

インテル® ソフトウェア開発製品 利用方法

Revision 6.1K

エクセルソフト株式会社

黒澤 一平

XLSOFT



インテル® Parallel Studio XE Cluster Edition 概要

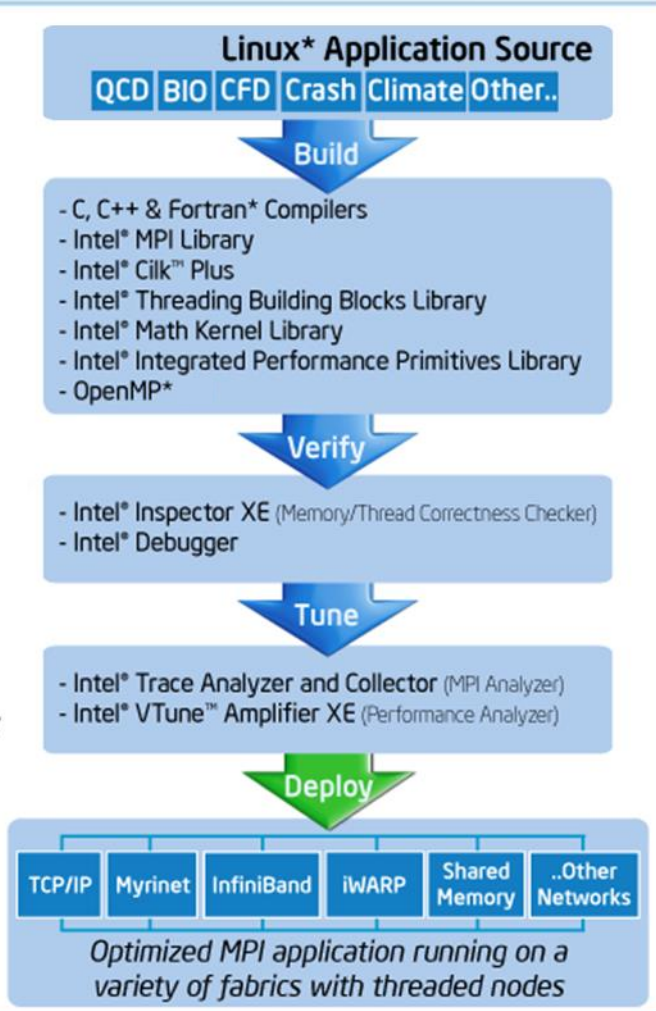
インテル® Parallel Studio XE 2016 Cluster Edition

クラスターシステム向けの総合開発ツール



Scale Forward,
Scale Faster

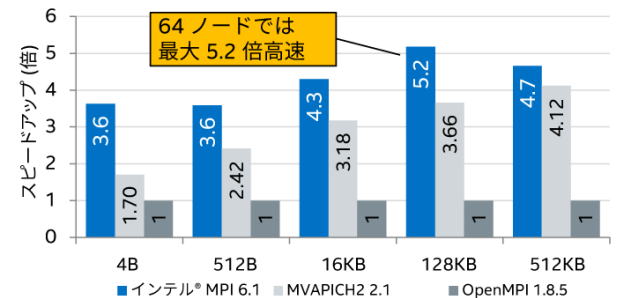
for HPC Clusters



それぞれのフェーズに対応する
ハイパフォーマンス・ツール

- ・高速化に特化したC、C++、Fortran コンパイラー
- ・最適化されたライブラリー
- ・エラー検出ツール、デバッガー
- ・性能解析ツール







インテル® MPI ライブラリー 5.1 の優れたパフォーマンス
64 ビットの Linux* における相対 (相乗平均) MPI レイテンシー・ベンチマーク
64 ノードの 1792 プロセス (InfiniBand + 共有メモリー) (数値が大きいほど高性能)



システム構成：システム構成：ハードウェア：CPUデュアルインテル® Xeon® プロセッサ E5-2697v3@2.60GHz、64GB RAM、インターコネクト：Mellanox Technologies® MT27500 Family [ConnectX-3]、ソフトウェア：OS、OpenSUSE、インテル® C/C++ コンパイラー XE 15.0.3、インテル® MPI ライブラリー 5.1、インテル® MPI Benchmarks 4.1。性能に関するテストに使用されるソフトウェアワークロードは、性能がインテル® マイクロプロセッサ向け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・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勧めます。ベンチマークの出典：インテル® コーポレーション

最適化に関する注意事項：インテル® コンパイラーは、互換マイクロプロセッサ向けには、インテル製マイクロプロセッサ向けと同レベルの最適化が行えない可能性があります。これは、インテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2)、インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3)、ストリーミング SIMD 拡張命令 3 補完命令 (SSSE3) 命令セットに関する最適化によるものです。インテルは、インテル製マイクロプロセッサに対して、最適化の提供、検証、効果を保証していません。本製品のマイクロプロセッサ固有の最適化は、インテル製マイクロプロセッサでの使用を目的としています。インテル® マイクロプロセッサ向けに非固有の特定の最適化は、インテル製マイクロプロセッサ向けに予約されています。この注意事項の適用対象である特定の命令セットに関する詳細は、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。

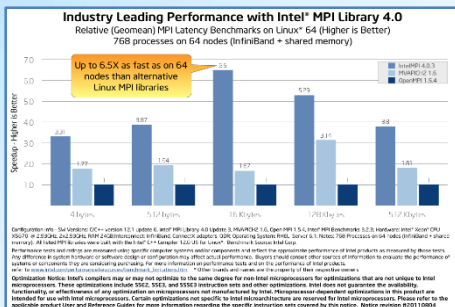
インテル® Parallel Studio XE 2016 の構成詳細

フェーズ	製品名	機能	利点
ビルド	 インテル® MPI ライブラリー	ハイパフォーマンスな MPI ライブラリー	<ul style="list-style-type: none"> ハイパフォーマンス、スケーラビリティ、インターコネクットの独立性、実行時のファブリック選択、アプリケーション・チューニングを実現
	 インテル® Composer XE	C/C++、Fortran コンパイラとパフォーマンス・ライブラリー <ul style="list-style-type: none"> インテル® スレッディング・ビルディング・ブロック インテル® Cilk™ Plus インテル® インテグレートッド・パフォーマンス・プリミティブ インテル® マス・カーネル・ライブラリー 	<ul style="list-style-type: none"> マルチコアと将来のメニーコアのパフォーマンスおよびスケーラビリティを引き出すアプリケーションを開発するためのソリューション
検証	 インテル® Inspector XE	コードの品質を高める動的なメモリー/スレッド解析とスタティック・セキュリティ解析	<ul style="list-style-type: none"> 生産性とコードの品質を高め、コストを削減し、早期にメモリー/スレッド/セキュリティの問題を発見 各クラスターノードで MPI に対応
検証 & チューニング	 インテル® トレース・アナライザー/コレクター	アプリケーションの正当性と動作を理解するための MPI パフォーマンス・プロファイラー	<ul style="list-style-type: none"> MPI プログラムのパフォーマンスを解析し、並列アプリケーションの動作と通信パターンを視覚化して、hotspot を特定
チューニング	 インテル® VTune™ Amplifier XE	アプリケーションのパフォーマンスとスケーラビリティを最適化するパフォーマンス・プロファイラー	<ul style="list-style-type: none"> 従来の推測作業を排除し、短時間で容易にパフォーマンスとスケーラビリティのボトルネックを特定 各クラスターノードで MPI に対応
並列化、ベクトル化	 インテル® Advisor	並列化とマルチスレッド化のアドバイザー	<ul style="list-style-type: none"> マルチスレッド化すべき場所を特定 マルチスレッド化性能予測 ベクトル化状況を解析

インテル® Composer XE に含まれるライブラリー

製品名	機能	利点
インテル® マス カーネル ライブラリー (インテル® MKL)	ハイパフォーマンスな数値演算ライブラリー ・LAPACK, ScaLAPAC ・BLAS, Sparse BLAS, PBLAS ・PARDISO ・FFT など	・高速化、並列化されており、プロセッサの性能を最大限活用することができる
インテル® インテグレートッド パフォーマンス プリミティブ (インテル® IPP)	ハイパフォーマンスなマルチメディアライブラリー ・信号処理、音声処理 ・画像、動画処理 ・データ処理 ・暗号化	・基関数を組み合わせるだけで様々な高速な関数を作成することができる
インテル® スレッディングビルディング ブロック (インテル® TBB)	C++言語用のマルチスレッド化対応テンプレートライブラリー	・マルチスレッド化のための抽象化されたテンプレート、コンテナ、およびクラスを使用することでハイパフォーマンスなマルチスレッドアプリケーションの作成を実現

それぞれの性能、インターフェース



インテル® MPI ライブラリー



インテル® Composer XE



インテル® Inspector XE



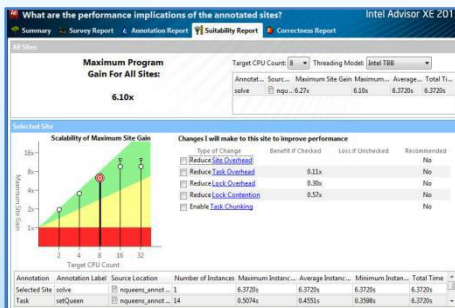
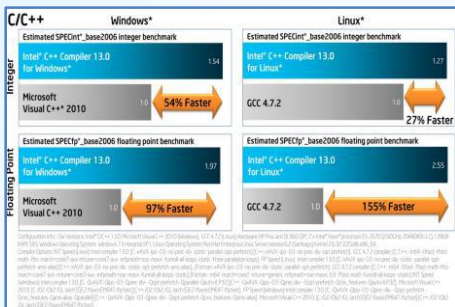
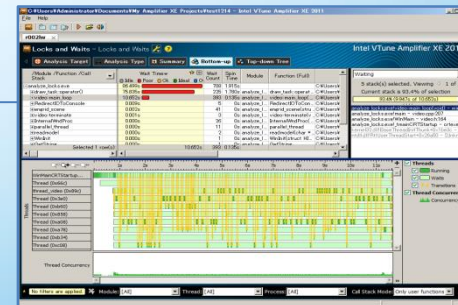
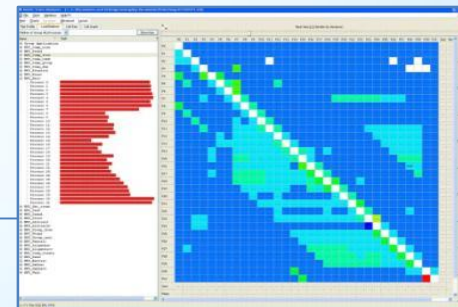
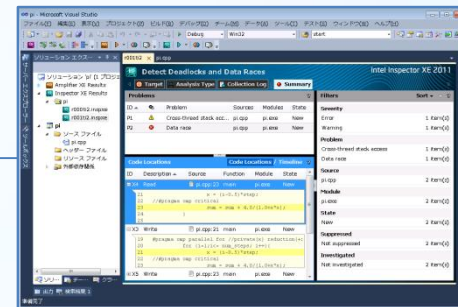
インテル® トレース・アナライザー/コレクター



インテル® VTune™ Amplifier XE



インテル® Advisor XE



インテル® C/C++, Fortran Composer XE (コンパイラー) コンパイル方法およびコンパイラーオプションの紹介

ドキュメント凡例、用語

icc	インテル® C コンパイラー
icpc	インテル® C++ コンパイラー
ifort	インテル® Fortran コンパイラー
mpiicc	mpicc のインテル® C コンパイラー版
mpicpc	mpic++, mpicxx のインテル® C++ コンパイラー版
mpifort	mpif77, mpif90 のインテル® Fortran コンパイラー版
<Install dir>	各製品のインストールディレクトリー
必須	製品を使用するための必須設定や基礎情報
参考	機能一覧や、使用の参考となる情報

使用頻度の高いオプション例 (Linux 版)

自動並列化(マルチスレッド)	<code>mpiicc -parallel <ファイル></code>
OpenMP:	<code>mpiicc -openmp <ファイル></code>
ループの最適化	<code>mpiicc -O3 <ファイル></code>
ベクトル化 (Sandy Bridge † 対応)	<code>mpiicc -xAVX <ファイル></code>
プロシージャ間の最適化	<code>mpiicc -ipo <ファイル></code>
インテル® Trace Collector をリンク	<code>mpiicc -trace -g <ファイル></code>

※詳細はインテル® Trace Analyzer/ Collector の使用方法にて説明

推奨設定:

```
mpiicc -O3 -xAVX -ipo [-openmp] [-parallel] <ファイル>
```

※ mpiicc の他に、icc や ifort、mpiifort を使用した場合も同様
(ただし -trace はMPI版コンパイラ使用時のみ)

主要なコンパイラオプション一覧

オプション	概要
-O0	全ての最適化オプションを無効にする
-O1, -O2, -O3	ループの最適化 (3が最も強力)
-x, -ax, -m	ベクトル化、各プロセッサ向けの最適化
-ipo	プロシージャ間の最適化
-qopenmp	OpenMP を利用
-parallel	自動並列化
-prof_gen, -prof_use	プロファイルによる最適化 (Profile Guided Optimization [PGO])
-guide	ガイド付き自動並列化
-qopt-report-phase= <i>n1, n2</i>	最適化フェーズ <i>n1, n2</i> ... の最適化レポート※を生成 <i>all, loop, vec, par, cg, ipo, pgo, offload</i>
-fast	-ipo、-O3、-no-prec-div、-static、-xHOST を有効にする
-fp-model < <i>model</i> >	浮動小数点数演算の精度を制御 (<i>model</i> は、 <i>strict, precise, fast</i> 等選択可)

※ 最適化レポートはソースファイル毎に、filename.optreport

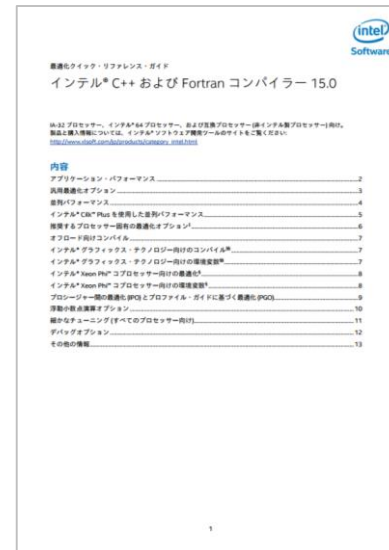
主要なコンパイラーオプション一覧つづき

オプション	概要
-prec-div	浮動小数点数演算の精度を向上させる
-static-intel, -shared-intel	インテル® ライブラリーのリンク方法を指定(初期値はstatic)
-mkl [=lib]	インテル® MKL をリンクする (=lib は以下を選択可能)
使用例: -mkl=cluster	<p><i>parallel</i> マルチスレッド化されたインテル® MKL のスレッドをリンクする。lib 未指定時のデフォルト</p> <p><i>sequential</i> シングルスレッドのインテル® MKL のスレッドをリンクする</p> <p><i>cluster</i> クラスターに対応した関数のインテル® MKL と、その他シングルスレッドのインテル® MKL をリンクする</p>

インテル® Composer XE 最適化クイック・リファレンス・ガイド

- 最適化の手順と、基本的なオプションが一覧になったドキュメント
http://jp.xlsoft.com/documents/intel/compiler/Quick-Reference-Card-Intel-Compilers-v15_JA.pdf

- ループの最適化
- ベクトル化
- プロシージャ間の最適化 (IPO)
プロファイルに基づく最適化 (PGO)
- ガイド付き自動並列化 (GAP)
- インテル® VTune™ Amplifier XE の利用



高度な最適化: -O3 (High-Level Optimization: HLO)

- ループの最適化
 - ループのアンロール、並べ替え、プリフェッチャの挿入
 - 自動ベクトル化オプション(-x, -ax)と併用することにより強力なデータ依存解析を行い、ベクトル化の可能性を高める

コンパイラーオプション	
-O0	最適化を無効にする
-O1	実行速度の最適化(コードサイズの増加なし)
-O2	実行速度の最適化(デフォルト)
-O3	高度な最適化

オプションの詳細 (-O)

Fortran: <https://software.intel.com/en-us/node/579292>

C/C++: <https://software.intel.com/en-us/node/581715>

自動ベクトル化: -m, -x, -ax (Vectorizer)

- スカラー演算を SIMD (Single Instruction Multiple Data) 演算に自動変換して、処理効率の良いコードを生成

例)

```
for ( i=1; i<=MAX; i++ )  
  c[i] = a[i] + b[i];
```

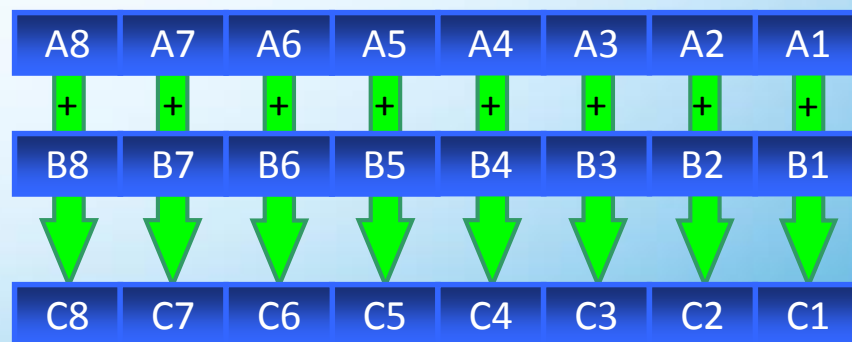
-no-vec (ベクトル化をしない)

-xAVX (Intel AVX 命令を使用)

スカラー演算: 1命令で1結果 × MAX 回



SIMD演算: 1命令で8結果 × (MAX/8)回



※ AVX、float の場合

自動ベクトル化オプション -m, -x, -ax の違い

オプション	概要
<code>-x<code></code> <code><code></code> =Host,AVX,SSE4.2,SSE4.1,SSE3_ATOM,SSSE3,SSE3,SSE2, CORE-AVX-I, CORE-AVX2, MIC-AVX512、他	インテルプロセッサ専用の最適化を行い、AVX、SSE4.2などの命令セットを生成する 例: <code>-xAVX</code> は AVX、SSE4.2、SSE4.1、SSSE3、SSE3、SSE2、SSE を生成
<code>-ax<code></code> <code><code></code> =AVX,SSE4.2,SSE4.1,SSSE3,SSE3,SSE2, CORE-AVX-I, CORE-AVX2, MIC-AVX512、他	インテルプロセッサ専用の最適化と、AVXやSSE4.2 等と、汎用命令セットを生成する 例: <code>-axAVX</code> は AVX、SSE4.2、SSE4.1、SSSE3、SSE3、SSE2、SSE と、汎用コードを生成
<code>-m<code></code> <code><code></code> =ia32, sse2, sse3, ssse3, sse4.1, sse4.2, avx	インテル互換プロセッサに対応する、ia32、sse2命令セットを生成する 例: <code>-mSSE4.1</code> は SSE4.1、SSSE3、SSE3、SSE2、SSE を生成

オプションの詳細 (-x)

Fortran: <https://software.intel.com/en-us/node/579311>

C/C++: <https://software.intel.com/en-us/node/581749>

-ax オプション補足

- 複数のプロセッサに対応させたい場合、対象プロセッサを複数設定することができる

例: -axAVX,SSE4.2,SSE3 は以下のように動作する

- AVX をサポートするプロセッサ: AVX までを実行
- SSE4.2 をサポートするプロセッサ: SSE4.2 までを実行
- SSE4.1 をサポートするプロセッサ: SSE3 までを実行
- SSE2 をサポートするプロセッサ: 汎用コードを実行

浮動小数点数演算の精度: -fp-model

- 浮動小数点数の演算は、コンパイラーの最適化により異なった結果を生じる場合がある
- precise や strict を指定した場合は、一部のベクトル化が抑制される (パフォーマンスと精度はトレードオフ)

ソースコードの記述

```
float t0, t1, t2  
t0 = 4.0f + 0.1f + t1 + t2
```

オプション	効果
-fp-model fast (デフォルト)	演算順序の変更を含む様々な最適化を行う $t0 = 4.1f + t1 + t2$
-fp-model precise	演算順序を変更しない、その他の最適化は行う $t0 = (4.1f + t1) + t2$
-fp-model strict	演算順序を変更せず、厳密に計算する $t0 = ((4.0f + 0.1f) + t1) + t2$

オプションの詳細 (-fp-model)

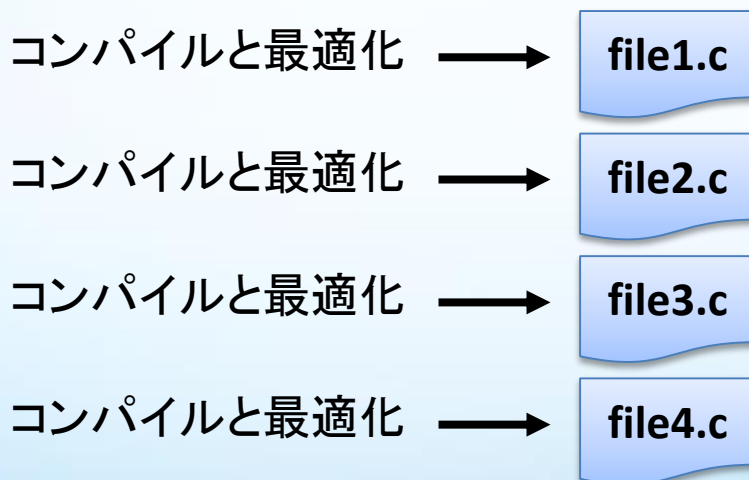
Fortran: <https://software.intel.com/en-us/node/579431>

C/C++: <https://software.intel.com/en-us/node/581881>

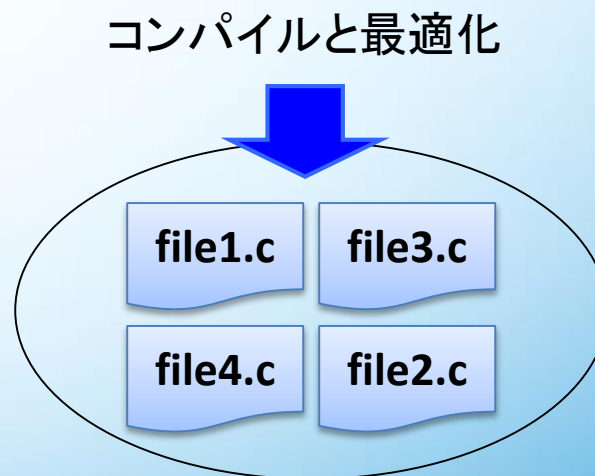
プロシージャ間の最適化: -ipo (Inter Procedural Optimization: IPO)

- 複数ソースファイル間での参照関係を考慮し、プログラム全体の構成を最適化
 - 関数のインライン展開、定数伝播、不要な処理の削除など
 - エイリアス解析などにより、自動ベクトル化を促進

IPOを使用しない場合



IPOを使用した場合



オプションの詳細 (-ipo)

Fortran: <https://software.intel.com/en-us/node/579318>

C/C++: <https://software.intel.com/en-us/node/581756>

自動並列化: -parallel (Parallelizer)

- ループ文を解析し、マルチスレッド化されたコードを生成
 - 安全性: 各反復に依存性があると並列化しない
 - 効率性: 処理量が小さい場合には並列化しない
- レポートオプションにて、並列化の成否を確認可能
 - -par-report[n] n は 0,1,2,3 のいずれか

```
int_sin.cpp(74): (col. 4) remark: ループは並列化されませんでした: 並列依存関係が存在しています。.
```

```
int_sin.cpp(92): (col. 6) remark: ループは並列化されませんでした: 計算量が不足しています。.
```

- 効率性については、しきい値による調整が可能
 - -par-threshold[n] n は 0~100 の間

```
int_sin.c(92): (col. 6) remark: ループが自動並列化されました。
```

オプションの詳細 (-parallel)

Fortran: <https://software.intel.com/en-us/node/579411>

C/C++: <https://software.intel.com/en-us/node/581861>

ガイド付き自動並列化: -guide (Guided Auto Parallelization: GAP)

- コンパイラーが最適化に関するアドバイスを提供
 - 自動ベクトル化 -guide-par
 - 自動並列化 -guide-vec
 - データ構造の改良 -guide-data-trans
- メッセージ例 (自動ベクトル化について)

gap_vec.c(8): remark #30536: (LOOP) 必要に応じて、-fargument-noalias オプションを追加してコンパイラーによる型ベースの一義化解析を向上できます (オプションはコンパイル全体に適用されます)。これにより、行 8 のループがベクトル化され最適化が向上します。[確認] コンパイル全体でこのオプションのセマンティクスに沿っていることを確認してください。[別の方法] ルーチン "matrix_mul_matrix" のすべてのポインター型の引数に "restrict" キーワードを追加することで同様の効果が得られます。これにより、行 8 のループがベクトル化され最適化が向上します。[確認] "restrict" ポインター修飾子のセマンティクスに沿っていることを確認してください。ルーチンにおいて、そのポインターによってアクセスされるすべてのデータは、ほかのポインターからはアクセスできません。

オプションの詳細 (-guide)

Fortran: <https://software.intel.com/en-us/node/579329>

C/C++: <https://software.intel.com/en-us/node/581774>

最適化レポート -qopt-report-phase=*name1,name2,...*

最適化フェーズ *name1*、*name2* 固有の最適化レポートを生成

name 引数には、以下のキーワードを指定可能

- all – すべてのフェーズのすべての最適化レポート (デフォルト)
- loop – ループの入れ子とメモリーの最適化
- vec – 自動ベクトル化と明示的なベクトル・プログラミング
- par – 自動並列化
- openmp – OpenMP* によるスレッド化
- cg – コード生成
- ipo – インライン展開を含むプロシージャラー間の最適化
- pgo – プロファイルに基づく最適化
- offload – インテル® MIC アーキテクチャーやインテル® グラフィックス・テクノロジーへのデータ/実行のオフロード

ベクトル化レポート: -qopt-report-phase=vec

ループがベクトル化されない理由

- Existence of vector dependence
"ベクトルの依存関係が存在する"
- Nonunit stride used
"ユニットストライドではないアクセス"
- Mixed Data Types
"データ型が混合している"
- Condition too Complex
"条件が複雑すぎる"
- Condition may protect exception
"分岐条件の変更が例外を招く"
- Low trip count
"ループ回数が少なすぎる"
- Subscript too complex
"添字が複雑すぎる"
- Unsupported Loop Structure
"サポートされないループ構造"
- Contains unvectorizable statement at line XX
"行 XX にベクトル化できない文が含まれている"
- Not Inner Loop
"内部ループでない"
- Vectorization possible but seems inefficient
"ベクトル化しても非効率"
- Operator unsuited for vectorization
"演算子がベクトル化に適していない"

コマンドとプロセスマネージャーの関係

コマンド	プロセスマネージャー	特長
mpiexec	MPD	パフォーマンスなどの観点から使用は推奨されない。
mpiexec.hydra	Hydra	MPD の欠点を改良したものであり、スケラビリティに優れる。 単一のHydraプロセスが、複数のMPIプロセスを管理する。
mpirun	Hydra (デフォルト) MPD ※環境変数 I_MPI_PROCESS_MANAGERにより 変更可能 I_MPI_PROCESS_MANAGER=hydra I_MPI_PROCESS_MANAGER=mpd	<ul style="list-style-type: none">Hydraの場合、mpiexec.hydra と同様の動作。MPDの場合、mpdboot により MPD が起動され、続いて mpiexec により MPI プログラムが実行される。プログラムが終了すると mpdallexit により MPD を終了する。

インテル® Inspector XE による動的 / 静的エラー検出方法の紹介

インテル® Inspector XE の動的解析を使用するための準備

- ・**-g オプションを指定してコンパイル**

例: `icc -g openmp_app.c`

※-g オプションを指定してデバッグ情報を生成することで、
エラー検出時にソースコードと連携して表示させることが出来る

- ・**GUI または コマンドラインから実行**
次ページより手順記載

GUI からの利用手順

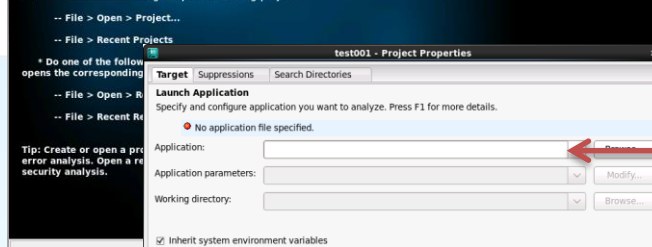
環境変数を設定し、次のコマンドを実行する
inspxe-gui



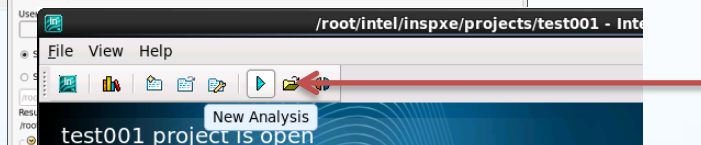
File > New > Project にて、任意のプロジェクト名を指定する



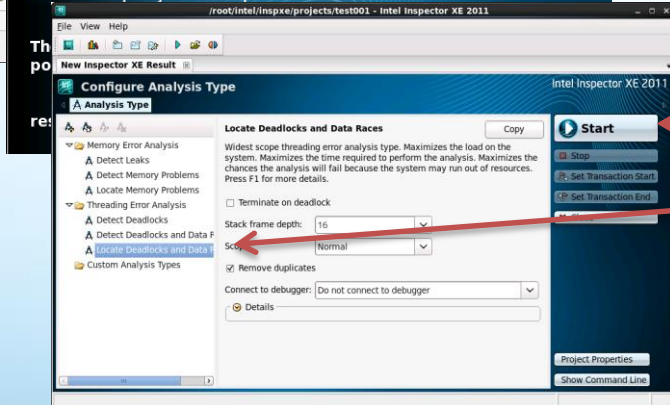
対象アプリケーションを指定する



New Analysis ボタンを押す

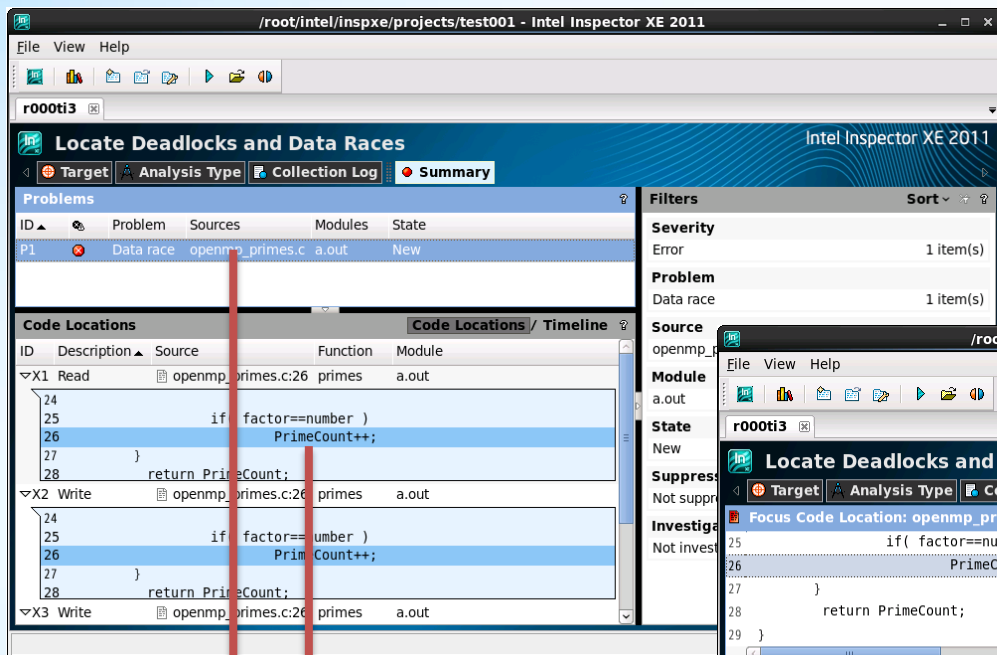


検出タイプを選択し、Start ボタンを押す

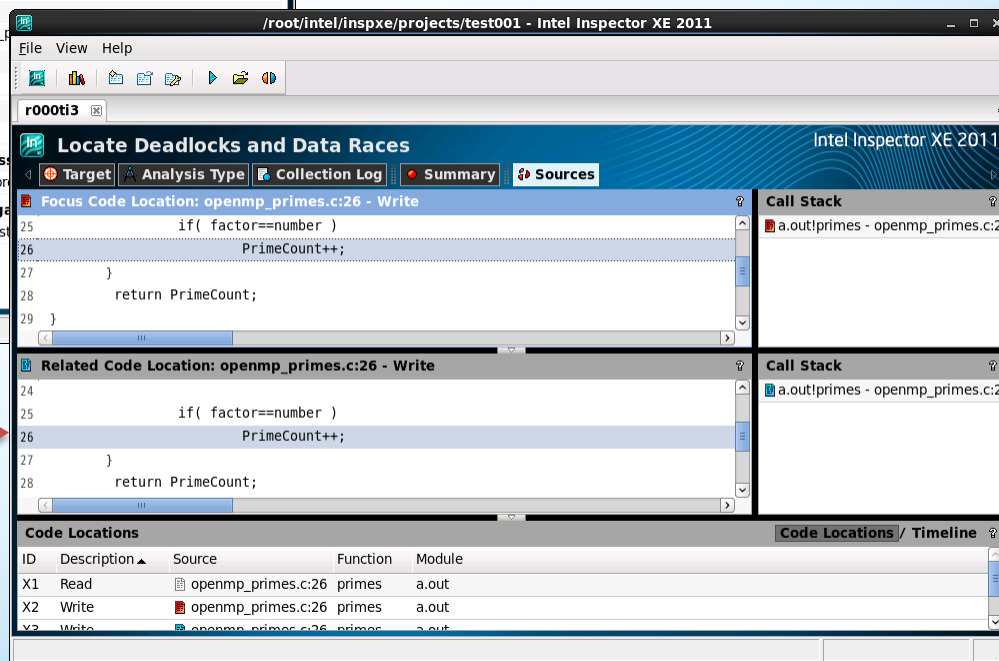


エラー検出画面

エラーの種類と該当箇所が表示される



該当箇所をダブルクリックすると
詳細が表示される



コマンドラインからの動的解析の利用手順

```
inspxe-cl -collect=ti3 -result-dir=hoge ./a.out ②
```

インテル® Inspector XE のコマンドを実行

必要に応じて、検出結果が保存されたフォルダ(上記の場合はフォルダ hoge)をGUI 利用可能なシステムにコピーして、環境変数を設定

```
inspxe-gui
```

インテル® Inspector XE GUI の起動コマンドを実行

File > Open > Result にてリザルトファイル(拡張子 .inspxe)を開く

コマンドラインからの動的解析の利用手順 つづき

コマンド記述方法

```
inspxe-cl -collect=<string> [-action-option] [-global-option] [--]  
<target> [<target options>]
```

<string>

mi1 メモリーリークを検出
mi2 mi1 + 他のメモリー問題を検出
mi3 mi2 + 問題の場所を検出
ti1 デッドロックを検出
ti2 ti1 + データ競合を検出
ti3 ti2 + 問題の場所を検出

[-action-option]

ここでは、結果を保存するディレクトリを指定するオプションを指定する
-result-dir=<string>

[-global-option]

詳細は、"inspxe-cl -help collect" を参照

コマンド例:

```
inspxe-cl -collect=ti3 -result-dir=hoge ./a.out
```

※この場合、./hoge にリザルト ファイルが保存される

インテル® VTune™ Amplifier XE によるパフォーマンス解析

インテル® VTune™ Amplifier XE

- パフォーマンス向上のための解析ツール
ソースコードの修正や、特別なビルドなしに該当箇所を特定

Hotspot

処理に時間を要する箇所を特定

Concurrency

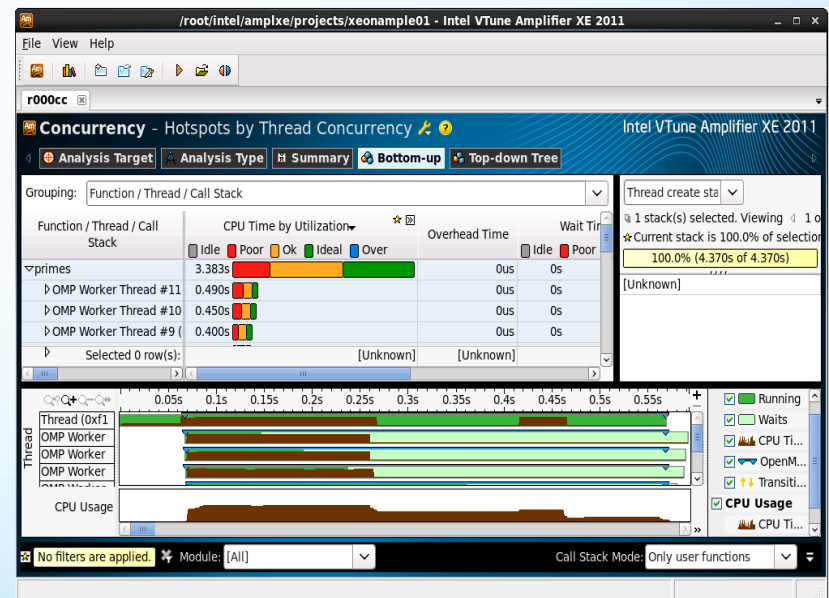
マルチスレッドの並列性を表示

Lock & Wait

スレッド間通信などの情報

Hardware Event

プロセッサで生じたイベントを表示



プリセットされた解析タイプ

- GUI およびコマンドラインから、プリセットされた解析タイプを選択するだけで、主要な情報を取得することができる

<Analysis Type> 例 :

hotspots

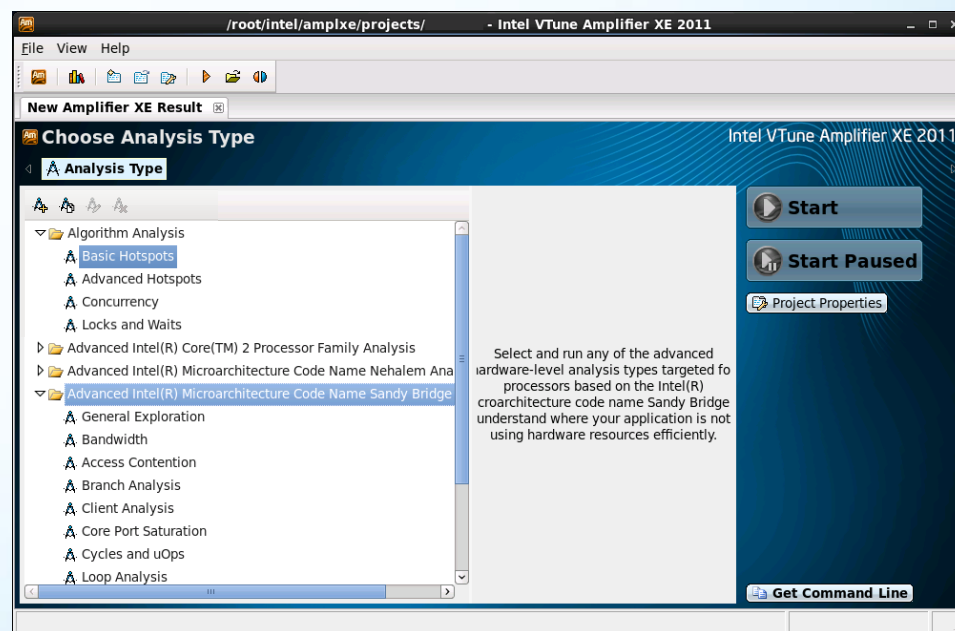
concurrency

locksandwaits

advanced-hotspots

general-exploration

bandwidth

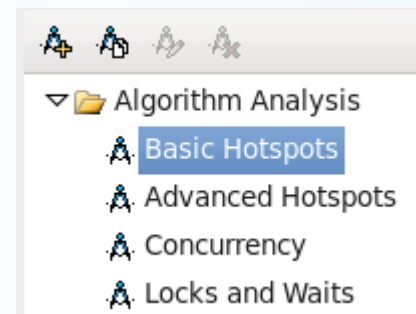


GUI は、初めての利用者でも使用できるように、選択箇所の説明が表示される

解析タイプ

◆ Advanced Hotspots (-collect advanced-hotspots)

- CPU の処理に時間を要している箇所を検出する
- オーバーヘッドが少ない
- 関数コールスタック情報を収集しない
- 論理コア単位の動作状況も表示できる



◆ Basic Hotspots (-collect hotstspots)

- CPU の処理に時間を要している箇所を検出する (オーバーヘッドは約5%)
- 実行中のインストラクション・ポインター(IP)、関数コールスタック情報などを収集する
- 同時実行コア数のヒストグラム表示ができる
- アプリケーションまたはプロセスの検出を行う(システム全体の検出はできない)

◆ Concurrency (-collect concurrency)

- 詳細なスレッド動作の検出を行う
- スレッドの同時利用状況、スレッド間の同期状態などの情報を収集する
- アプリケーションまたはプロセスの検出を行う(システム全体の検出はできない)

◆ Locks and Waits (-collect locksandwaits)

- アイドル状態の検出を行う
- スリープ処理や同期オブジェクト、I/O 処理待ち状態などの情報を収集する
- アプリケーションまたはプロセスの検出を行う(システム全体の検出はできない)

プロセッサーイベントに関する解析

1. 解析タイプ“General Exploration”からサンプリングをスタートする
コマンドラインの場合には (-collect general-exploration)
2. 解析が完了したら、ビューポイントを“(General Exploration)”に設定し、
“Summary”ウィンドウから「Hardware Event-based Metrics」を確認する
 - この評価基準は、インテルアーキテクトによって定められたイベント比率であり、評価基準の閾(しきい)値が定義されている
3. 関数単位の Metrics 内容を確認するために、Bottom-up ボタンをクリック
 - 5%以上のCPU 使用率をもつ関数はホットスポットと認識され、またその基準値を超えている場合は、潜在的な問題と見なされ、ピンクでマークされる
 - 各項目にマウスポインターでフォーカスすると、その項目の内容とイベント比率を求める公式が表示される
 - ピンクでマークされている箇所にマウスポインターでフォーカスすると、問題修正のヒントや設定された基準の閾値が表示される
4. 問題のある関数はダブルクリックしてソース/アセンブリコードで調査する
5. 発見された特定の問題に関する解析タイプを使用して更なる分析を行う

基準値に関する情報（マウスカーソルで表示）

General Exploration - Hardware Issues Intel VTune Amplifier XE 2011

Analysis Target Analysis Type Collection Log Summary Bottom-up grid.cpp

Grouping: Function

Function	Hardware Event Count by Hardware Event Type		CPI Rate	Retire Stalls	LLC Miss	LLC Load Misses Serviced B
	CPU_CLK_UNHALTED.THREAD	INST_RETIRED.ANY				
grid_intersect	13,934,800,000	12,581,000,000	1.108	0.719	0.008	

Sort By Retire Stalls

This metric is defined as a ratio of the number of cycles when no micro-operations are retired to all cycles. In the absence of performance issues, long latency operations, and dependency chains, retire stalls are insignificant. Otherwise, retire stalls result in a performance penalty. On processors based on the Intel(R) microarchitecture code name Nehalem, this metric is based on precise events that do not suffer from significant skid.
 Formula: (Hardware Event Count where Hardware Event Type is UOPS_RETIRED.STALL_CYCLES / Clockticks where Hardware Event Type is CPU_CLK_UNHALTED.THREAD)

基準項目

基準値
の説明
と公式

General Exploration - Hardware Issues Intel VTune Amplifier XE 2011

Analysis Target Analysis Type Collection Log Summary Bottom-up grid.cpp

Grouping: Function

Function	Hardware Event Count by Hardware Event Type		CPI Rate	Retire Stalls	LLC Miss	LLC Load Misses Serviced B
	CPU_CLK_UNHALTED.THREAD	INST_RETIRED.ANY				
grid_intersect	13,934,800,000	12,581,000,000	1.108	0.719	0.008	
sphere_intersect	11,241,400,000	8,921,200,000	1.260	0.563	0.001	
grid_bounds_intersect	1,396,600,000	799,000,000	1.748	0.752	0.002	

A high number of retire stalls is detected. This may result from branch misprediction, instruction starvation, long latency operations, and other issues. Use this metric to find where you have stalled instructions. Once you have located the problem, analyze metrics such as LLC Miss, Execution Stalls, Remote Accesses, Data Sharing, and Contested Accesses, or look for long-latency instructions like divisions and string operations to understand the cause.

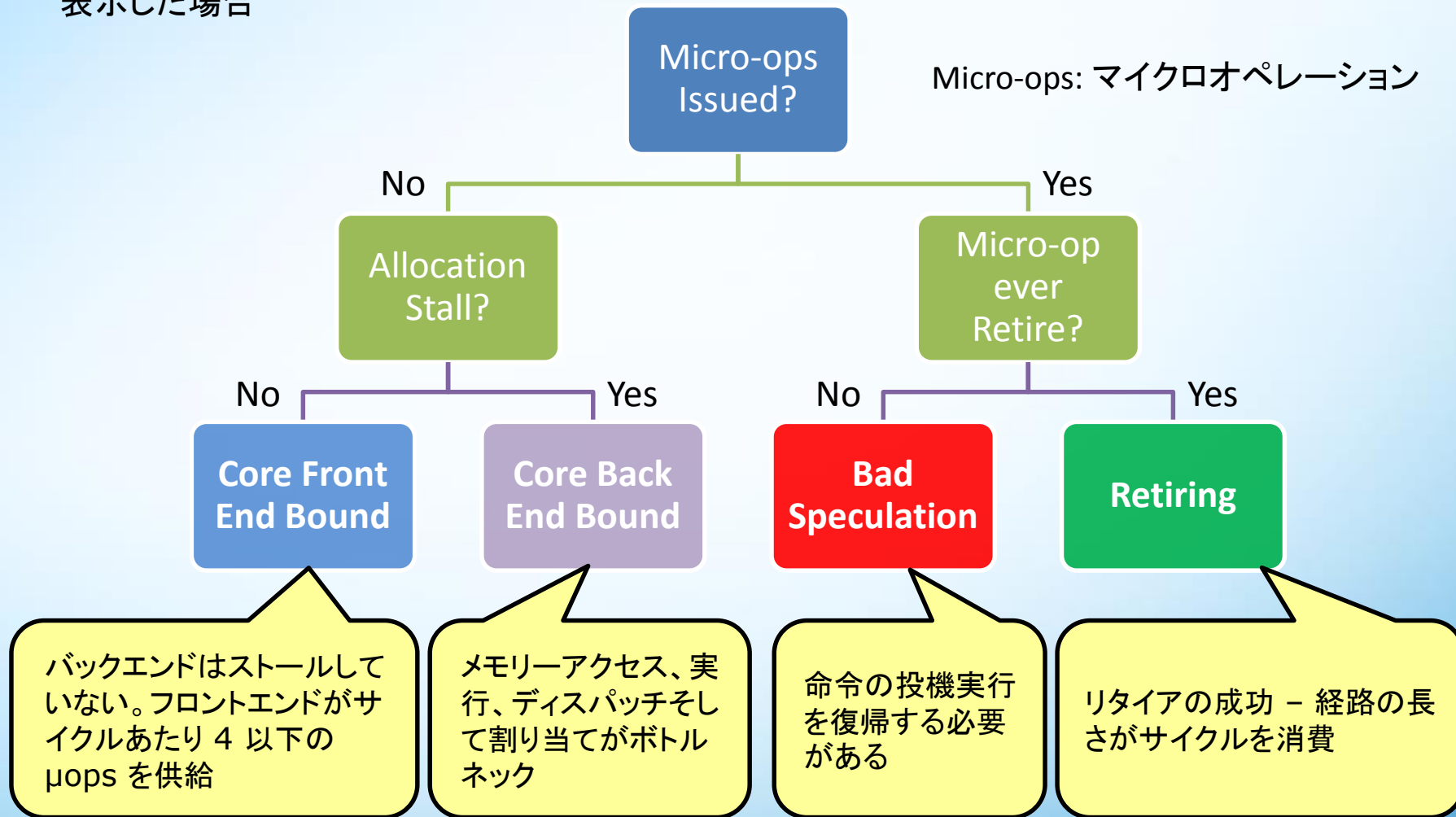
Threshold: (((Hardware Event Count where Hardware Event Type is UOPS_RETIRED.STALL_CYCLES / Clockticks where Hardware Event Type is CPU_CLK_UNHALTED.THREAD) 0.35) * (Clockticks where Hardware Event Type is CPU_CLK_UNHALTED.THREAD / query all Clockticks where Hardware Event Type is CPU_CLK_UNHALTED.THREAD > 0.05))

問題解決
のヒント
と閾値

※ 他Metrics は、Intel VTune Amplifier XE Help ドキュメントから、[Key Concepts] – [Performance Metrics] – [Hardware Event-based Metrics] に記載

プロセッサ イベント サンプリング 解析の考え方(手順)

- ◆ 解析タイプ“General Exploration”でのサンプリング結果を“General Exploration”ビューポイントで表示した場合



インテル® VTune™ Amplifier XE を使用するための準備

・-g オプションを指定してコンパイル

例: `icc -g openmp_app.c -O2`

※-g オプションを指定してデバッグ情報を生成することで、エラー検出時にソースコードと連携して表示させることが出来るようになる。
-g オプションを付けると最適化が無効になるため、-O2 や -O3 -xHOST などの最適化オプションを明示的に指定する。

・GUI または コマンドラインから実行する 次ページより手順記載

GUI からの利用手順

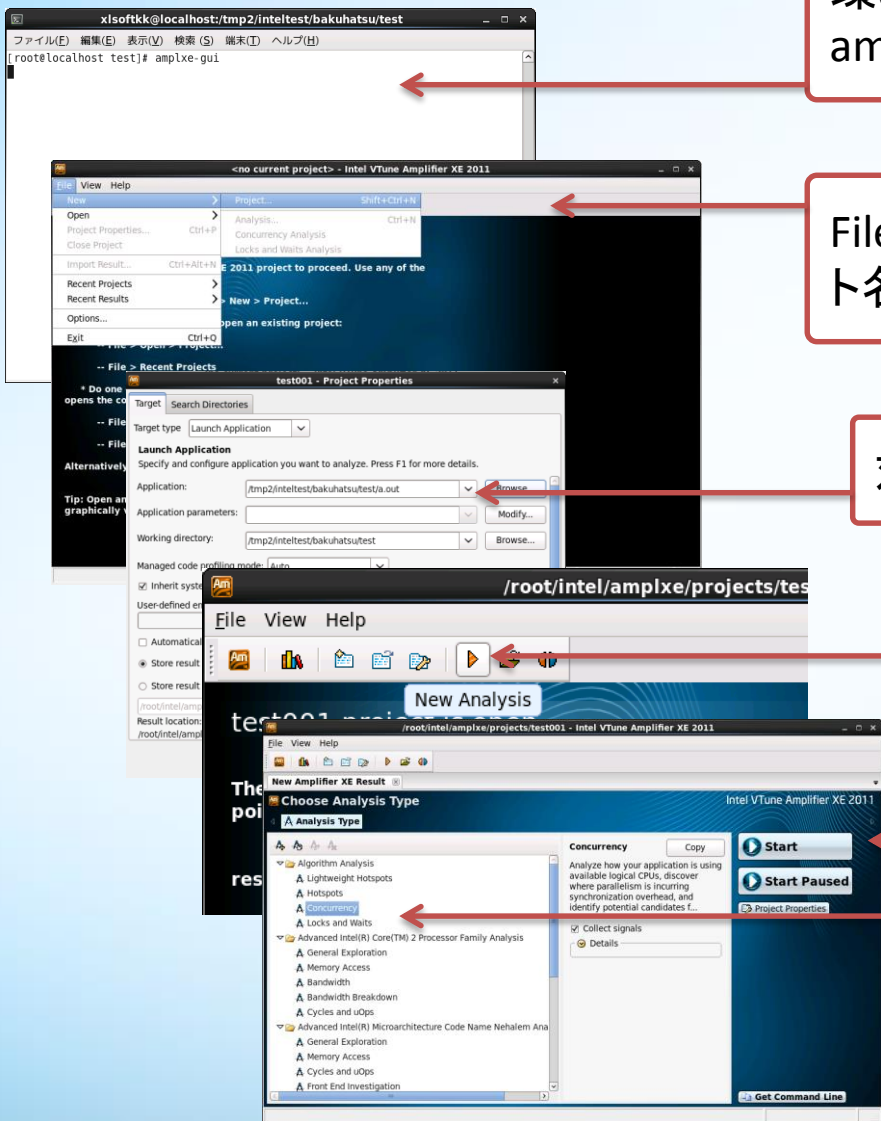
環境変数を設定し、次のコマンドを実行する
amplxe-gui

File > New > Project にて、任意のプロジェクト名を指定する

対象アプリケーションを指定する

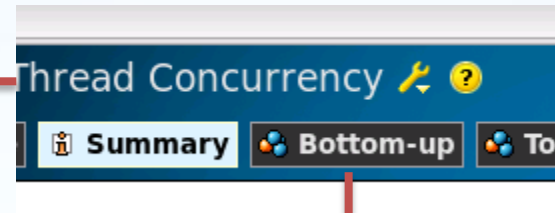
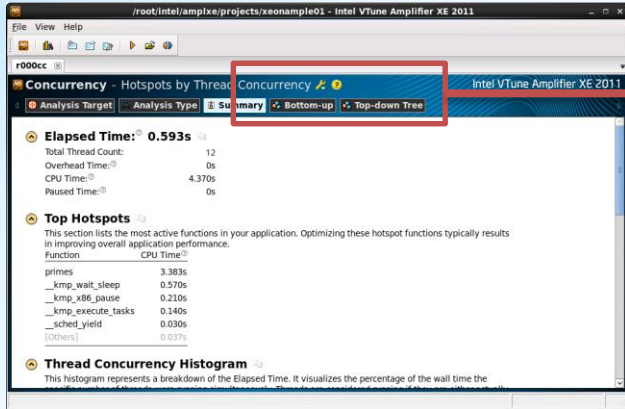
▶ New Analysis ボタンを押す

解析タイプを選択し、Start ボタンを押す※詳細については次ページ

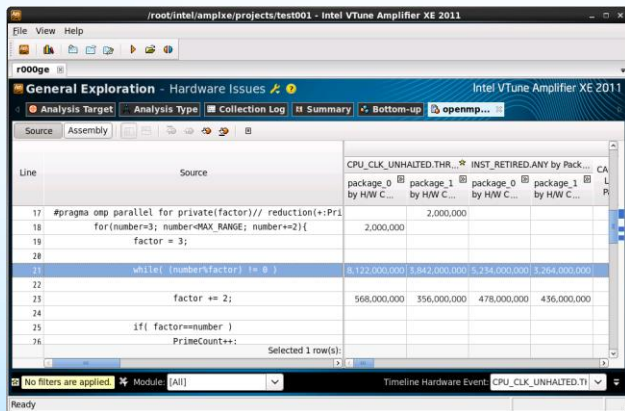
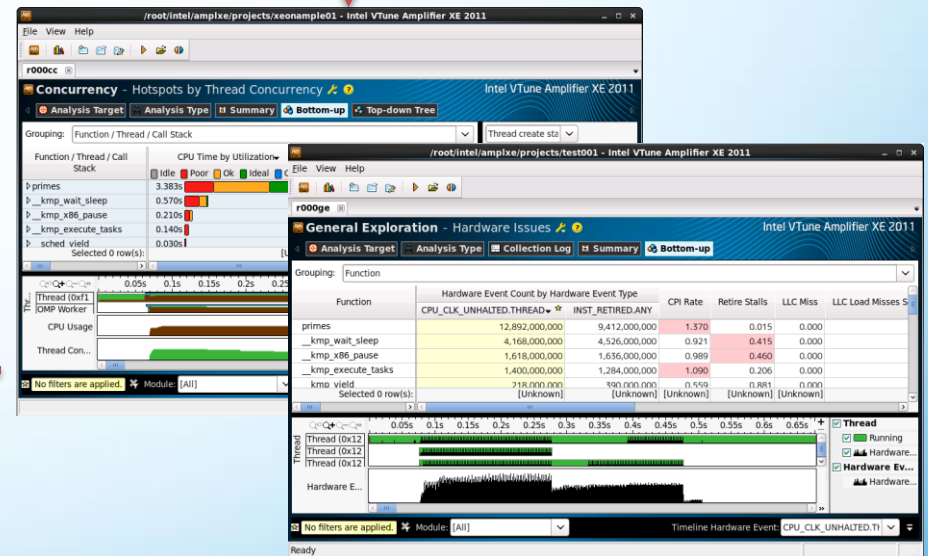


解析結果の操作方法

最初の画面ではサマリーが表示される



Bottom-up をクリック



ソースコード行ごとの情報が表示される

解析タイプにより異なる表示方法の
特定箇所をダブルクリック

インテル® VTune™ Amplifier XE のコマンドライン

※MPI アプリケーションを解析する場合にはコマンドラインからの解析が必須

コマンドからインテル® VTune Amplifier XE を実行する

- 1プロセスのみの解析

```
amplxe-cl -collect <Analysis Type> -- <program>
```

例: `amplxe-cl -collect snb_general-exploration -- ./a.out`

- 複数プロセスの解析 (MPIアプリケーション)

次ページ

インテル® VTune™ Amplifier XE を使用して、複数プロセスの解析を行う場合

- インテル® VTune Amplifier XE は、一部のサンプリング方式では Performance Monitoring Unit (PMU) からデータを取得する

PMUを使用する解析タイプ例

<Analysis Type> =advanced-hotspots, general-exploration, bandwidth

以下のいずれかの方法で可能

```
# mpiexec.hydra -genvall -gtool "amplxe-cl -r <my_result> -collect  
<analysis type>:all=exclusive" -n <n> <my_app> [my_app_options]
```

```
# amplxe-cl -collect <Analysis Type> -- mpiexec.hydra -n 1 -host  
hoge1 ././a.out ...
```

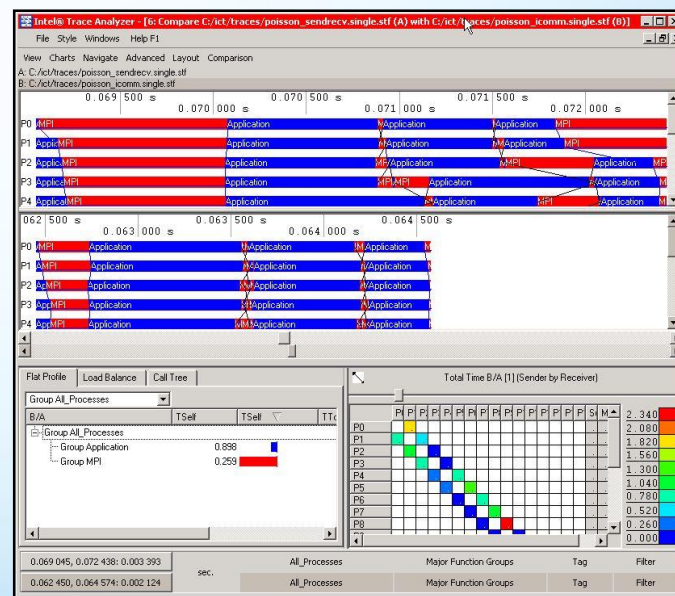
または、a.out : -n 1 -host hoge2

```
# mpiexec.hydra -host host001 -n 1 amplxe-cl -collect <Analysis Type>  
-result_dir <dir name> -- <program>: -host host001 -n 15 <program>:  
-host host 002 -n 1 amplxe-cl -collect <Analysis Type> -result_dir <dir  
name> -- <program>: -host host002 -n 15 ...
```

インテル® Trace Analyzer/ Collector の使用方法

インテル® Trace Analyzer / Collectorの概要

- インテル® Trace Analyzer/Collector
 - MPI アプリケーションの動作やパフォーマンス問題を視覚化
 - プロファイリング統計とロードバランス、通信 hotspot の解析
- 機能
 - イベントベースのアプローチ
 - MPI 関数とユーザーコードを分離
 - 低いオーバーヘッド
 - 複数のプロファイルの比較
 - 強力な集計機能およびフィルタリング機能
 - MPI 正当性検証

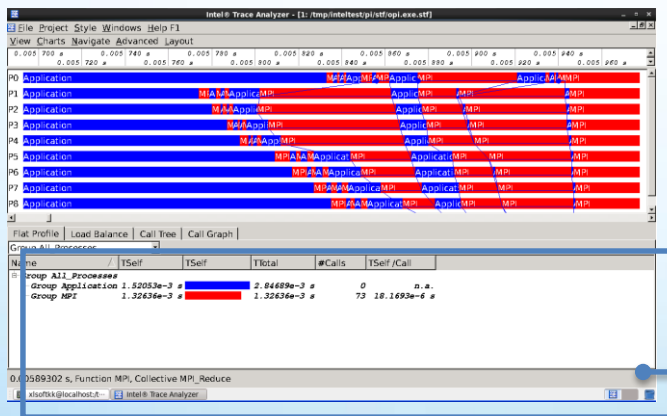


インテル® Trace Analyzer/ Collector の動作テスト1 実行時実装

- コンパイルは通常通り行う
- 実行 (stf ファイルの生成)
 - `mpiexec.hydra -trace -n 8 ./test.exe`
- インテル Trace Analyzer の起動
 - `traceanalyzer text.exe.stf`
 - または、以下コマンドでGUI起動後に stfファイルを開く
 - `traceanalyzer`

インテル® Trace Analyzer/ Collector の動作テスト2

- stf ファイルを開いていない場合: [File]-[Open] より stfファイルを選択
- タイムラインを表示: [Charts]-[Event Timeline]
- タイムライン表示部分をマウสดラッグして拡大
- タイムラインの表示を変更することで Function Profile に表示される内容がフィルタリングされることを確認する

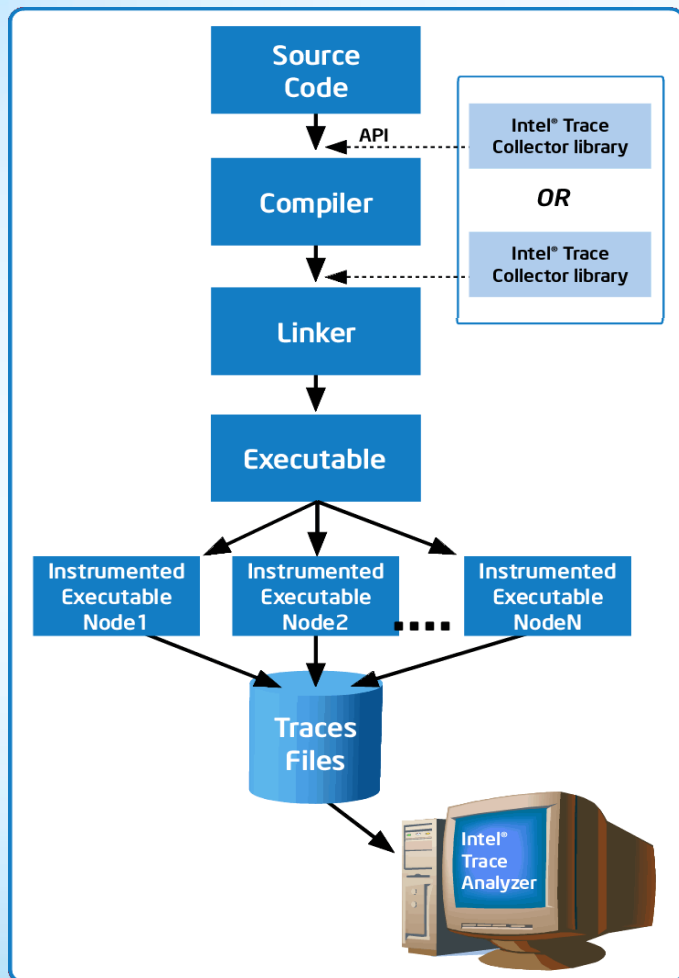


Tips: ショートカットキー

拡大(ズームイン): i
縮小(ズームアウト): o
ズームリセット: r

Function Profile

インテル® Trace Analyzer と インテル® Trace Collector



TC

インテル® Trace Collector

- MPI アプリケーションの解析を主目的とした情報収集ツール
- API、コンパイル時で実装することができ、解析情報は動的に収集される

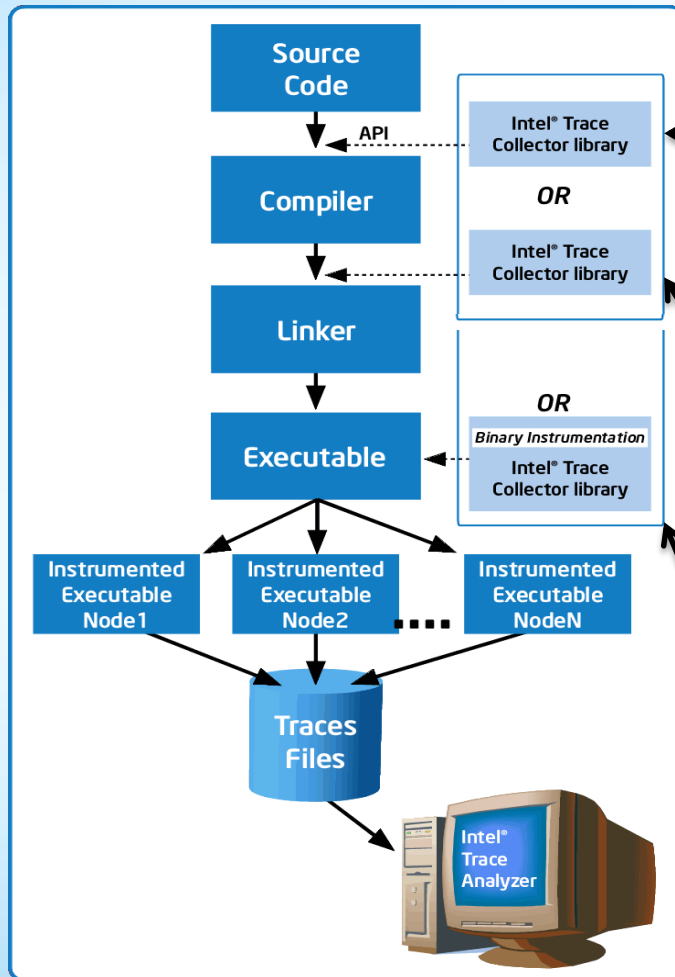
TA

インテル® Trace Analyzer

- インテル® Trace Collector が収集した解析情報を統計的に表示するツール
- 関数別、ユーザーコード/MPI別などにソートし、タイムライン表示やソースコード表示により解析結果を確認することができる

インテル® Trace Collector (TC) の実装

以下のいずれかの方法で実装



・組み込み関数の使用

ソースコードに組み込み関数を挿入してコンパイルすることで、TCが組み込まれたバイナリーが生成される。設定や環境変数により設定を変更することができる。

・コンパイル/リンク時

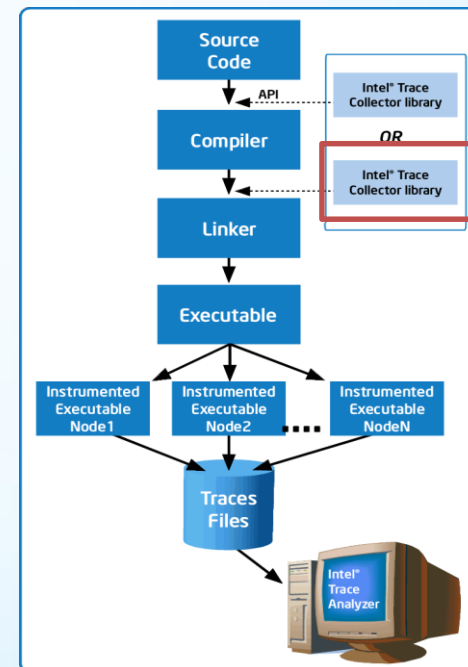
コンパイル時にTCに対応させるオプションを指定することができる。コンパイル時または後から環境変数によって動作を設定することができる。

・実行時

実行時に `-trace` オプションを指定することで解析可能。

コンパイル時のインテル® Trace Collector 実装 1

- コンパイル時に実装する目的
 - コンパイル時実装の場合、実行ファイルを実行するたびにトレースファイルが生成される
 - 入力データを複数パターン試す場合など、コンパイルせずに何度か実行する場合に、実装時間を削減することができる



ソースファイル

コンパイル時に実装

通常どおりの実行で
トレースファイルが生成

コンパイル時のインテル® Trace Collector 実装 2 (バージョン 9.0 以降)

- コンパイル フォーマット

- > mpiicc sample.c -trace -profile=vt

- > mpiicpc sample.cpp -trace -profile=vt

- > mpiifort sample.f -trace -profile=vt

- > mpiicc sample.c -trace -profile=vt

トレースの種類を決めるプロファイル・オプション

-profile=	概要
-IVT	トレースファイルアプリケーション終了時に生成する
-IVTfs	トレースファイルを随時生成する、Fail-safe 方式
-IVTmc	正確性チェックを追加 (エラー検出)

実行は通常通り

- > mpiexec.hydra -n 8 ./a.out

コンパイル時のインテル® Trace Collector 実装 3 (バージョン 8.1 以前の利用方法、9.0 でも利用可能)

- TCライブラリーの選択

> `mpiicc sample.c -L$VT_LIB_DIR -IVT`

-IVT を以下のいずれかに変更することでトレース方法を変更することができる

ライブラリー		概要
-IVTnull	libVTnull.so	擬似ライブラリー。実行時にトレースファイルを生成しない
-IVT	libVT.so	トレースファイルアプリケーション終了時に生成する
-IVTfs	libVTfs.so	トレースファイルを随時生成する、Fail-safe 方式
-IVTmc	libVTmc.so	正確性チェックを追加 (エラー検出)
-IVTcs	libVTcs.so	シングルプロセスのみ生成してトレースファイルを生成

Table 1.4 Usage of Components

http://software.intel.com/sites/products/documentation/hpc/ics/itac/81/ITC_Reference_Guide/hh_goto.htm#1_2_System_Requirements_and_Supported_Features.htm

ソースビューの表示方法

1) `-g` オプションを指定してコンパイルし、実行時に環境変数、`VT_PCTRACE` を設定する。または事前に `export (setenv)` も可

```
>mpicc -g sample.c -o sample.exe -trace -O2
```

```
>mpiexec.hydra -genv VT_PCTRACE=5 -n 8 ./sample.exe
```

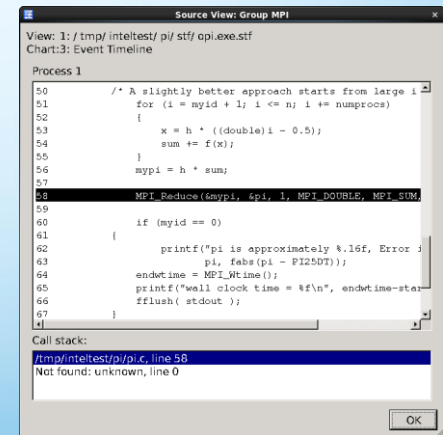
2) TA を起動して、`sample.stf` を開く

3) タイムライン表示、[Charts]-[Event Timeline] の任意の箇所を右クリックして [Details on Function] をクリック

4) [Show Source] ボタンをクリック

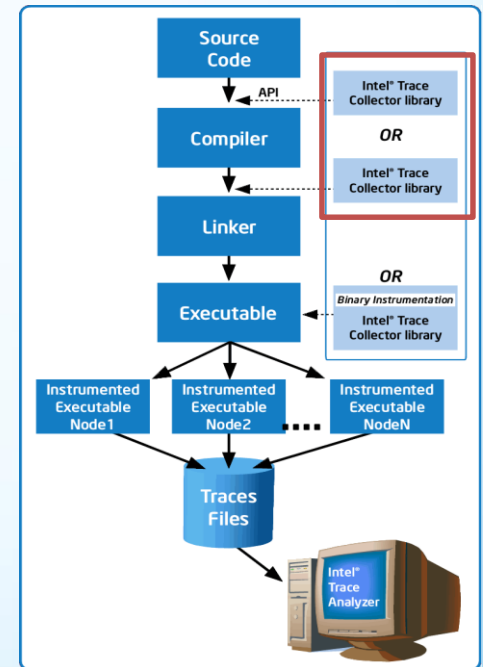
5) 該当箇所のソースコードが表示される

環境変数 `VT_PCTRACE` を設定することで、コールスタックのプログラムカウンター(PC)を記録し、ソースコードの位置と分析結果を紐付ける



組み込み関数 1

- 組み込み関数を使用する目的
 - トレースに必要な時間が長い場合や、大量のMPI関数をコールする際に、データを見やすくし、トレースファイルのサイズを小さくすることができる
 - 任意の箇所に関数を配置することができるため、高い柔軟性がある



ソースファイルに
組み込み関数を追加

コンパイル時に実装

通常どおりの実行で
トレースファイルが生成

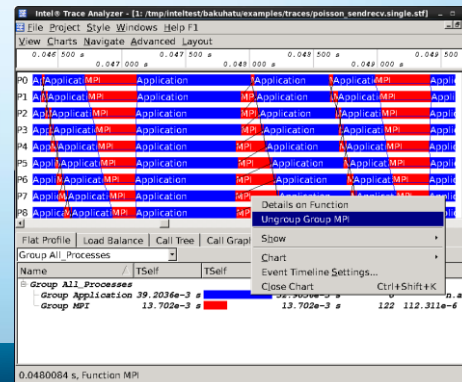
インテル® Trace Analyzer の基本機能

- マニュアル、Tutorial_Analyzing_MPI_Application.htm
を参照
- トレース済みのサンプルファイル
<Install dir>/examples/traces/
poisson_sendrecv.single.stf: MPI_sendrecv を用いたブロック通信
poisson_icommm.single.stf: ブロック無し通信への改良版

インテル® Trace Analyzer で上記ファイルを開く

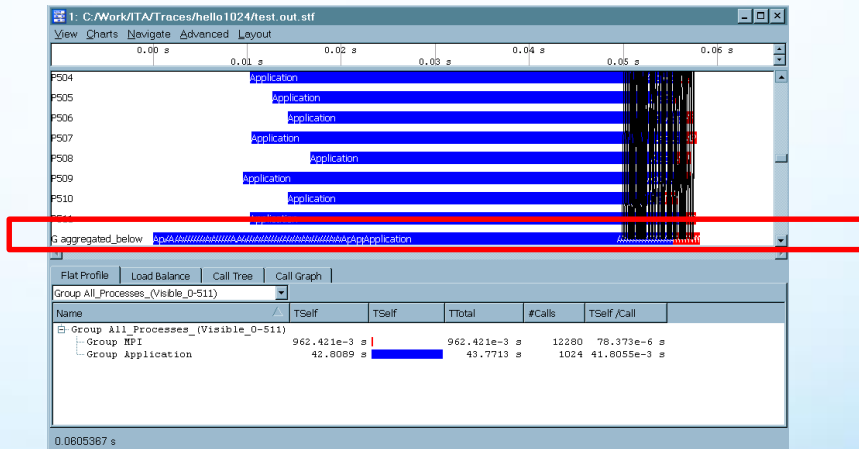
インテル® Trace Analyzer の基本機能

- タイムラインを表示 [Charts]-[Event Timeline]
 - グラフを拡大してアプリケーション (青色部分)とMPI (赤色部分)の関係を
確認する
 - MPIタイムの比率が大きすぎないか
 - 各プロセスが均一か
- 問題の可能性のある箇所の詳細を確認
 - Event Timeline の赤色部分を右クリックし、[MPI Ungroup Group MPI] を選択すると使用されている MPI 関数の詳細が表示される



Advanced Aggregation: データの集約 1

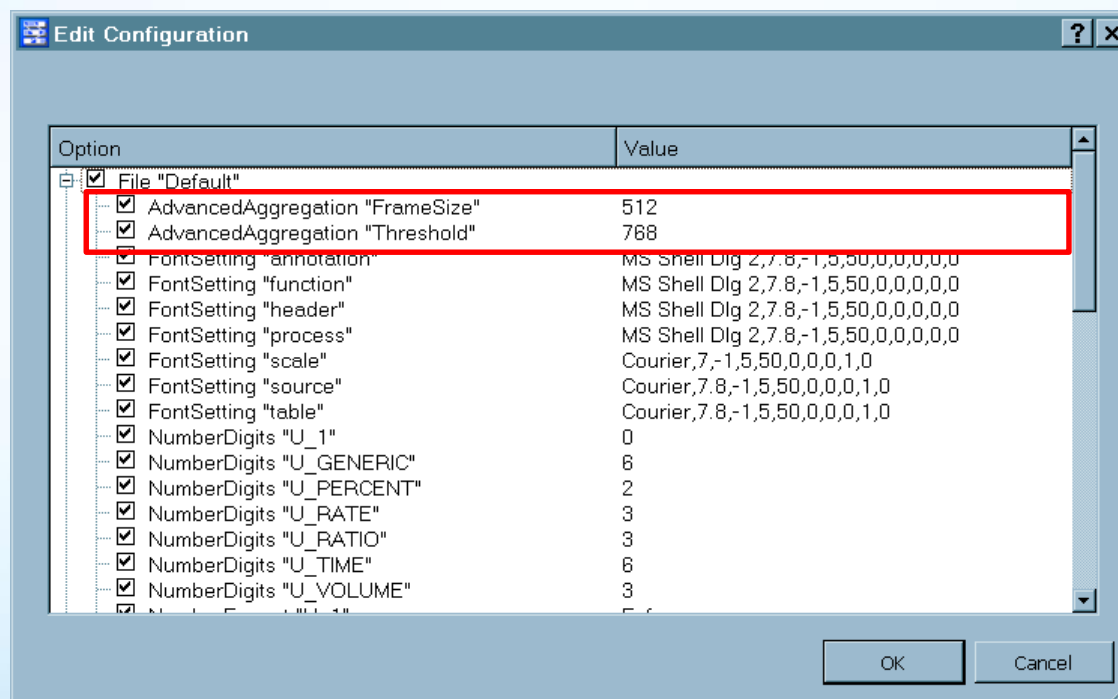
- インテル® Trace Analyzer 8.0.3 より実装された Advanced Aggregation 機能は、大量のデータを特定の事象ごとに集約して表示
 - プロセス数に応じてデータを集約する
 - 1000プロセス以上のデータを扱うことができ、初期値では 768 以上のプロセスは自動的に集約表示される



集約されたプロセスは纏められ、1プロセスのように表示される

Advanced Aggregation: データの集約 2

- 自動集約される閾値は 変更可能
[File]-[Edit Configuration] にて表示されるダイアログで設定



参考: MPI エラーの検出方法

システムB,Cで使用可能
(システムA,D,Eでは使用不可)

MPI エラーの検出方法1

- Message Checking ライブラリーを使用してデッドロック、不正なメモリーアクセス、競合状態、MPIエラー等を自動検出

```
> mpiicc sample.c -trace -profile=vtmc
```

vtmc	Message Checking ライブラリーを指定
------	----------------------------

```
> mpiexec.hydra -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s  
-genv VT_DEADLOCK_WARNING 25s -n 8 ./a.out
```

VT_CHECK_TRACING on	Message Checking のトレーシングをオン
VT_DEADLOCK_TIMEOUT 20s	停止した場合のタイムアウト指定 (20秒)
VT_DEADLOCK_WARNING 25s	停止した場合の警告を表示。デッドロックやロードインバランスの発見に有効 (25秒)

MPI エラーの検出方法2

デッドロックの検出例

➤ サンプルコードを入手

<http://www.shodor.org/refdesk/Resources/Tutorials/BasicMPI/deadlock.c>

➤ libVTmc.so を使用してコンパイルし、デッドロックによる停止に対応できる設定を行い、トレースする (または、後記の実行時の実装などでも可能)

```
> mpiicc -g deadlock.c -o deadlock.exe -L${VT_LIB_DIR} -lVTmc ${VT_ADD_LIBS}
```

```
> mpiexec.hydra -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s -genv VT_DEADLOCK_WARNING 25s -n 2 ./deadlock.exe 0 80000
```

```
[0] ERROR: no progress observed in any process for over 0:20 minutes, aborting application
[0] WARNING: starting emergency trace file writing
```

```
[0] ERROR: GLOBAL:DEADLOCK:HARD: fatal error
[0] ERROR: Application aborted because no progress was observed for over 0:20 minutes,
[0] ERROR: check for real deadlock (cycle of processes waiting for data) or
[0] ERROR: potential deadlock (processes sending data to each other and getting blocked
[0] ERROR: because the MPI might wait for the corresponding receive).
[0] ERROR: [0] no progress observed for over 0:20 minutes, process is currently in MPI call:
[0] ERROR: MPI_Recv(*buf=0x7fff1c4b2474, count=800000, datatype=MPI_INT, source=1,
tag=999, comm=MPI_COMM_WORLD, *status=0x7fff1d0bf874)
[0] ERROR: main (/tmp/inteltest/focus/deadlock.c:49)
[0] ERROR: (/lib64/libc-2.12.so)
[0] ERROR: (/tmp/inteltest/focus/odeadlock.exe)
```

```
[0] ERROR: [1] no progress observed for over 0:20 minutes, process is currently in MPI call:
[0] ERROR: MPI_Recv(*buf=0x7ffa4bd4ce4, count=800000, datatype=MPI_INT, source=0,
tag=999, comm=MPI_COMM_WORLD, *status=0x7ffa4ee20e4)
[0] ERROR: main (/tmp/inteltest/focus/deadlock.c:49)
[0] ERROR: (/lib64/libc-2.12.so)
[0] ERROR: (/tmp/inteltest/focus/odeadlock.exe)
[0] INFO: Writing tracefile odeadlock.exe.stf in /tmp/inteltest/focus/stf
```

```
[0] INFO: GLOBAL:DEADLOCK:HARD: found 1 time (1 error + 0 warnings), 0 reports were suppressed
[0] INFO: Found 1 problem (1 error + 0 warnings), 0 reports were suppressed.
```

```
APPLICATION TERMINATED WITH THE EXIT STRING: Hangup (signal 1)
```

実行時に表示されるメッセージ

MPI エラーの検出方法3

TA による表示

The screenshot shows the Intel Trace Analyzer interface. The main window displays a timeline for two processes, P0 and P1, both labeled 'MPI'. A red bar represents the execution of the MPI function. A black circle on the right end of the P0 bar is highlighted with a blue arrow pointing to the text '右クリックして [Details on Function]'. A secondary window titled 'Details on Function MPI, 1 Issue' is open, showing the following table:

Process	Show Source	Time [s]	Type	Level	Description
P0		20...	GLOBAL:DEADLOCK:HARD fatal error		Application aborted because no progress was observed for over 0:20 minutes check for real deadlock (cycle of processes waiting for data) or potential deadlock (processes sending data to each other and getting blocked because the MPI might wait for the corresponding receive)

タイムライン表示、[Charts]-[Event Timeline] の黒丸を右クリックして [Details on Function] をクリックし、[Issue] タブをクリックすると詳細が表示される。

インテル® Advisor XE による並列化の設計手法

インテル® Advisor XE

- マルチスレッドによる並列化の可能性を検証するツール
実装する前に、パフォーマンスの向上率や起こり得る問題を確認

The screenshot displays the Intel Advisor XE 2016 interface, divided into several panels:

- Summary:** Shows "Maximum Program Gain For All Sites: 6.63x". It includes metrics for serial time (8.9010s), predicted parallel time (1.3435s), and combined site metrics for a "solve" site.
- Site Performance Scalability:** Features a "Scalability of Maximum Site Gain" graph showing performance vs. CPU count. It also includes "Loop Iterations (Tasks) Modeling" with metrics like "Avg. Number of Iterations (Tasks): 14" and "Avg. Iteration (Task) Duration: 0.6309s".
- Where should I add vectorization and/or threading parallelism?:** A table listing function call sites and loops with columns for Self Time, Total Time, Trip Counts, Loop Type, Why No Vectorization?, and Vectorized Loops. It highlights issues like "non-vectorizable loop ins...", "Ineffective peeled/rem...", "High vector register...", and "Assumed dependency...".
- Compiler Diagnostic Details:** Shows source code for "loopstl.cpp:3509 s273_". It highlights a loop that is "Vectorized AVX Loop processing Float32; Float64; Int32 data type(s) having Inserts; Extracts; Masked St" and another loop that is "Scalar Loop. Not vectorized: inner loop was already vectorized".

※ 入門ガイド

<https://software.intel.com/en-us/get-started-with-advisor>

インテル® Advisor XE を使用するための準備

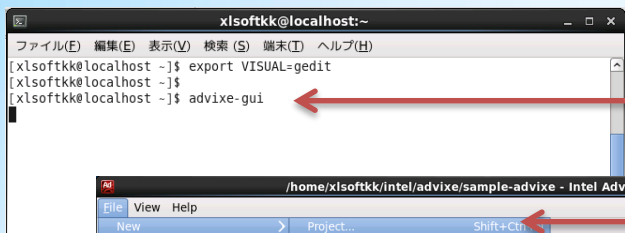
- ・**-g オプションを指定してコンパイル**

例: `icc -g sample_app.c`

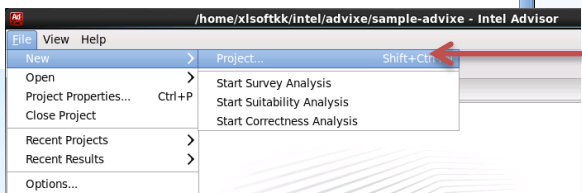
※-g オプションを指定してデバッグ情報を生成することで、
エラー検出時にソースコードと連携して表示させることが出来る

- ・**GUI または コマンドラインから実行**
次ページより手順記載

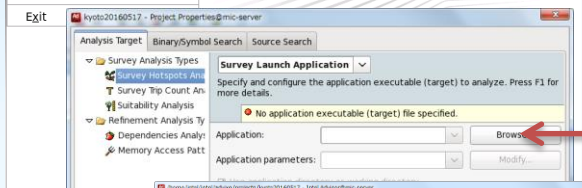
GUI からの利用手順



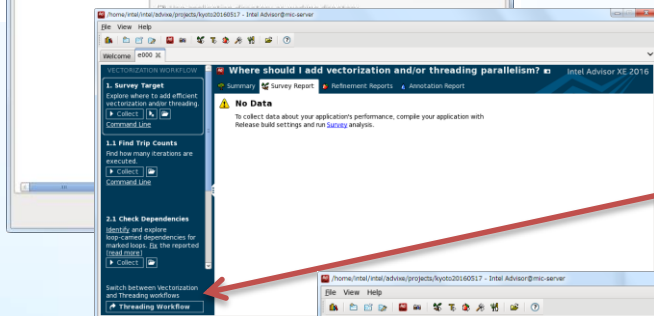
環境変数を設定し、次のコマンドを実行する
advixe-gui



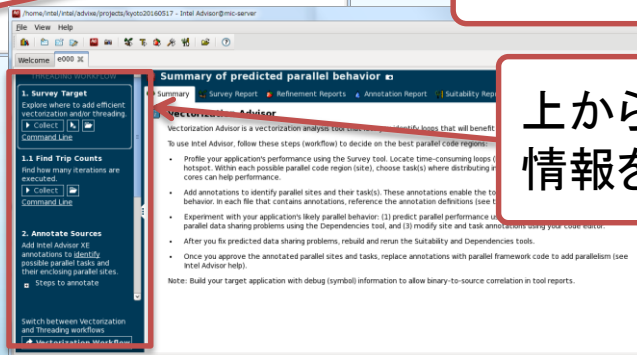
File > New > Project にて、任意のプロジェクト名を指定する



対象アプリケーションを指定する



Threading/Vectorization Workflow
ボタンで得たいアドバイスのもの
を選択する



上から順番にフローをクリックし、
情報を得る

Survey Report 画面

CPU 時間を費やしている関数およびループが表示される

The screenshot displays the Intel Advisor Survey Report interface. The main window shows a table of function call sites and loops with columns for Self Time, Total Time, Trip Counts, and Loop Type. A red box highlights the 'Top Loops' section, which lists loops with their respective times and types. A red arrow points from this section to a detailed view of a loop in the source code.

Top Loops 項目にマークのついたループは、並列化の効果が高いと推測されるもので、ベクトル化の状況も確認できる

該当箇所をダブルクリックすると詳細が表示される

※ Survey Report に関する詳細は製品ヘルプの以下を参照
<https://software.intel.com/en-us/node/608293>

ベクトル化に関するアドバイス

Vectorization Workflow を選択した場合、Survey Report には
ループ毎のベクトル化状況が表示される

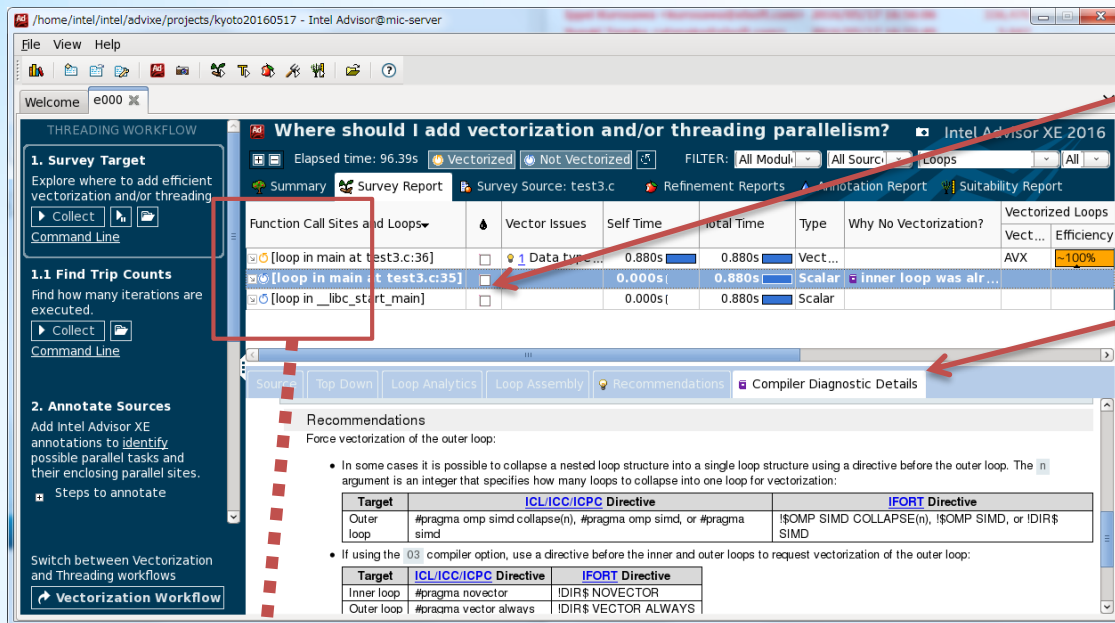
The screenshot shows the Intel Advisor XE 2016 interface. At the top, there's a warning: "Some target modules are compiled with inline debug information disabled". Below that, a table titled "Vectorized Loops" is displayed. The table has columns for Vector ISA, Efficiency, Gain Estimate, VL (Vector Length), Traits, and Data Type. Two rows are visible: one for SSE2 with ~100% efficiency and 2.85x gain, and another with ~71% efficiency and 1.42x gain. A legend on the right explains the efficiency indicators: orange for achieved efficiency higher than reference, yellow for reference efficiency (50%), and light orange for theoretical maximum efficiency (100%).

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

Legend:

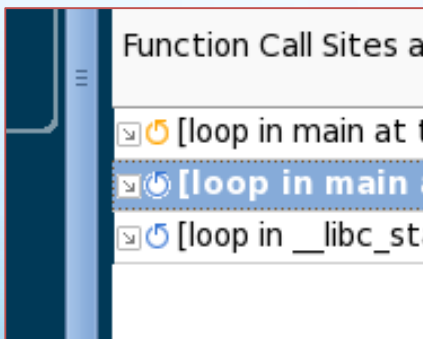
- Orange color** = Achieved vectorization efficiency is higher than reference efficiency for original scalar loop
- Yellow** (50%): Reference Efficiency for original scalar loop
Reference Efficiency = $(1x / \text{Vector Length}) * 100\%$
- Light Orange** (100%): Theoretical Maximum Vectorization Efficiency
Maximum Vectorization Efficiency = $(\text{Theoretical Maximum Gain} / \text{Vector Length}) * 100\%$
Theoretical Maximum Gain = Currently selected Vector Length = 2

ベクトル化できていないループをベクトル化するための情報を得る手順



ベクトル化できていないループの項目をクリックし

Compiler Diagnostic Details タブをクリックすることでベクトル化のアドバイスを得ることができる



ベクトル化できているループ



ベクトル化できていないループ

マルチスレッドに関するアドバイス

- 実装する前に、パフォーマンスの向上率や起こり得る問題を確認

作業手順



1. Survey Target

処理に時間を要する箇所を探し、並列化の候補を決定する



2. Annotate Sources

Annotation (インテル Advisor XE が提供するマクロ処理) をソースコードに追記して、並列処理のモデルを作成する



3. Check Suitability

作成した並列処理のモデルに対して、性能向上率を検証する



4. Check Correctness

マルチスレッドを実装した際に起こり得る問題を確認する



5. Add Parallel Framework

1. ~4. で確認した情報を元に、実際にマルチスレッド化を行う

※ チュートリアル

C/C++言語: https://software.intel.com/en-us/advisorxe_2015_tut_lin_c

Fortran言語: https://software.intel.com/en-us/advisorxe_2015_tut_lin_f

Annotation の追加方法 (C/C++ 言語)

```
#include "advisor-annotate.h"
```

～ 省略 ～

```
ANNOTATE_SITE_BEGIN(site1);
```

```
for (i=0; i<max; i++) {  
    ANNOTATE_TASK_BEGIN(task1);  
    hoge(i);  
    ANNOTATE_TASK_END();  
}
```

```
ANNOTATE_SITE_END();
```

ヘッダファイルをインクルードする

並列化対象となるループ(または関数)
全体を ANNOTATE_SITE_BEGIN(sitename)
～ ANNOTATE_SITE_END() でくる

ループ(または関数)の内部について、並列
に実行させるタスク(処理)を
ANNOTATE_TASK_BEGIN(taskname) ～
ANNOTATE_TASK_END() でくる

コンパイル

例) `icc -g -O2 -c sample.c -I${ADVISOR_XE_2013_DIR}/include`

リンク

例) `icc sample.o -ldl`

※ Annotation に関する詳細は製品ヘルプの以下を参照
<https://software.intel.com/en-us/intel-advisor-2016-user-guide-linux>

Annotation の追加方法 (Fortran 言語)

```
use advisor_annotate
```

～ 省略 ～

```
call annotate_site_begin(site1)
```

```
do i=0, i<max
```

```
  call annotate_task_begin(task1)
```

```
  call hoge(i)
```

```
  call annotate_task_end()
```

```
enddo
```

```
call annotate_site_end()
```

モジュールの利用宣言を追加する

並列化対象となるループ(または関数)
全体を call annotate_site_begin(sitename)
～ call annotate_site_end() でくる

ループ(または関数)の内部について、並列
に実行させるタスク(処理)を
call annotate_task_begin(taskname) ～ call
annotate_task_end() でくる

コンパイル

例) ifort -g -O2 -c sample.f90 -I\${ADVISOR_XE_2013_DIR}/include/intel64

リンク

例) ifort sample.o -L\${ADVISOR_XE_2013_DIR}/lib64 -ladvisor

※ Annotation に関する詳細は製品ヘルプの以下を参照

<https://software.intel.com/en-us/intel-advisor-2016-user-guide-linux>

Suitability Report 画面

Annotation に基づいた、並列実行時の性能向上率が表示される

Maximum Program Gain For All Sites:
1.33x

Target CPU Count: 8 Threading Model: Intel TBB

Annotation Label	Source Location	Maximum Site G...	Maximum Total G...	Average Instance ...	Total Time
site1	primes.c:15	1.33x	1.33x	6.7896s	6.7896s

Scalability of Maximum Site Gain

Changes I will make to this site to improve performance

Type of Change	Benefit if Checked	Loss if Unchecked	Recommended
<input type="checkbox"/> Reduce Site Overhead			No
<input type="checkbox"/> Reduce Task Overhead	5.27x		No
<input type="checkbox"/> Reduce Lock Overhead			No
<input type="checkbox"/> Reduce Lock Contention			No
<input type="checkbox"/> Enable Task Chunking	6.63x		Yes

Annotation	Annotation Label	Source Location	Number of Instances	Maximum Instance Time	Average Instance Time	Minimum Instance Time	Total Time
Selected Site	site1	primes.c:15	1	6.7896s	6.7896s	6.7896s	6.7896s
Task	task1	primes.c:18	249,999	< 0.0001s	< 0.0001s	< 0.0001s	6.7824s

コア数とスレッド化方法を選択すると、その条件での性能向上率を確認できる

実装時に考慮すべき点と、適用した場合に改善される性能効率が示される

※ Suitability Report に関する詳細は製品ヘルプの以下を参照
<https://software.intel.com/en-us/node/608272>

Correctness Report 画面

エラーの種類と該当箇所が表示される

問題の例: "Data Communication"
マルチスレッドを実装した際に、データ競合を起こすメモリアクセスを指摘

The screenshot shows the Intel Advisor interface with the Correctness Report tab selected. The 'Problems and Messages' table lists two issues:

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	site1	primes.c	a.out	✓ Not a problem
P2	Data communication	site1	primes.c	a.out	New

A red arrow points from the 'Data communication' row to a detailed view of the issue. The detailed view shows the source code for 'primes.c:23 - Read' and 'primes.c:23 - Write'. The 'Data communication: Code Locations' table at the bottom of the detailed view is as follows:

ID	Description	Source	Function	Module	State
X2	Parallel site	primes.c:14	primes	a.out	New
X3	Read	primes.c:23	primes	a.out	New
X4	Write	primes.c:23	primes	a.out	New

The 'Relationship Diagram' at the bottom right shows a flow from 'Write' (primes.c:23) to 'Read' (primes.c:23).

該当箇所をダブルクリックすると
詳細が表示される

※ Correctness Report に関する詳細は製品ヘルプの以下を参照
<https://software.intel.com/en-us/node/608104>

コマンドラインからの利用手順

```
advixe-cl -collect=survey -project-dir=hoge ./a.out ②
```

インテル® Advisor XE のコマンドを実行

必要に応じて、検出結果が保存されたフォルダ(上記の場合はフォルダ hoge)を GUI 利用可能なシステムにコピーして、環境変数を設定

```
advixe-gui
```

インテル® Advisor XE GUI の起動コマンドを実行

File > Open > Result にてリザルトファイル(拡張子 .advixe)を開く

※ コマンドラインからの利用手順に関する詳細は製品ヘルプの以下を参照
<https://software.intel.com/en-us/node/608400>

コマンドラインからの利用手順 つづき

コマンド記述方法

`advixe-cl -collect=<string> [-option] [--] <target> [<target options>]`

<string>

survey	処理時間の計測
suitability	性能向上率を推定
correctness	並列時の問題を検出

[-option]

詳細は、“advixe-cl -help” の出力を参照

例：結果を保存するディレクトリを指定するオプション
`-project-dir=<PATH>`

コマンド例：

```
advixe-cl -collect=survey -project-dir=hoge ./a.out
```

※この場合、./hoge にリザルト ファイルが保存される

※ コマンドラインオプションの詳細は製品ヘルプの以下を参照
<https://software.intel.com/en-us/node/608410>

最適化に関する注意事項

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 SSSE3 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

Optimization Notice

最適化に関する注意事項

インテル コンパイラーは、互換マイクロプロセッサ向けには、インテル製マイクロプロセッサ向けと同等レベルの最適化が行われない可能性があります。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 SSE2)、インテル ストリーミング SIMD 拡張命令 3 (インテル® SSE3)、ストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。インテルでは、インテル製ではないマイクロプロセッサに対して、最適化の提供、機能、効果を保証していません。本製品のマイクロプロセッサ固有の最適化は、インテル製マイクロプロセッサでの使用を目的としています。インテル® マイクロアーキテクチャーに非固有の特定の最適化は、インテル製マイクロプロセッサ向けに予約されています。この注意事項の適用対象である特定の命令セットの詳細は、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。

改訂 #20110804