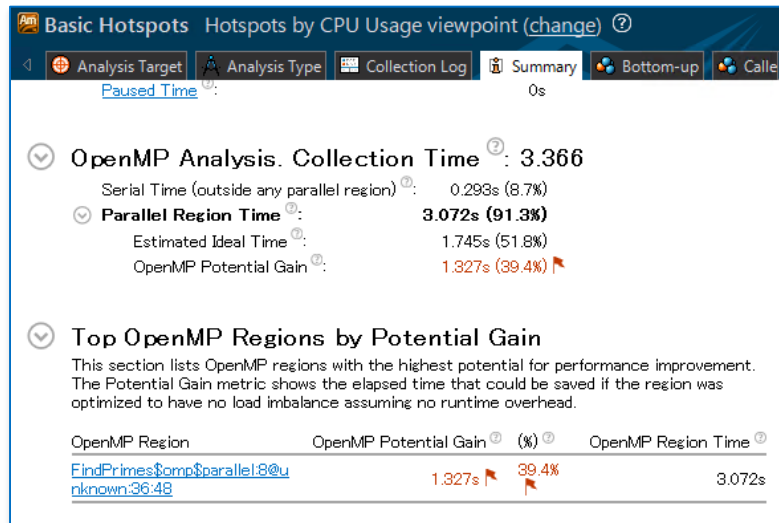
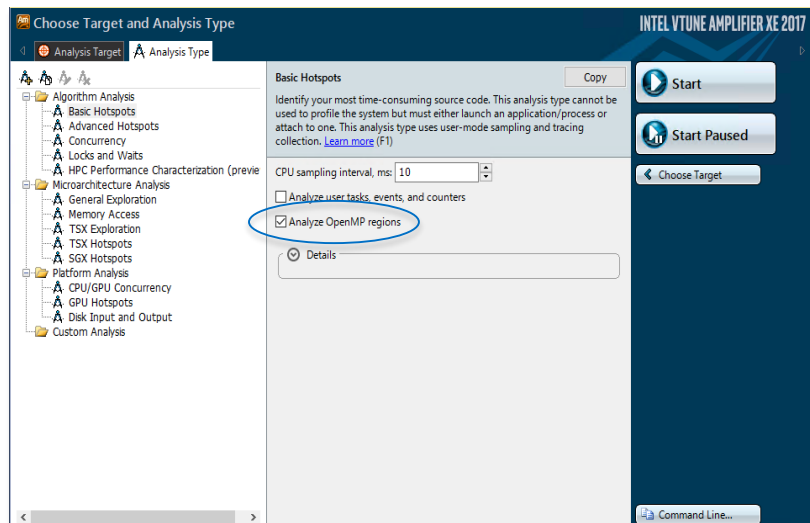
# KNL に向けた準備

作業	対応ツール	実施内容
コンパイル/リンク	インテル® コンパイラー	インテル® Xeon® プロセッサの場合と同じようにコンパイル、実装、解析することができます。  ただし、512ビットのベクトル化と、高並列性を目指す必要があります。
ベクトル化	インテル® コンパイラー インテル® Advisor	
マルチスレッド化	インテル® コンパイラー インテル® Advisor	
ベクトル/マルチスレッド性能解析	インテル® Advisor インテル® VTune Amplifier XE	
MPI 性能解析	インテル® Trace Analyzer & Collector	

インテル® ソフトウェア開発製品は初心者でも上級者に近い最適化を行えるような様々な補助機能を提供します。

# OpenMP\* アドバイス機能

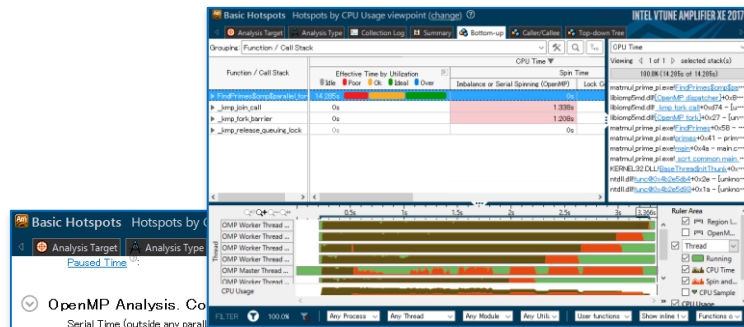
- インテル® VTune™ Amplifier XE の OpenMP\* 解析機能を使用することで、OpenMP\* を用いたマルチスレッド化のパフォーマンス問題と、改善点を確認することができ、修正した場合のパフォーマンスの向上度合いが表示されます。



# OpenMP\* のパフォーマンス問題

- 仕事の不均一性、ロックなどによるスピン時間と、OpenMP\* スレッドの生成、スケジューリング、リダクション、アトミック演算、などのオーバーヘッド時間の問題を特定することができます。

Function / Call Stack	Effective Time by Utilization				Spin Time
	Idle	Poor	Ok	Ideal	
FindPrimes\$omp\$parallel_for	14.285s				0s
_kmp_join_call	0s				1.339s
_kmp_fork_barrier	0s				1.200s



Top OpenMP Regions by Potential Gain

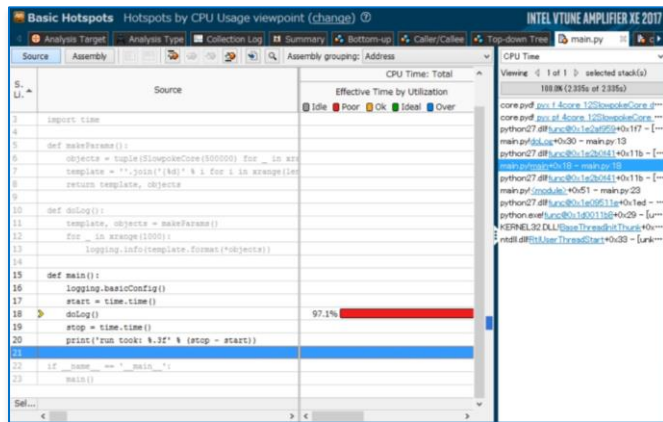
This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	OpenMP Potential Gain (%)	OpenMP Region Time
FindPrimes\$omp\$parallel.8@u nknown.36.48	39.4%	3.072s

# インテル® Distribution for Python\* または Go と性能解析ツール

- パフォーマンスが問題なく発揮されているか、インテル® VTune™ Amplifier XE で素早く確認することができます。

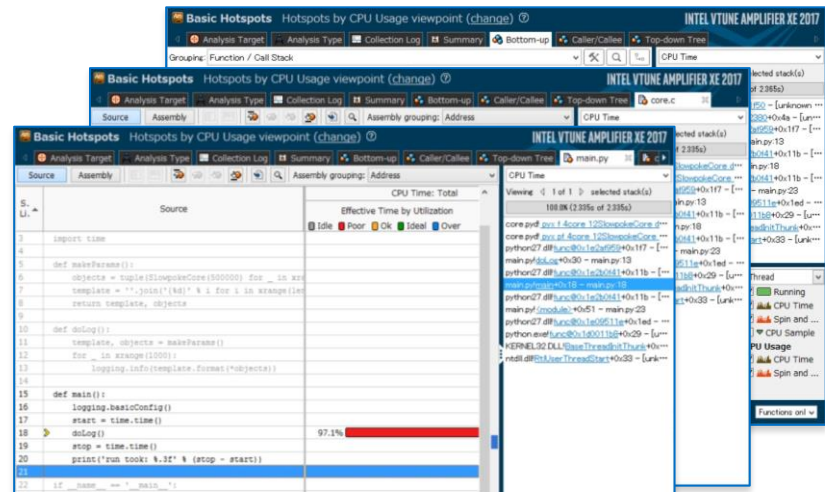
処理が Python\* または Go コード側で行われているのか、C ライブラリーやインテル® MKL に置換されているのかをすぐ確認することができ、Hotspots を即座に発見することができます。



- インテル® Distribution for Python\* は通常の Python\* コードと使用して内部でインテル® MKL などを呼び出して処理するため、より短時間で演算を行うことができます。

# インテル® VTune™ Amplifier XE による Python\* コードの性能解析

- Python\* または Go アプリケーションのパフォーマンス問題を確認することができます。  
また、Cython やインテル® MKL のネイティブ・ライブラリーと、Python\* または Go の混在も同様に解析することができます。



```
15 def main():
16     logging.basicConfig()
17     start = time.time()
18     doLog()
19     stop = time.time()
20     print('run took: %.3f' % (stop - start))
```

97.1%

# インテル® DAAL

DAAL : (Data Analytics Acceleration Library)

- インテル・プロセッサ上で動作する  
ビッグデータ解析処理を最適化するためのライブラリー

大量のデータを取り扱う  
問題分析や意思決定の高速化

主な問題をカバー

知識発見  
データマイニング  
マシンラーニング  
予測分析  
AI  
パターン認識、  
ニューロ・コンピューティング



# ビッグデータ (数百テラ～ペタ規模のデータ)

- 一般的に市販されている管理ツールや、データベースでは解析、保管が困難なデータ集合体

## ビッグデータの特徴

- データサイズが膨大
- データの種類が多い
- データが頻繁に更新される

## データ例:

単純なテキスト、画像、動画、音楽、  
センサーから送られてくるデータ、  
利用者の趣味趣向、金融

# インテル® DAAL

- データ分析で行われる全てのステージをカバー可能

データソース

ビジネス  
科学  
工学  
Web/SNS

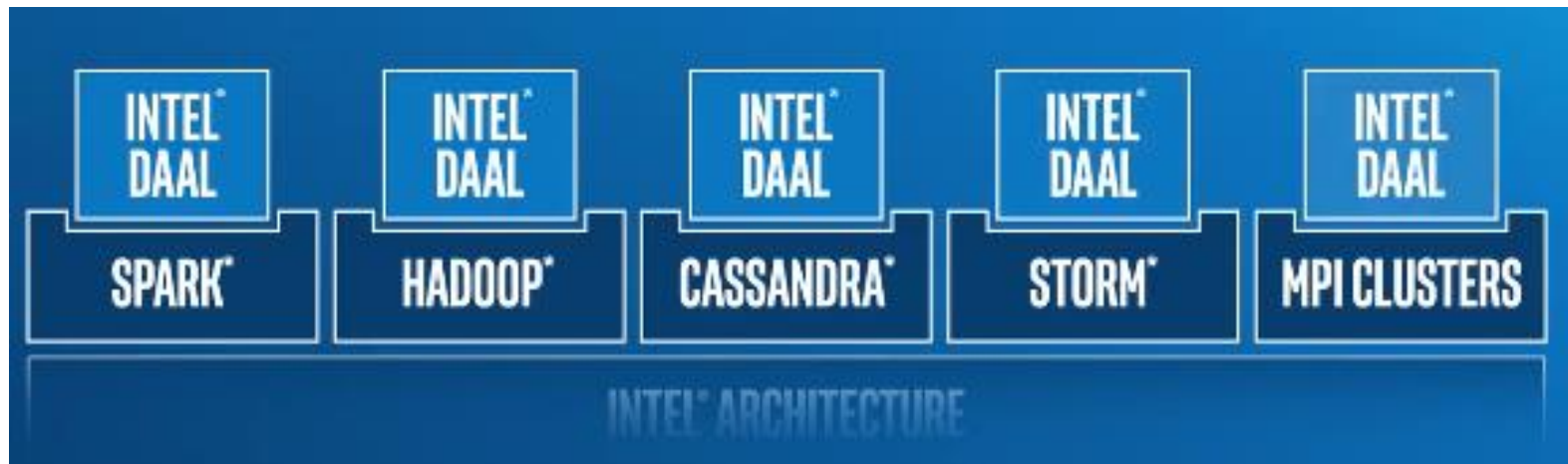
前処理	変換	解析	モデリング	検証	意志決定
<ul style="list-style-type: none"><li>展開</li><li>フィルター</li><li>正規化</li></ul>	<ul style="list-style-type: none"><li>集計</li><li>次元縮退</li></ul>	<ul style="list-style-type: none"><li>サマリー統計</li><li>クラスタリング</li></ul>	<ul style="list-style-type: none"><li>機械学習</li><li>パラメーター推定</li><li>シミュレーション</li></ul>	<ul style="list-style-type: none"><li>仮説検証</li><li>モデルエラー</li></ul>	<ul style="list-style-type: none"><li>予測</li><li>決定木</li><li>その他</li></ul>

それぞれのステージに対して最適化されたアルゴリズムを提供

# インテル® DAAL の特徴

- メジャーな解析プラットフォームの解析処理からインテル® DAAL のアルゴリズムを接続して使用

解析処理の開発にかかる時間を短縮



# インテル® DAAL の性能

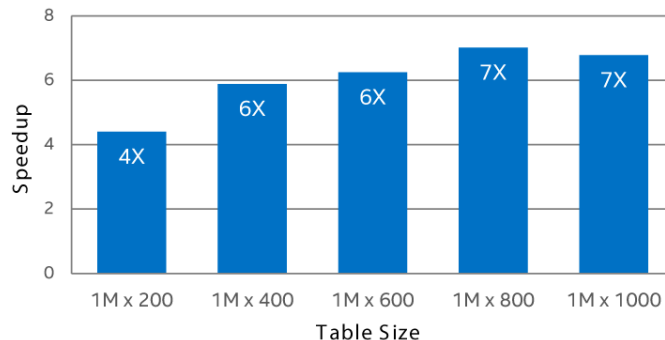
- ✓ さまざまなインテルプロセッサ向けに最適化済み
- ✓ 次世代プロセッサへの移行を簡単化

## 対応プロセッサ

- インテル® Atom™ プロセッサ
- インテル® Core™ プロセッサ
- インテル® Xeon® プロセッサ、
- インテル® Xeon Phi™ プロセッサ

DAAL の内部実装は IPP と MKL が提供する関数

PCA Performance Boost  
Using Intel® DAAL vs. Spark\* MLLib



Configuration Info - Versions: Intel® Data Analytics Acceleration Library 2016, CDH v5.3.1, Apache Spark\* v1.2.0; Hardware: Intel® Xeon® Processor E5-2699 v3, 2 Eighteen-core CPUs (45MB LLC, 2.3GHz), 128GB of RAM per node; Operating System: CentOS 6.6 x86\_64. PCA normalized input.

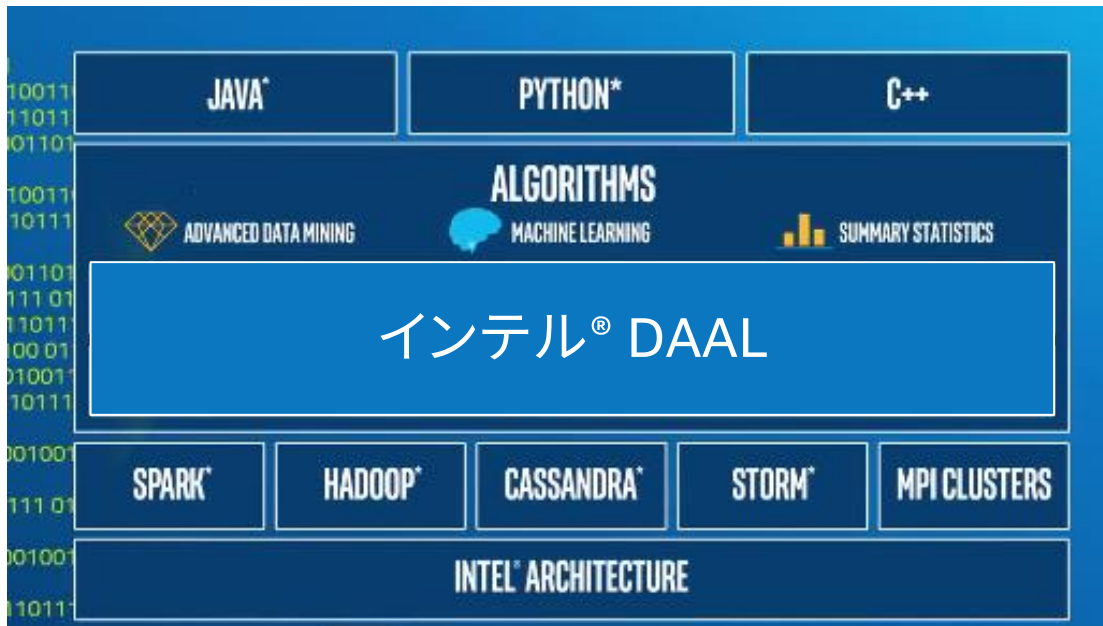
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. \*Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3E instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804.

# インテル® DAAL 対応言語

- Java、Python、C++ 言語に対応

Java などのマネージドコード環境でも、ネイティブコードの性能が得られます



# インテル® MKL に追加された機能 ディープ・ニューラル・ネットワーク (DNN)

- 人間の脳細胞を模倣した構成を持つ機械学習システム
- 幾つかの層に分けられ、それぞれで異なる処理を実装

従来のニューラル・ネットワークでは判断基準を教える必要がある  
例:パンダの画像を認識 → 人間がパンダの特徴を教える

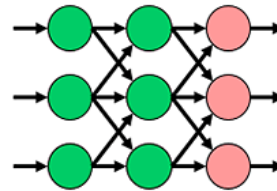
DNN ではコンピューター自身が判断基準を学習することができる  
例:パンダの画像を認識 → マシン自身がパンダの特徴を学習

活用例:

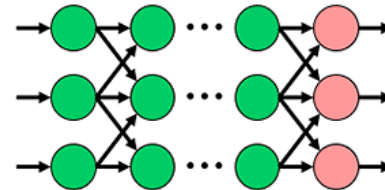
日本語翻訳 → より「日本語らしい」文章を作成

検索エンジン → より検索意図に近い情報を表示

従来のニューラルネットワーク



Deep Learning



出典: NTT DATA (本格化する「人工頭脳」のビジネス活用)  
([http://www.nttdata.com/jip/ja/insights/trend\\_keyword/2013110701.html](http://www.nttdata.com/jip/ja/insights/trend_keyword/2013110701.html))



# DNNの実装に使用される一般的なフレームワーク

- Caffe:  
ディープラーニング向け  
フレームワーク
- 画像認識に関する  
コミュニティが活発

Caffe 上で インテル® MKL の数学関数を利用することが可能

## Caffe

Deep learning framework by the [BVLC](#)

Created by [Yangqing Jia](#)  
Lead Developer  
[Evan Shelhamer](#)

[View On GitHub](#)

## Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center ([BVLC](#)) and by community contributors. [Yangqing Jia](#) created the project during his PhD at UC Berkeley. Caffe is released under the [BSD 2-Clause license](#).

Check out our web image classification [demo!](#)

### Why Caffe?

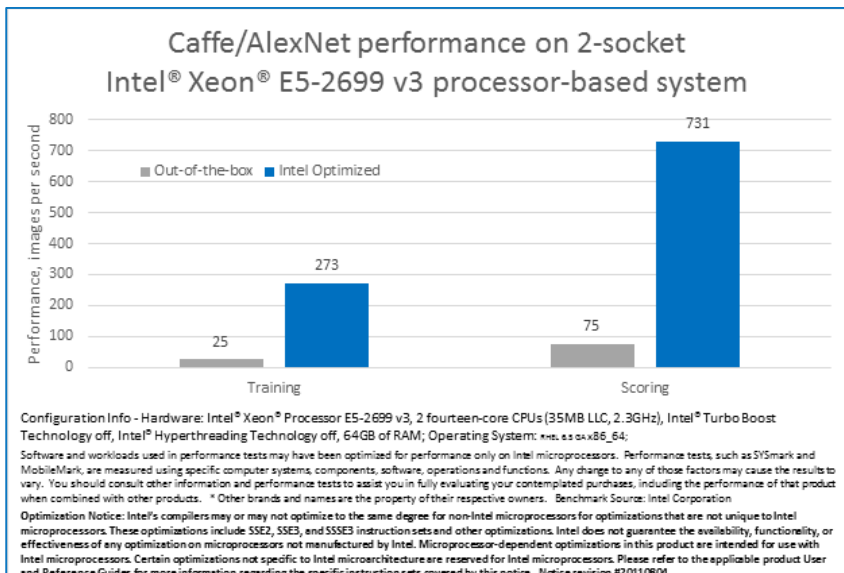
**Expressive architecture** encourages application and innovation. Models and optimization are defined by configuration without hard-coding. Switch between CPU and GPU by setting a single flag to train on a GPU machine then deploy to commodity clusters or mobile devices.

**Extensible code** fosters active development. In Caffe's first year, it has been forked by over 1,000 developers and had many significant changes contributed back. Thanks to these contributors the framework tracks the state-of-the-art in both code and models.

**Speed** makes Caffe perfect for research experiments and industry deployment. Caffe can process over **60M images per day** with a single NVIDIA K40 GPU\*. That's 1 ms/image for inference and 4 ms/image for learning. We believe that Caffe is the fastest convnet implementation available.

出典: <http://caffe.berkeleyvision.org/>

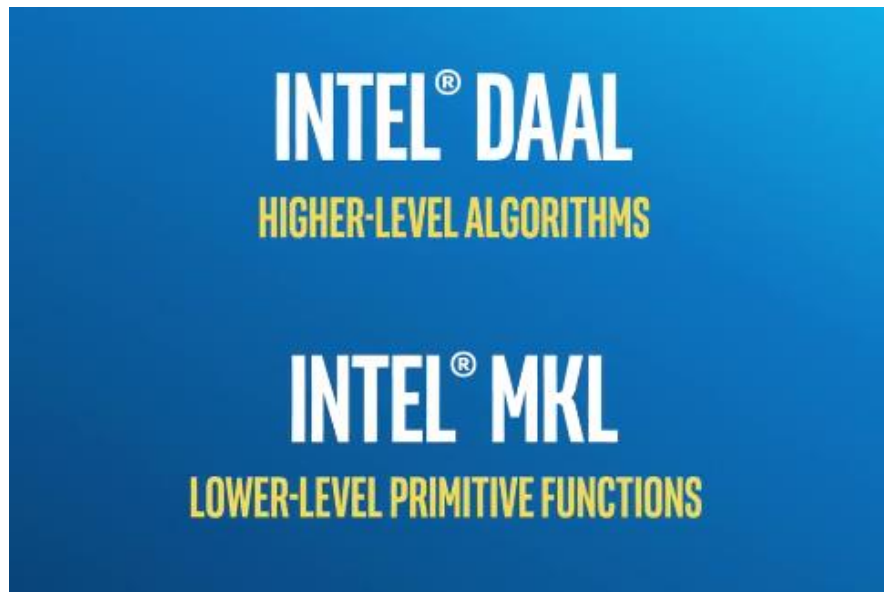
# インテル® MKL (DNN) の性能



- 2つの処理を高速化
- ベクトル化と並列化により、**学習スピード**の最適化
- 特徴の**分類スピード**の最適化

AVX 2 以上の命令セットを有する  
プロセッサをサポート  
(Haswell 以降)

# インテル® DAAL とインテル® MKL の違い



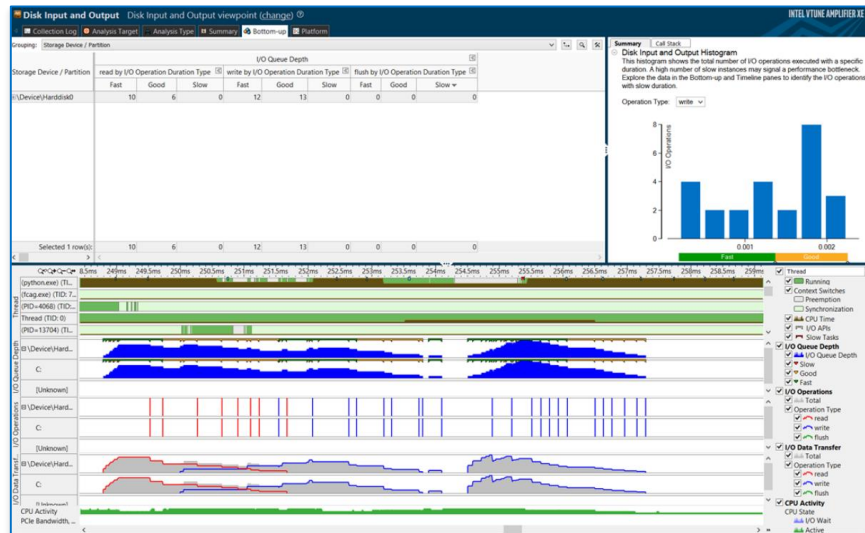
インテル® DAAL:  
データ分析に即使用できる  
様々なアルゴリズムを提供

インテル® MKL:  
ニューラル・ネットワークの  
実装を支援する関数群を提供

# ディスクアクセスに関する解析を行う

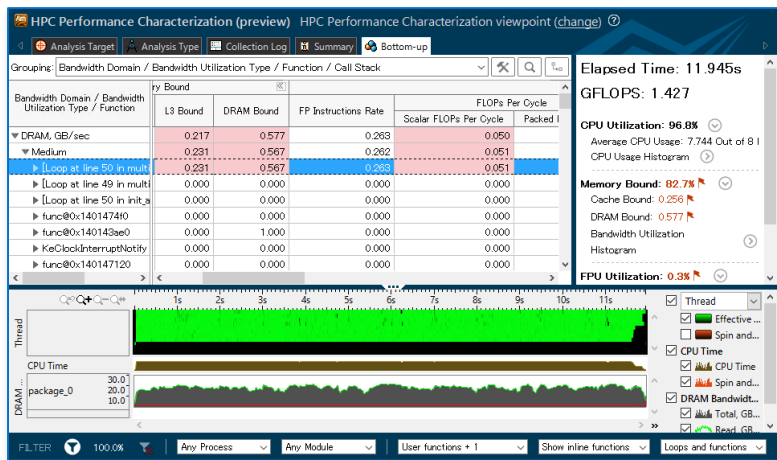
SSD や HDD への読み書きに関する解析を行う Disk Input and Output Analysis が追加されています。この機能を使用することで、読み込み遅延、書き込み遅延の発生を検出し、どのプロセス/モジュール/スレッドが発生原因であるかを素早く特定することができます。

SSD デバイスが複数ある場合や、ドライブが複数ある場合、それらを区別して表示させることができますようになっています。



# HPC 向けの新しい解析タイプ

- HPC Performance Characterization Analysis は、HPC 分野で有用な情報である GFLOPs や、関数/ループごとの CPU 使用率や CPU 使用効率、メモリー/キャッシュに関する情報、1 サイクルあたりの FLOPs、ベクトル化状況を確認することができます。



# 詳細な解析をする前のスナップショット

Step 1 –  
スナップショット



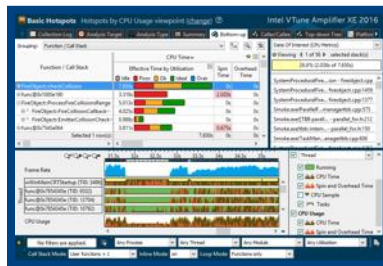
Application

MPI

Storage



Step 2 – 詳細な解析



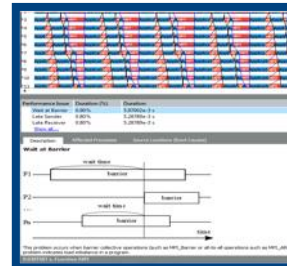
Intel® VTune™  
Amplifier

Application &  
Storage Profiling  
& Analysis

Vector...	Efficiency	Gain...	VL (...)	Compiler Es...
AVX2	~54%	2,15x	4	2,15x
AVX2			4	2,15x
AVX512			16	3,20x
AVX512			16	2,70x
AVX2	~100%	13,54x	8	<13,54x

Intel® Advisor

Vectorization  
Optimization  
& Thread Prototyping



Intel® Trace  
Analyzer &  
Collector

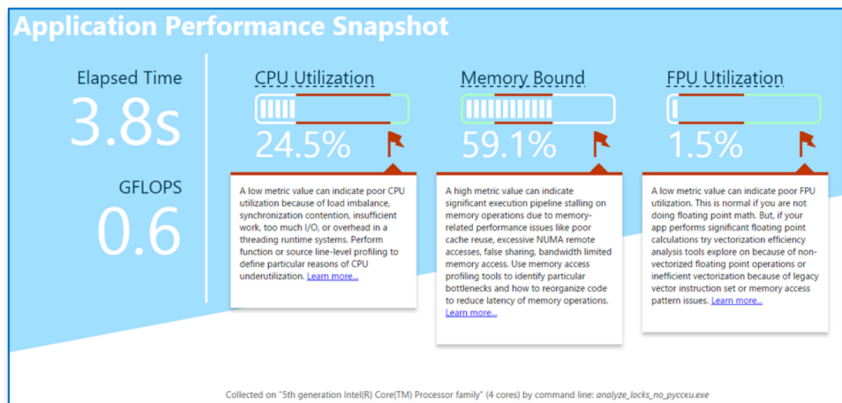
MPI Profiling  
& Analysis



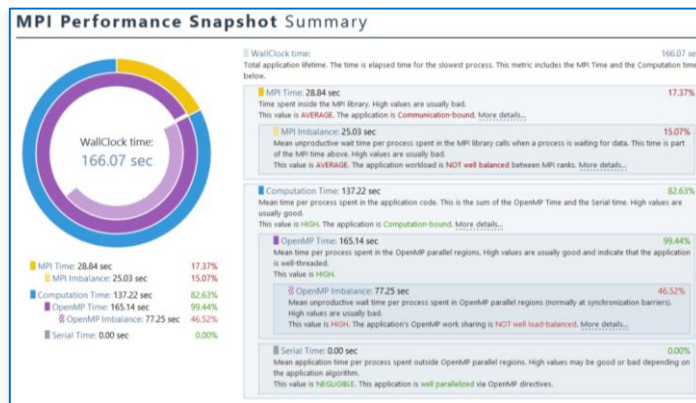
# よりシンプルに全体のパフォーマンスを確認

インテル® Performance Snapshot はアプリケーション全体のパフォーマンスを簡単に表示することができます。

そもそも本格的な解析が必要かを、事前に素早く確認することができます。



インテル® Performance Snapshot  
コードの近代化に関する情報を表示

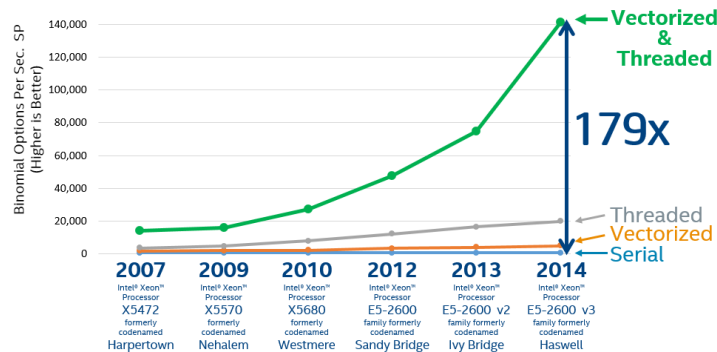


MPI Performance Snapshot  
パフォーマンスのスケールを表示

# インテル® AVX512 向けの最適化

- インテル® Xeon Phi™ プロセッサ (開発コード名: Knights Landing) を始めに、今後多くのインテル® AVX512 命令セットをサポートするプロセッサがリリースされていきます。
- 今日、コードの近代化を行うことでインテル® AVX512 命令セットや多くのコアが搭載されたプロセッサ向けの、将来にも有効な最適化を行うことができます。

必要な作業は  
ベクトル化 + マルチスレッド化



# 高速なコードを素早く開発: インテル® Advisor

最新プロセッサで性能を出すためにはベクトル化とマルチスレッド化が必須

- さらに、将来のプロセッサではより差が顕著に

## ベクトル化でおきる問題:

- インテル® AVX2 を使用したのに速くならない
- そもそもどこをベクトル化すれば良い?
- 最新プロセッサ用の組込み関数を使用する必要がある?
- コンパイラーのベクトル化レポートのどこを見れば良い?

## マルチスレッド化でおきる問題:

- マルチスレッド化したけれど速くならない
- スレッド数を増やしたら性能劣化する
- マルチスレッド化に時間がかかってしまう

これらの問題、疑問をインテル® Advisor が解決します

# 正しいベクトル化のためには正しい情報を

ベクトル化されたループをフィルター

トリップカウントを表示

ベクトル化を妨げる原因を特定

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops		
							Vecto...	Efficiency	Vector L..
[loop at stl_algo.h:4740 in std::tr...		0.170s	0.170s		Scalar	non-vectorizable loop ins...			
[loop at loopstl.cpp:2449 in s234_]	2 Ineffective peeled/rem...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
[loop at loopstl.cpp:2449 in s...		0.150s	0.150s	12	Vectorized (Body)		AVX		4
[loop at loopstl.cpp:2449 in s...		0.020s	0.020s	4	Remainder				
[loop at stl_algo.h:7900 in vas_]		0.170s	0.170s	500	Scalar	vectorization possible but...			4
[loop at loopstl.cpp:3509 in s2...	1 High vector register ...	0.160s	0.160s	12	Expand	Expand	AVX	~69%	8
[loop at loopstl.cpp:3891 in s279_]	2 Ineffective peeled/rem...	0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%	8
[loop at loopstl.cpp:6249 in s414_]		0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
[loop at numeric.h:247 in std...	1 Assumed dependency...	0.150s	0.150s	49	Scalar	vector dependence preve...			

ホットループを表示

ベクトル化の問題を表示

ベクトル命令の世代を表示

ベクトル化後の効率を表示

高速なコードを素早く開発

# ベクトル化したコードの効率性を インテル® Advisor で評価

Where should I add vectorization and/or threading parallelism? Intel Advisor XE 2016

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Elapsed time: 54.44s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops
							Vector... Efficiency Vector L...
[loop at stl_algo.h:4740 in std:tr...		0.170s	0.170s	12; 4	Scalar	non-vectorizable loop ins...	
[loop at loopstl.cpp:2449 in s234_]	2 Ineffective peeled/rem...	0.170s	0.170s	12; 4	Collapse	Collapse	
[loop at loopstl.cpp:2449 in s...		0.150s	0.150s	12	Vectorized (Body)		AVX ~100% 4
[loop at loopstl.cpp:2449 in s...		0.020s	0.020s	4	Remainder		AVX ~100% 4
[loop at loopstl.cpp:7900 in vas_]		0.170s	0.170s	500	Scalar	vectorization possible but...	
[loop at loopstl.cpp:3509 in s2...	1 High vector register ...	0.160s	0.160s	12	Expand	Expand	AVX ~69% 8
[loop at loopstl.cpp:3891 in s279_]	2 Ineffective peeled/rem...	0.150s	0.150s	125; 4	Expand	Expand	AVX ~96% 8
[loop at loopstl.cpp:6249 in s414_]		0.150s	0.150s	12	Expand	Expand	AVX ~100% 4
[loop at stl_numeric.h:247 in std...	1 Assumed dependency...	0.150s	0.150s	49	Scalar	vector dependence preve...	

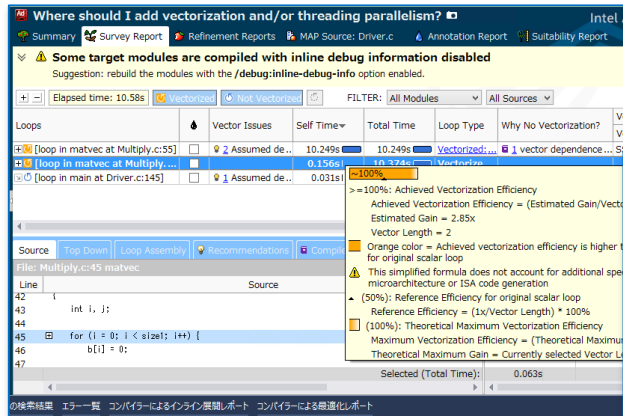
Top Down Source Loop Assembly Assistance Recommendations Compiler Diagnostic Details

File: loopstl.cpp:3509 s273\_

Line	Source	Total Time	%	Loop Time	%
3504	forttime (&t1);				
3505	i_1 = *ntimes;				
3506	for (nl = 1; nl <= i_1; ++nl)	0.010s		0.200s	
	[loop at loopstl.cpp:3506 in s273_]				
	Scalar Loop. Not vectorized: inner loop was already vectorized No loop transformations were applied				
3507	{				
3508	i_2 = *n;				
3509	for (i_1 = 1; i_1 <= i_2; ++i_1)	0.010s		0.160s	
	[loop at loopstl.cpp:3509 in s273_]				
	Vectorized AVX Loop processing Float32; Float64; Int32 data type(s) having Inserts; Extracts; Masked St...				
	Selected (Total Time):	0.010s			

Vectorized Loops			
	Vector...	Efficiency	Vector L...
op ins ...			
	AVX	~100%	4
	AVX	~100%	4
le but ...			4
	AVX	~69%	8
	AVX	~96%	8
	AVX	~100%	4
preve ...			

# ベクトル化の効率性に関する情報を一度に表示



Vectorized Loops				Instruction Set...	
Vector ISA	Efficiency	Gain Estimate	VL (Vector Length)	Traits	Data T...
SSE2	~100%	2.85x	2		Float32...
	~71%	1.42x	2		Float64

~100%

>= 100%: Achieved Vectorization Efficiency

Achieved Vectorization Efficiency = (Estimated Gain/Vector Length) \* 100%

Estimated Gain = 2.85x

Vector Length = 2

Orange color = Achieved vectorization efficiency is higher than reference efficiency for original scalar loop

⚠ This simplified formula does not account for additional speed-ups related to microarchitecture or ISA code generation

▲ (50%): Reference Efficiency for original scalar loop

Reference Efficiency = (1x/Vector Length) \* 100%

■ (100%): Theoretical Maximum Vectorization Efficiency

Maximum Vectorization Efficiency = (Theoretical Maximum Gain/Vector Length) \* 100%

Theoretical Maximum Gain = Currently selected Vector Length = 2



# ループのベクトル化

1. ベクトル化可能だがベクトル化できていないループ  
少しの修正でベクトル化できる可能性があります。
2. ベクトル化されているがあまり性能が上がらない  
性能を容易に向上できる可能性があります。
3. ベクトル化されているがデータレイアウトの影響で  
性能がでない  
高速なデータ参照方法を使用します。
4. ベクトル化されていて、正しく性能を発揮する  
ほかの個所の最適化に進みます。

# ベクトル化できていないループの例

- ループの依存関係の存在
- エイリアスによる依存関係の可能性なのか、実際に依存関係があるか確認します。



```
10 for(i=0; i<NUM; i++){
11     a[i] = i%10;
12 }
13
14 for(j=1; j<NUM; j++){
15     a[j] = a[j-1] + x;
16 }
```

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

Loops	🔥	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?
<a href="#">[loop in fCalcInteraction_ShanChen at lbpFOR...</a>	<input type="checkbox"/>	1 Ineffective peeled/remainder loop(...)	0.894s	0.894s	Peeled/Remain...	
<a href="#">[loop in fGetSpeedSite at lbpGET.cpp:321]</a>	<input type="checkbox"/>	2 Ineffective peeled/remainder loop(...)	0.796s	0.796s	Vectorized (Bo...	
<a href="#">[loop in fPropagationSwap at lbpSUB.cpp:13 ...]</a>	<input type="checkbox"/>	2 Assumed dependency present	0.673s	1.988s	Scalar	vector dependence preve...

## Issue: Assumed dependency present

The compiler assumed there is an anti-dependency (Write after read – WAR) or true dependency (Read after write – RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

🔍 Recommendation: Confirm dependency is real

Confidence: 🌟 Need More Data

There is no confirmation that a real dependency is present in the loop. To confirm: Run a Dependencies analysis.

# インテル® Advisor によるアドバイス機能

## Issue: Assumed dependency present

The compiler assumed there is an anti-dependency (Write after read – WAR) or true dependency (Read after write – RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

### ⊟ Enable vectorization

Potential performance gain: Information not available until Beta Update release

Confidence this recommendation applies to your code: Information not available until Beta Update release

The Correctness analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the `restrict` keyword or a [directive](#).

ICL/ICC/ICPC Directive	IFORT Directive	Outcome
<code>#pragma simd</code> or <code>#pragma omp simd</code>	<code>!DIR\$ SIMD</code> or <code>!\$OMP SIMD</code>	Ignores all dependencies in the loop
<code>#pragma ivdep</code>	<code>!DIR\$ IVDEP</code>	Ignores only vector dependencies (which is safest)

### Read More:

- [User and Reference Guide for the Intel C++ Compiler 15.0](#) > [Compiler Reference](#) > [Pragmas](#) > [Intel-specific Pragma Reference](#) >
  - `ivdep`
  - `omp simd`

エイリアスによる依存関係の可能性がベクトル化を妨げている場合、インテル® Advisor は修正案を提供します。  
ここでは `#pragma simd` や `#pragma ivdep` の使用を提案されました。

# メモリー・アクセス・パターン解析

Where should I add vectorization and/or threading parallelism?

Summary Survey Report Refinement Reports Annotation

Elapsed time: 8,52s Vectorized Not Vectorized FILTER: All

Function Call Sites and Loops

Function Call Sites and Loops	Vector Issues	Self time	Loop type	Vectorization?
[loop at fractal.cpp:179 in <lambda1>::op ...]	4 High vector...	0,013sI	12,020s	Collapse Collapse
[loop at fractal.cpp:179 in <lambda1>::o ...]	1 Serialized use ...	0,013sI	11,281s	Vectorized (Body)
[loop at fractal.cpp:179 in <lambda1>::o ...]	2 Data type co ...	0,000sI	0,163sI	Peeled
[loop at fractal.cpp:179 in <lambda1>::o ...]	2 Data type co ...	0,000sI	0,576sI	Remainder
[loop at fractal.cpp:177 in <lambda1>::oper ...]	2 Data type co ...	0,010sI	12,030s	Scalar

対象ループのチェックボックスに  
チェックを入れ、

## 2.2 Check Memory Access Patterns

Identify and explore complex memory accesses for marked loops. Fix the reported problems.



Command Line

[Check Memory Access Patterns] を使用して、  
メモリー・アクセス・パターンを解析します。

# データレイアウトが問題でベクトル化の性能が発揮できない例

Loops	Vector Issues	Vectorized Loops	
		Vector ISA	Efficiency
⊕ [loop in fGetEquilibriumF at lbpSUB.cp...]	2 Inefficient memory access patterns present	AVX	~56%
⊖ [loop in fPropagationSwap at lbpSUB.cp...]	2 Assumed dependency present		
⊖ [loop in fSiteFluidCollisionBGK at lbpBG...]			
⊖ [loop in IIO_OUTPUT]			
⊖ [loop in fSiteFluidCollisionBGK at lbpBG...]			
⊕ [loop in fCollisionBGK at lbpBGK.cpp:840]	1 Ineffective peeled/remainder loop(s) present	AVX	~100%
⊕ [loop in fGetOneMassSite at lbpGET.cpp:...	1 Ineffective peeled/remainder loop(s) present	AVX	~35%

この例では 84% が非ユニットストライドと検出されたため、構造体を Structure of Array に変更することを検討

Source | Top Down | Loop Assembly | Recommendations | Compiler Diagnostic Details

**Issue: Inefficient memory access patterns present**

There is a high of percentage memory instructions with irregular (variable or random) stride and... Improve performance by investigating and handling accordingly.

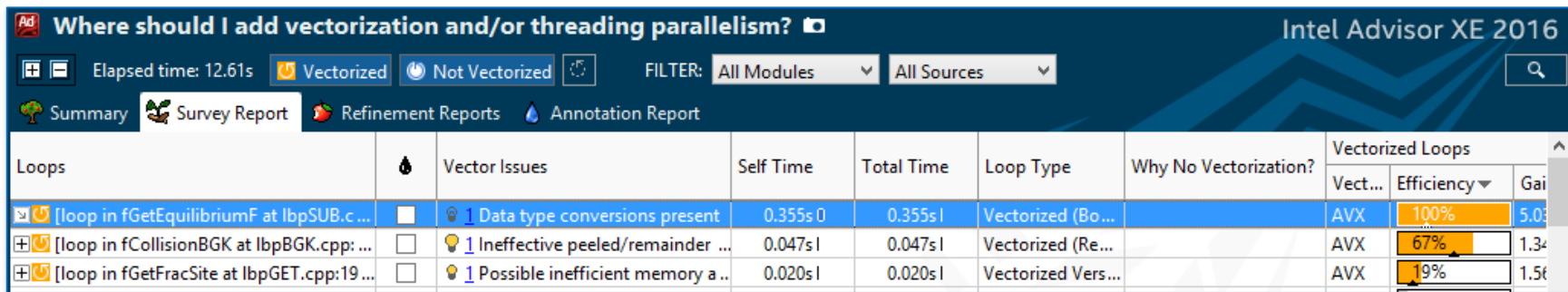
⊖ Recommendation: Use SoA instead of AoS      Confidence: Low

An array is the most common type of data structure containing a contiguous collection of items that can be accessed by an ordinal index. You can organize this data as an array of structures (AoS) or as a structure of arrays (SoA). While AoS organization is excellent for encapsulation, it can hinder effective vector processing. To fix: Rewrite code to organize using SoA instead of AoS.

16% / 84% / 0%      Mixed strides

- 16%: percentage of memory instructions with unit stride or stride 0 accesses  
Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration  
Stride 0 = Instruction accesses the same memory from iteration to iteration
- 84%: percentage of memory instructions with fixed or constant non-unit stride accesses  
Constant stride (stride N) = Instruction accesses memory by N elements from iteration to iteration  
Example: for the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4\*sizeof(double)) with each iteration
- 0%: percentage of memory instructions with irregular (variable or random) stride accesses  
Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration  
Typically observed for indirect indexed array accesses, for example, a[index[i]]  
- gather (irregular) accesses, detected for v(p)gather\* instructions on AVX2 Instruction Set Architecture

# ベクトル化個所が性能を発揮しているか、 インテル® Advisor を用いて確認



Intel Advisor XE 2016

Elapsed time: 12.61s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources

Summary | Survey Report | Refinement Reports | Annotation Report

Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops		
						Vect...	Efficiency	Gai
[loop in fGetEquilibriumF at lbpSUB.c ...]	1 Data type conversions present	0.355s	0.355s	Vectorized (Bo...	AVX	100%	5.03	
[loop in fCollisionBGK at lbpBGK.cpp: ...]	1 Ineffective peeled/remainder ...	0.047s	0.047s	Vectorized (Re...	AVX	67%	1.34	
[loop in fGetFracSite at lbpGET.cpp:19 ...]	1 Possible inefficient memory a...	0.020s	0.020s	Vectorized Vers...	AVX	19%	1.54	

- インテル® AVX でベクトル化され、効率が高く、短時間で処理できるループになりました。
- 数値的、視覚的に、最適化の効果を確認することで作業効率を高めることができます。

# まとめ

- より高度な最適化を短時間で行うことができます。
- 初心者も上級者もツールを使う大きなメリットがあります。
- 数年先まで有効なコードを作ることができるようになります。



# 法務上の注意書きと最適化に関する注意事項

本資料の情報は、現状のまま提供され、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。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証（特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他、知的財産権の侵害への保証を含む）をするものではありません。

性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサ用に最適化されていることがあります。SYSmark\* や MobileMark\* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。

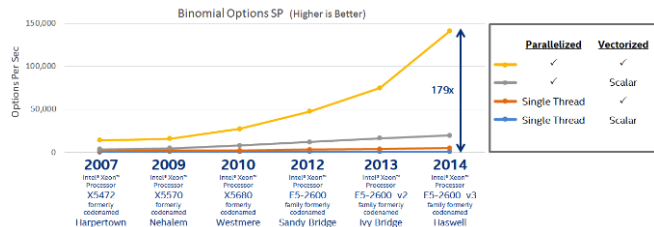
© 2016 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Xeon、Intel Xeon Phi、VTune は、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。\*その他の社名、製品名などは、一般に各社の商標または登録商標です。

## 最適化に関する注意事項

インテル® コンパイラーでは、インテル® マイクロプロセッサに限定されない最適化に関して、他社製マイクロプロセッサ用に同等の最適化を行えないことがあります。これには、インテル® ストリーミング SIMD 拡張命令 2、インテル® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサ依存の最適化は、インテル® マイクロプロセッサでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサ用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。

注意事項の改訂 #20110804

# 補足資料: 2 項オプション SP のシステム構成



## 最適化に関する注意事項

Intel® コンパイラーでは、Intel® マイクロプロセッサに限定されない最適化に関して、他社製マイクロプロセッサ用に同等の最適化を行えないことがあります。これには、Intel® ストリーミング SIMD 拡張命令 2、Intel® ストリーミング SIMD 拡張命令 3、Intel® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。Intel は、他社製マイクロプロセッサに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサ依存の最適化は、Intel® マイクロプロセッサでの使用を前提としています。Intel® マイクロアーキテクチャーに限定されない最適化のなかにも、Intel® マイクロプロセッサ用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。

注意事項の改訂 #20110804

Intel 社内での測定値。

## システム構成

プラットフォーム	スケールン されていない コアクロックの 周波数	コア/ ソケット	ソケット 数	L1 データ キャッシュ	L1 命令 キャッシュ	L2 キャッシュ	L3 キャッシュ	メモリー 周波数	メモリー アクセス	H/W プリフェッチ 有効	HT 有効	ターボ 有効	C ステート	OS	カーネル	コンパイラー・ バージョン	
Intel® Xeon® プロセッサ 5472	3GHz	4	2	32K	32K	12MB	なし	32GB	800MHz	UMA	Y	N	N	無効	Fedora* 20	3.11.10- 301.fc20	icc 14.0.1
Intel® Xeon® プロセッサ X5570	2.90GHz	4	2	32K	32K	256K	8MB	48GB	1333MHz	NUMA	Y	Y	Y	無効	Fedora* 20	3.11.10- 301.fc20	icc 14.0.1
Intel® Xeon® プロセッサ X5680	3.33GHz	6	2	32K	32K	256K	12MB	48MB	1333MHz	NUMA	Y	Y	Y	無効	Fedora* 20	3.11.10- 301.fc20	icc 14.0.1
Intel® Xeon® プロセッサ E5-2690	2.90GHz	8	2	32K	32K	256K	20MB	64GB	1600MHz	NUMA	Y	Y	Y	無効	Fedora* 20	3.11.10- 301.fc20	icc 14.0.1
Intel® Xeon® プロセッサ E5-2697v2	2.70GHz	12	2	32K	32K	256K	30MB	64GB	1867MHz	NUMA	Y	Y	Y	無効	Fedora* 20	3.11.10- 301.fc20	icc 14.0.1
製品ファミリー 開発コード名 Haswell	2.20GHz	14	2	32K	32K	256K	35MB	64GB	2133MHz	NUMA	Y	Y	Y	無効	Fedora* 20	3.13.5- 202.fc20	icc 14.0.1

