インテル® コンパイラー入門

LLVM を採用した新しいインテル® コンパイラー

エクセルソフト株式会社

2026年7月15日

はじめに

- この資料は2026/07/02 時点の情報をベースとしています
- パッケージバージョン 2025.3 を対象としています
- 資料中のコンパイラー・オプションは Linux 向けで表記しています
- 利用手順は大阪大学様の Web ページにて
  公開されているスパコンマニュアルをあわせてご参考ください
- 資料中では記載の都合により
  icc/icpc、ifort を「以前のインテル® コンパイラー」
  icx/icpx、ifx を「新しいインテル® コンパイラー」と
  一部表記しています
- 一部の画像は旧バージョンの製品を利用しています

ご紹介内容

- インテル® oneAPI ツールキット全体のご案内
- LLVM を採用した新しいインテル® コンパイラーの概要
- 以前より提供されていたインテル® コンパイラーとの動作の違いや新機能
- 新しいインテル コンパイラーへの切り替えにあたって
- インテル VTune™ プロファイラーを利用して性能情報を取得する手順

インテル® oneAPI ベース & HPC ツールキット

コンパイラーや実行環境

- インテル® Fortran コンパイラー
- インテル® oneAPI DPC++/C++ コンパイラー
- インテル® ディストリビューションの Python*

ツール/ライブラリー

- インテル® MPI ライブラリー
- インテル® oneAPI スレッディング・ビルディング・ブロック (インテル® oneTBB)
- インテル® oneAPI DPC++ ライブラリー (インテル® oneDPL)
- インテル® oneAPI コレクティブ・コミュニケーション・ライブラリー (インテル® oneCCL)
- インテル® oneAPI マス・カーネル・ライブラリー (インテル® oneMKL)
- インテル® oneAPI ディープ・ニューラル・ネットワーク・ライブラリー (インテル® oneDNN)
- インテル® oneAPI データ・アナリティクス・ライブラリー (インテル® oneDAL)
- インテル® インテグレーテッド・パフォーマンス・プリミティブ (インテル® IPP)
- インテル® DPC++ 互換性ツール
- インテル® oneAPI DPC++/C++ コンパイラー用 FPGA サポートパッケージ

解析/デバッグツール

- インテル® VTune™ プロファイラー
- インテル® Advisor
- インテル® ディストリビューションの GDB

■ インテル® **HPC** ツールキットにのみ含まれるもの

インテル® oneAPI ベース & HPC ツールキット

- インテル® Parallel Studio XE の後継製品
  - ✓ コンパイラーやライブラリーなどのコンポーネントを引き続き提供
- インテル® プロセッサー上で動作するアプリケーションの最適化を支援
  - ✓ インテル® コンパイラー
    後述します
  - ✓ インテル® oneMKL
    インテル® MKL の後継
    BLAS、LAPACK、FFT などの計算処理を実装した MPI 対応の数値演算ライブラリー
    プロセッサーの ISA を認識して最適な関数を実装するディスパッチ機能
  - ✓ インテル® MPI ライブラリー
    オープンソースの MPICH の仕様を実装した MPI ライブラリー
    時間を消費しやすい集合操作などをチューニングするためのユーティリティを提供

インテル® コンパイラー

- インテル製のハードウェア向けに最適化された
  アプリケーションを開発するための機能を提供するコンパイラー
  - ✓ プロセッサーメーカー製のコンパイラー
  - ✓ 最新のインテル® アーキテクチャ向けに継続してアップデート
- oneAPI ツールキットがリリースしたタイミングにて
  新しいコンパイラーをリリース

4種類のインテル® コンパイラー

- インテル® C++ コンパイラー クラシック
  インテル® Fortran コンパイラー クラシック
  - ✓ インテル® Parallel Studio XE より提供が続けられていたコンパイラー
- インテル® oneAPI DPC++/C++ コンパイラー
  インテル® Fortran コンパイラー
  - ✓ インテル® oneAPI ツールキットにて提供が開始されたコンパイラー

インテル® Fortran コンパイラー・クラシック (ifort)
インテル® C++ コンパイラー・クラシック (icc/icpc)

- 以前より提供されてきたインテルによるコンパイラー実装
  - ✓ コンパイラーのコマンド名から ifort および icc とも呼ばれます
- パッケージバージョン 2025.0.0 にて ifort の提供が終了
- icc/icpc は 2024.0 リリースより提供を終了
- 最終バージョン - ()内はパッケージバージョン

  icc/icpc　2021.10 (2023.2)
  ifort　　2021.13 (2024.2)
- パッケージバージョン 2025 以降は新しいコンパイラーのみ提供

インテル® oneAPI DPC++/C++ コンパイラー icx/icpx

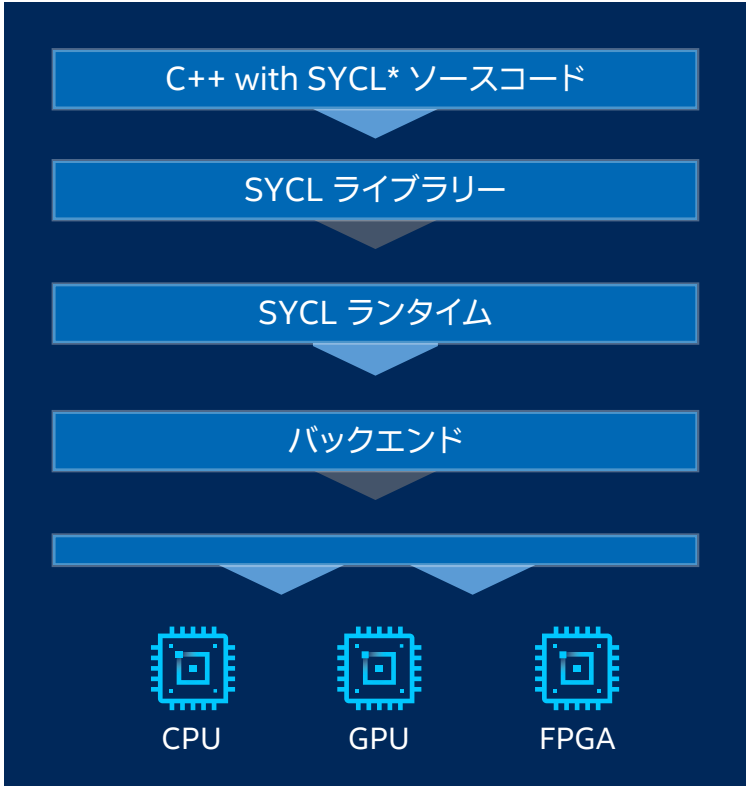
- Clang フロントエンドおよび LLVM バックエンドを採用した新しいコンパイラー
  - ✓ インテルによる独自のオプティマイザーとコード生成を実装
    引き続きインテルコンパイラーの特長を継承
  - ✓ Clang のコンパイルオプションを認識しつつ、インテルのコンパイラー実装と同じオプションを利用可能
- SYCL* 2020 サポート
  - ✓ バージョン 2024 にて SYCL* 2020 に準拠した初めてのコンパイラーとして認定

補足: SYCL

- クロノス・グループが定義する
  並列プログラミングモデル
  - ✓ SYCL Overview (khronos.org)
- CPU、GPU、FPGA などの異なる
  デバイスを抽象化
  - ✓ 単一の C++ コードから
    各デバイス上で計算処理を実行可能

    テンプレートとラムダ関数を使用した
    C++ ベースの汎用プログラミング

    OpenCL や CUDA、HIP などの
    広範なバックエンド API へ対応



icx/icpx の言語、OpenMP* サポート

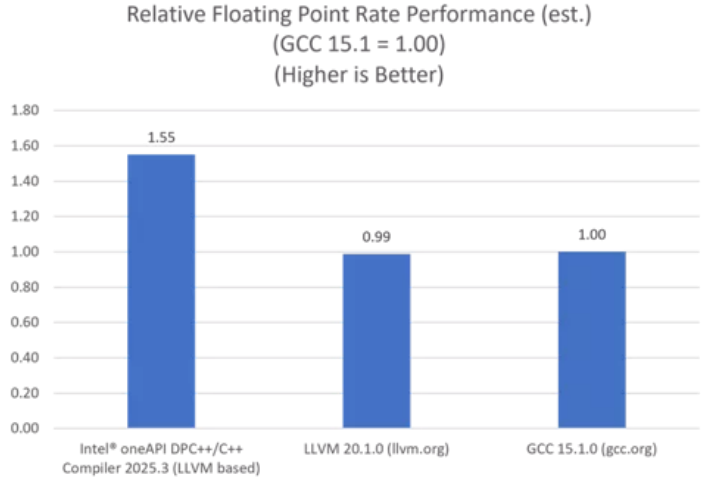
- C/C++ 言語規格への対応は Clang フロントエンドに準拠
  - ✓ インテル® コンパイラーが採用している Clang バージョンはリリースノートにて公開
    コンパイラーバージョン 2025.0 は Clang 19
    Intel® oneAPI DPC++/C++ Compiler Release Notes (intel.com)
- OpenMP*
  - ✓ 4.5 フルサポート、5.0、5.1、5.2、6.0 TR11
    OpenMP 仕様およびインテル OpenMP 拡張の対応
    OpenMP* Features supported (intel.com)

Standard	Intel Feature Support
C++23 standard (ISO/IEC 14882:2023)	Partial support
C++20 standard (ISO/IEC 14882:2020)	Partial support
C++17 standard (ISO/IEC 14882:2017)	Full support
C++14 standard (ISO/IEC 14882:2014)	Full support
C++11 standard (ISO/IEC 14882:2011)	Full support
C++98 standard (ISO/IEC 14882:1998)	Full support (except for `export`)
C23 standard (ISO/IEC 9899:2018)	Partial support
C17 standard (ISO/IEC 9899:2018)	Partial support
C11 standard (ISO/IEC 9899:2011)	Partial support
C99 standard (ISO/IEC 9899:1999)	Full support

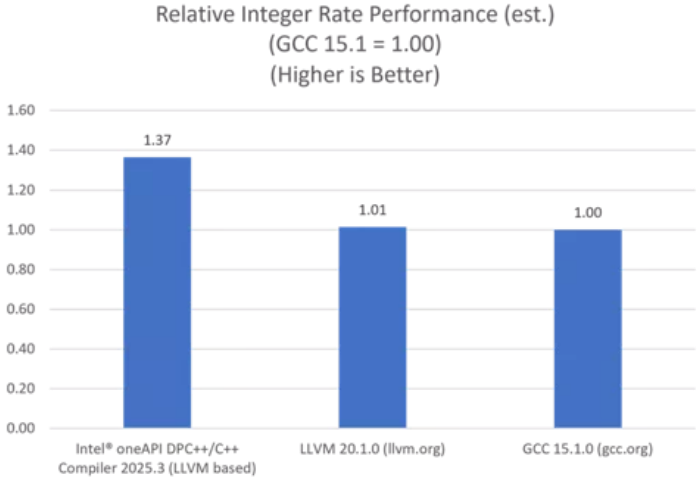
出典: Standards Conformance (intel.com)

Intel® Compilers Boost C++ Application Performance on Linux*

Performance advantage relative to other compilers on Intel® Xeon® 6980P Processor



Estimated: internal measurement of the geometric mean of the C/C++ workloads from the SPECrate* 2017 Floating Point suite (base tune)



Estimated: internal measurement of the geometric mean of the C/C++ workloads from the SPECrate* 2017 Integer suite (base tune)

Testing Date: Performance results are **based on testing by Intel as of October 10, 2025** and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: Intel® Xeon® 6980P CPU @ 2.0GHz, 2 sockets, Hyper Thread on, Turbo on, 64G x24 DDR5 8800 (1DPC). Software: Intel® oneAPI DPC++/C++ Compiler for applications running on Intel® 64, Version 2025.3.0 Build 20251010; GCC 15.1.0; LLVM 20.1.0. Ubuntu 24.04 LTS, 6.8.0-55-generic. SPECint®_rate_base_2017 compiler switches: Intel® oneAPI DPC++/C++ Compiler: -xgraniterapids -O3 -ffast-math -flto -mfpmath=sse -funroll-loops -qopt-mem-layout-trans=4. GCC: -march=native -mfpmath=sse -Ofast -funroll-loops -flto. LLVM: -march=native -mfpmath=sse -Ofast -funroll-loops –flto. qkmalloc used for Intel compiler. jemalloc 5.0.1 used for gcc and llvm. SPECfp®_rate_base_2017 compiler switches: Intel® oneAPI DPC++/C++ Compiler: -xgraniterapids -mprefer-vector-width=512 –O3 -ffast-math -flto -mfpmath=sse -funroll-loops -qopt-mem-layout-trans=4. GCC: -march=native -mfpmath=sse -Ofast -funroll-loops -flto. LLVM: -march=native -mfpmath=sse -Ofast -funroll-loops -flto. jemalloc 5.0.1 used for Intel compilers, GCC and LLVM.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. More information on the SPEC benchmarks can be found at: http://www.spec.org

ベンチマーク：出典

インテル® Fortran コンパイラー (ifx)

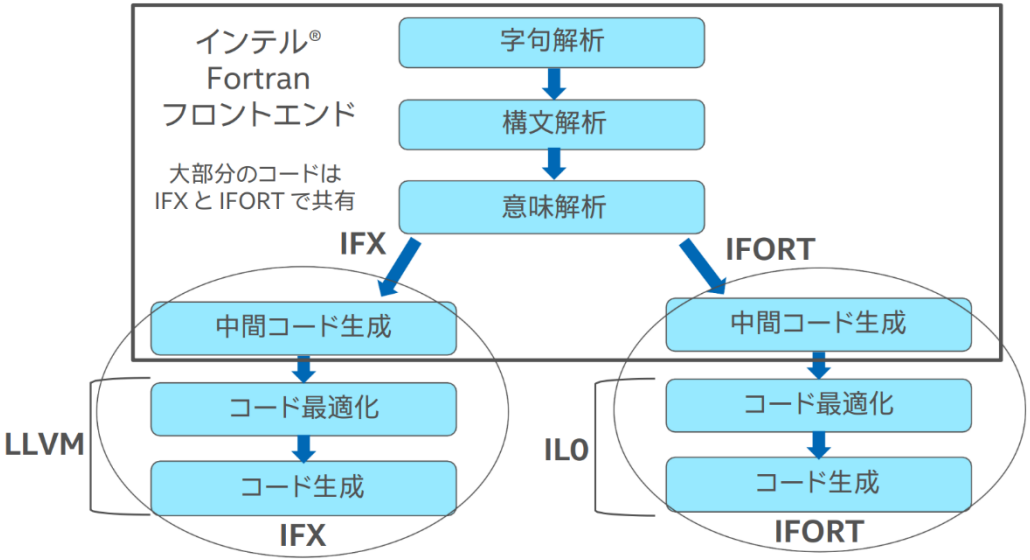
- ifort で採用されていたフロントエンドおよび
  ランタイムを引き継ぎつつ LLVM を採用したコンパイラー
  - ✓ 2018 年頃より開発計画がスタート、2021年にベータ版が公開
  - ✓ パッケージバージョン 2023 にて、計画されていた ifort に
    相当する機能の実装が完了して正式にリリース

    余談: 当時の開発マネージャーから ifx がリリースされるまでの動きについて
    ブログ記事が掲載されています

    The Next Chapter for the Intel® Fortran Compiler (intel.com)

ifx のコンパイラーデザイン

- ifort で利用していた主要機能をそのまま使える一方で
  最適化やコード生成における実装が異なる

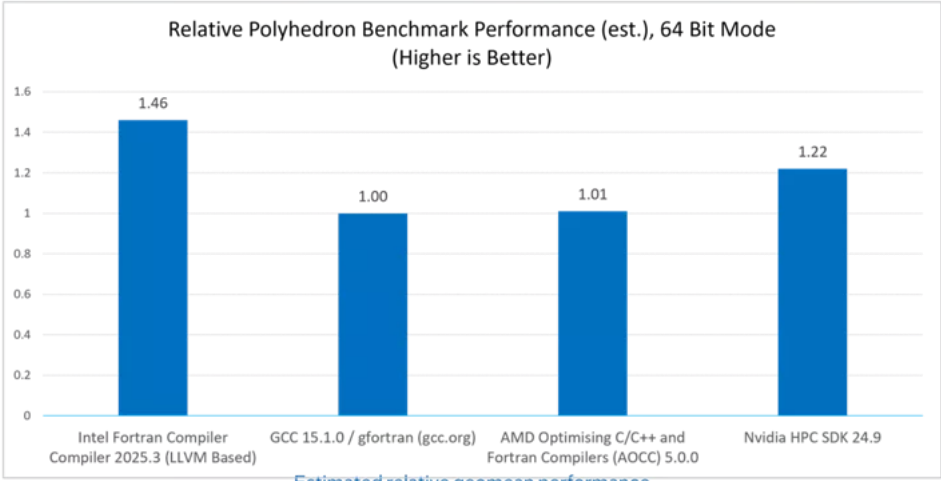


ifx の言語、OpenMP* サポート

- Conformance, Compatibility, and Fortran Features (intel.com)
- Fortran 言語規格
  - ✓ 77, 90, 95, 2003, 2008, 2018 サポート
    および 2023 の一部を実装
- OpenMP* 規格
  - ✓ OpenMP* 4.5 以前、5.0、5.1
    + 5.2の大部分 と 6.0 一部
  - ✓ インテル® GPU 向け TARGET 構文のサポート

Intel® Fortran Compiler Boosts Application Performance on Linux*

Performance Advantage Measured by Polyhedron* Fortran Benchmark on Intel® Xeon® 6980P Processor



Testing Date: Performance results are **based on testing by Intel as of October 10, 2025** and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: Intel® Xeon® 6980P CPU @ 2.0GHz, 2 sockets, Hyper Thread on, Turbo on, 64G x24 DDR5 8800 (1DPC). Software: Intel® Fortran Compiler for applications running on Intel® 64, Version 2025.3.0 Build 20251010; GCC 15.1.0 / gfortran; AMD* Optimizing C/C++ Compiler 5.0.0 / flang - AMD clang version 17.0.6 (CLANG: AOCC_5.0.0-Build#1377 2024_09_24); NVIDIA* HPC SDK 24.9 / nvfortran; Ubuntu 24.04 LTS, 6.8.0-55-generic. Compiler switches: Intel® Fortran Compiler: -C -xCORE-AVX512 -flto -nostandard-realloc-lhs. GCC / gfortran: -Ofast -mfpmath=sse -flto -march=skylake-avx512 -funroll-loops. AMD* Optimizing C/C++ Compiler / flang: -O3 -ffast-math -march=znver5 -fveclib=AMDLIBM -flto -mllvm -unroll-aggressive -mllv unroll-threshold=500. NVIDIA* HPC SDK / nvfortran: -fast -Mipa=fast,inline -Mallocatable=03 -Mfprelaxed -Mstack_arrays.

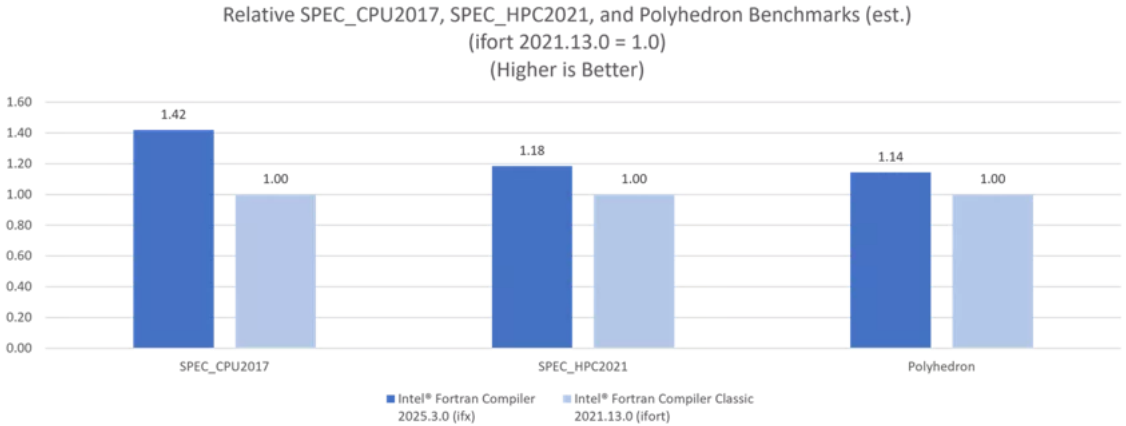
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary.

ベンチマーク：出典

Intel® Fortran Compiler (ifx) & Intel® Fortran Compiler Classic (ifort) Performance on Linux*

Performance Advantage of ifx to ifort on Intel® Xeon® 6980P Processor



Estimated: internal measurement of the geometric mean of each of the SPEC_CPU2017, SPEC_HPC2021, and Polyhedron Benchmarks suites (base tune)

Testing Date: Performance results are **based on testing by Intel as of October 10, 2025** and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: Intel® Xeon® 6980P CPU @ 2.0GHz, 2 sockets, Hyper Thread on, Turbo on, 24x 64G MRDIMM DDR5-8800 (1DPC). Software: Intel® Fortran Compiler for applications running on Intel® 64, Version 2025.3.0 Build 20251009; Intel® Fortran Compiler Classic for applications running on Intel® 64, Version 2021.13.0 Build 20240602_000000. Ubuntu 24.04 LTS, 6.8.0-55-generic. Compiler switches: Intel® Fortran Compiler for SPEC_CPU2017: -xgraniterapids -mprefer-vector-width=512 -Ofast -ffast-math -flto -mfpmath=sse -funroll-loops -qopt-mem-layout-trans=4 -ljemalloc. Intel® Fortran Compiler Classic for SPEC_CPU2017: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -ffinite-math-only -qopt-multiple-gather-scatter-by-shuffles -qopt-mem-layout-trans=4 -ljemalloc. jemalloc 5.0.1 used for Intel compilers. Intel® Fortran Compiler for SPEC_HPC2021: -Ofast -xgraniterapids -mprefer-vector-width=512 -fiopenmp -flto -qopt-multiple-gather-scatter-by-shuffles -funroll-loops -qopt-streaming-stores=always -nostandard-realloc-lhs -align array64byte. Intel® Fortran Compiler Classic for SPEC_HPC2021: -Ofast -xCORE-AVX512 -qopt-zmm-usage=high -qopt-multiple-gather-scatter-by-shuffles -ansi-alias -qopenmp -ipo. Intel® Fortran Compiler for Polyhedron: -Ofast -xCORE-AVX512 -flto -nostandard-realloc-lhs -qopt-dynamic-align -Qoption,f,"-target-feature -fast-vector-fsqrt" -auto. Intel® Fortran Compiler Classic for Polyhedron: -O3 -xCORE-AVX512 -ipo -nostandard-realloc-lhs -no-prec-div.

ベンチマーク：出典

コマンド名

- それぞれのコンパイラーに対応したコンパイラー ドライバーを提供
  - ✓ 例: ソースファイル名 app をコンパイル、リンクして run.out を生成

ドライバー	製品
icx/icpx	インテル® oneAPI DPC++/C++ コンパイラー
ifx	インテル® Fortran コンパイラー
icc/icpc	インテル® C++ コンパイラー・クラシック
ifort	インテル® Fortran コンパイラー・クラシック

```
> icpx ./app.cpp -o run.out
```

```
> icx ./app.c -o run.out
```

```
> ifx ./app.f90 -o run.out
```

コンパイラー・オプション

- icc/icpc および ifort にて実装されていた
  コンパイラー・オプションの多くを引き続き実装
- icx/icpx は Clang コンパイラーのコンパイラー・オプションを認識
  - ✓ Clang のコンパイラー・オプションの詳細は Clang のドキュメントを参照
    Clang Compiler User's Manual (clang.llvm.org)

コンパイラー・オプション

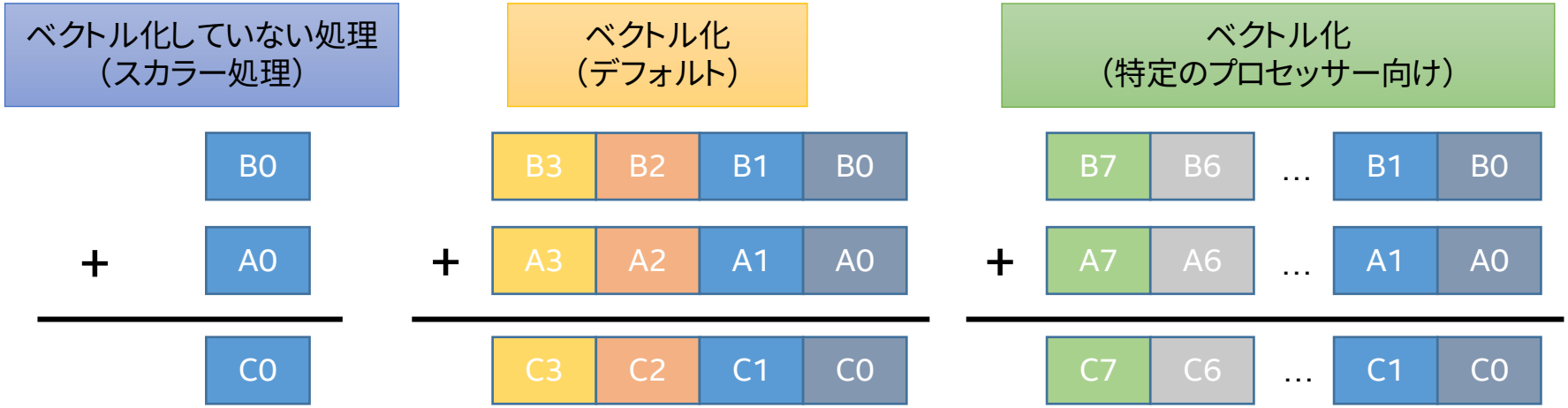
- ベクトル化を含むプロセッサー固有の最適化
- IPO (LTO)
- PGO
- 最適化レポート
- LLVM サニタイザー
- OpenMP* サポート

ベクトル化

```
for (int i = 0; i < count; i++)
{
    C[i] = A[i] + B[i];
}
```

- ループ処理や関数を対象にプロセッサーでサポートされる SIMD 操作を用いるコードを生成して実行効率を高めます

SIMD : Single Instruction Multiple Data



SIMD 拡張命令のサポート

- インテル® ストリーミング SIMD 拡張
  - ✓ インテル® SSE2,インテル® SSE3,インテル® SSE4
    128bit 長のレジスタ (xmm)を使用した SIMD 操作
- インテル® Advanced Vector Extensions
  - ✓ インテル® AVX およびインテル® AVX2
    256bit 長のレジスタ (ymm)を利用した SSE の拡張命令セット
- インテル® Advanced Vector Extensions 512
  - ✓ インテル® AVX-512
    512bit 長のレジスタ (zmm)を利用して AVX の2倍、
    SSE の4倍の要素を処理できるように拡張された命令セット

[SSE] float 4 要素を一度に演算
```
addps %xmm1, %xmm2
```

[AVX] float 8 要素を一度に演算
```
vaddps %ymm1, %ymm2, %ymm3
```

[AVX512] float 16 要素を一度に演算
```
vaddps %zmm1, %zmm2, %zmm3
```

自動ベクトル化機能

- データの依存関係の解析、および変換にかかるコストモデルなどの評価をもとにスカラー処理を自動的にベクトル化
  - ✓ -O2 以上のコンパイラー・オプションの指定が必要
- インテル® コンパイラーは LLVM コミュニティーで実装されているベクトライザーとは別にインテル製CPU、GPU 向けに新しく設計
  - ✓ LLVM コミュニティーにおける VPlan ベクトライザーをインテル アーキテクチャ向けに独自に拡張

プロセッサー固有の最適化

- SIMD 操作を含む特定のインテル® プロセッサーで利用可能なコードの生成、および最適化を指示
- -x<target>
  - ✓ インテル® プロセッサー向けの専用コードを生成
  - ✓ <target>キーワードはマイクロアーキテクチャーもしくはSIMD 拡張命令セットを指定

    実行ファイルは非インテル® プロセッサーや、<target> に対応した命令をサポートしていないインテル® プロセッサーで動作しません

意味	<target> 抜粋
プロセッサーのマイクロアーキテクチャー	ARROWLAKE ARROWLAKE-S DIAMONDRAPIDS EMERALDRAPIDS GRANITERAPIDS LUNARLAKE PANTHERLAKE
SIMD 拡張命令セット	CORE-AVX512 CORE-AVX2

プロセッサー固有の最適化

- -xhost
  - ✓ コンパイラーを実行しているシステムに搭載された
    プロセッサーをターゲットにした専用コードを生成
- -ax<target>,<target>…
  - ✓ インテル® プロセッサー向け専用コードに加えて
    x86 プロセッサー向け汎用コードを生成
    専用コードを実行できない環境では汎用コードを実行
  - ✓ 複数の <target> を指定することで、それぞれの専用コードを生成

プロセッサー固有の最適化

- -qopt-zmm-usage=low / high
  - ✓ zmm レジスターの利用に用いるヒューリスティックの制御
    low は明確な性能向上があると評価した場合に zmm レジスターを利用したコードを生成
  - ✓ -x, -ax の指定により変化しますが、多くの場合 low が設定
  - ✓ 積極的に zmm レジスターの使用したコードを生成するには high を指定
- AVX512 のコード生成が有効ではない場合は無視されます

プロセッサー固有の最適化オプション 続き

- -qopt-zmm-usage=low と -qopt-zmm-usage=high の asm 出力
  - ✓ 例: 16 要素の倍精度型の FMA 演算
    ymm レジスター 4本を利用 → zmm レジスター 2本の利用

```
vmovupd    b(%rsi,%r8,8), %ymm1
vmovupd    32+b(%rsi,%r8,8), %ymm2
vmovupd    64+b(%rsi,%r8,8), %ymm3
vmovupd    96+b(%rsi,%r8,8), %ymm4
vfmadd213pd c(%rax,%r8,8), %ymm0, %ymm1
vfmadd213pd 32+c(%rax,%r8,8), %ymm0, %ymm2
vfmadd213pd 64+c(%rax,%r8,8), %ymm0, %ymm3
vfmadd213pd 96+c(%rax,%r8,8), %ymm0, %ymm4
vmovupd    %ymm1, c(%rax,%r8,8)
vmovupd    %ymm2, 32+c(%rax,%r8,8)
vmovupd    %ymm3, 64+c(%rax,%r8,8)
vmovupd    %ymm4, 96+c(%rax,%r8,8)
```

```
vmovupd    b(%r11,%rbx,8), %zmm0
vmovupd    64+b(%r11,%rbx,8), %zmm1
vfmadd213pd c(%rax,%rbx,8), %zmm2, %zmm0
vfmadd213pd 64+c(%rax,%rbx,8), %zmm2, %zmm1
vmovupd    %zmm0, c(%rax,%rbx,8)
vmovupd    %zmm1, 64+c(%rax,%rbx,8)
```

プロセッサー固有の最適化

- SQUID システムのフロントエンドからコンパイルするにあたり
  -xhost もしくは -xicelake-server を利用ください
  - ✓ 汎用CPUノード群はフロントエンドと同じプロセッサーを
    採用しているため -xhost でも対応いただけます

    Intel Xeon Platinum 8368
- -x, -ax, -march はいずれか1つのみ指定ください
  - ✓ 複数指定した場合、コマンドの最後に記述されたものが適用されます
- 指定のない場合、SSE および SSE2 を利用した汎用コードを生成します
  (-O2 以上が必須)

プロセッサー固有の最適化 オプション例

- 自動ベクトル化が有効な最適化されたコードを生成

```
> icx -O2 ./source.c -o run.out
```

- 主要なAVX-512 命令を含む専用コードを生成

```
> icx -O2 -xCORE-AVX512  ./source.c -o run.out
```

- 主要な AVX-512 命令を含む専用コード +
  AVX2 命令を含む専用コード+
  汎用コードを生成

```
> icx -O2 -axCORE-AVX512,CORE-AVX2  ./source.c -o run.out
```

プロセッサー固有の最適化 オプション例

- ホストプロセッサー向けに最適化された専用コードの生成 + zmm レジスターの積極的な利用を指示

```
> icx -O2 -xhost -qopt-zmm-usage=high ./source.c -o run.out
```

- 第3世代 インテル® Xeon® スケーラブル・プロセッサー向け専用コードの生成 + zmm レジスターの積極的な利用を指示

```
> icx -O2 -xicelake-server -qopt-zmm-usage=high  ./source.c -o run.out
```

- MPI プログラムにおけるオプションの指定

```
> mpiicx -O2 -xicelake-server -qopt-zmm-usage=high ¥
source.c -o run.out
```

補足: 専用コードの実行

- -x<target> オプションにより生成された専用コードは
  <target> がサポートする命令を実行できないシステムでは動作しません
  - ✓ 複数システムで実行される場合は -ax<target> を利用ください

```
> icx -O2 -xgraniterapids -qopt-zmm-usage=high ./source.c -o run.out
> ./run.out

Please verify that both the operating system and the processor support
Intel(R) AVX512F, AVX512DQ, AVX512CD, AVX512BW, AVX512VL, AVX512VBMI,
AVX512_VPOPCNTDQ, AVX512_BITALG, AVX512_VBMI2, AVX512_VNNI and
AVX512_FP16 instructions.
```

リンク時最適化 - LTO
プロシージャー間の最適化 - IPO

- コンパイラーがプログラムを構成する複数のソースファイル間の関係を把握して追加の最適化を実施
  - ✓ ソースファイル間で利用されるグローバル変数の操作や関数/サブルーチンの呼び出し関係を識別できるようになりより多くの最適化が適用されやすい
    - 関数 / サブルーチンのインライン展開
    - ループ展開や定数伝播
    - 不要と判定された処理や変数、条件分岐の削除 など…

LTO (IPO) 機能

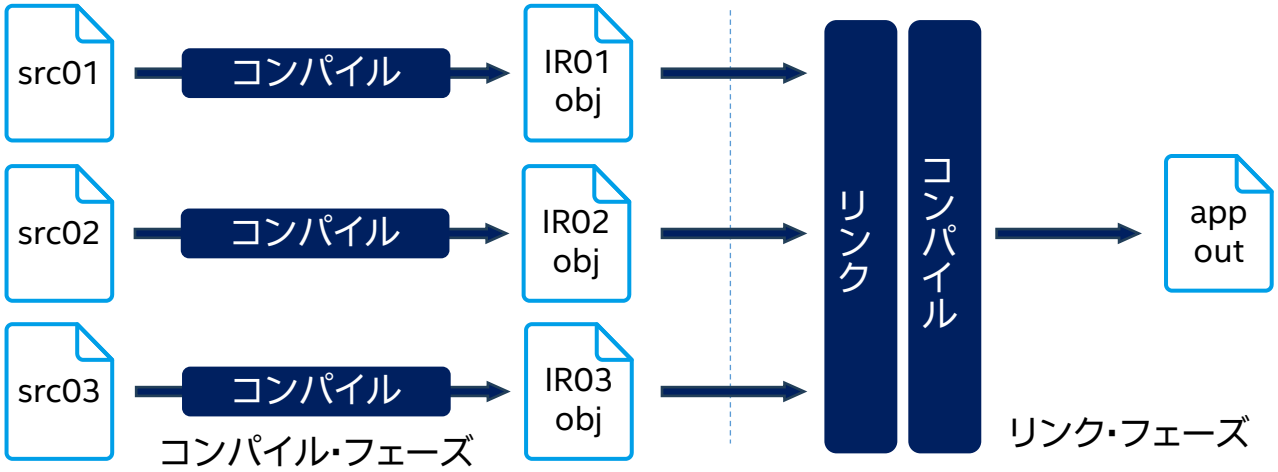
- 新しいインテル® コンパイラーでは LLVM が提供する LTO 機能に置き換え
- -flto、もしくは -fto=<arg> を指定すると LTO 機能を呼び出し

```
> ifx -O2 -xcore-avx2 -ipo source.f90
```

  - ✓ arg = thin もしくは full (デフォルト)
  - ✓ ThinLTO は full LTOと比較してメモリ消費量を抑えつつビルド時間の短縮を目指した異なる LTO の手続きを提供します
- IPO を有効化するコンパイラー・オプション -ipo は icx/icpx、ifx において -flto と解釈

LTO によるコンパイルとリンク

- コンパイル・フェーズ:
  個々のソースファイルをコンパイルして中間表現のオブジェクトを生成
- リンク・フェーズ:
  オブジェクトファイルを集約し、追加の最適化を含むコンパイルを実施



例 : 関数のインライン展開

- コンパイラーはデフォルトで単一ファイル内の関係を識別
- シンプルな例としてループ内で異なるファイルに定義された関数の呼び出しはループやベクトル化に関する最適化が抑制されやすい
  - ✓ 関数内の処理がインライン展開によりベクトル化して問題ないと評価された場合自動ベクトル化機能によってベクトル化されます

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

異なるソースファイルに記述された func 関数の実装

```
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;
}
```

プロファイルに基づく最適化 - PGO

- プログラム実行時にプロファイル情報を生成して、
  そのプロファイル情報をもとに再度コンパイル
- 実行時の情報をもとにCPU の命令キャッシング、
  メモリー・ページング、特に分岐予測の最適化を支援
  - ✓ より適切なレジスターの割り当て
  - ✓ 入れ子の If 文や、Switch 文の分岐最適化
  - ✓ 関数のインライン展開、自動ベクトル化、自動並列化による性能評価
  - ✓ 基本ブロックや関数の並び替え
    など…

PGO のアプローチ

- 新しいインテル® コンパイラーは2 つのアプローチを採用
- インストルメント方式の PGO
  - ✓ プログラムにプロファイル情報を取得する処理を挿入
  - ✓ 以前のインテル® コンパイラーでは、この方式が採用されており、
    新しいインテル® コンパイラーにおいては Clang が提供する機能に置き換え

    icc/icpc、ifort のオプション -prof-gen、-prof-use
    icx/icpx、ifx のオプション　-fprofile-generate、-fprofile-use
- サンプリング方式の PGO
  - ✓ ハードウェア支援によるサンプリングベースのプロファイリング情報を利用
  - ✓ 現時点では icx/icpx のみ利用可能

サンプリング方式の PGO

- 従来のインストルメント方式のPGOを代替する手段として提案され、2024.0.0 以降に機能追加された新しい PGO の実装
  - ✓ ハードウェア PGO (HWPGO) と呼ばれます
- HWPGOはインテル® プロセッサーに搭載されているパフォーマンスモニタリングユニット(PMU)を利用
- インストルメント方式と比較して低オーバーヘッドかつ、より高精度なプロファイル情報を収集、利用できることが特徴
  - ✓ 計測用のバイナリーが不要

パフォーマンス・モニタリング・ユニット - PMU

- PMUは CPU 以外にもメモリー、ストレージを含む
  幅広いハードウェアイベントを提供
- 提供するハードウェアイベントの例:
  - ✓ 実行された命令:
    プロセッサによって正常に実行された命令
  - ✓ キャッシュミス:
    キャッシュメモリーにデータや命令が見つからず、より高レベルの
    キャッシュメモリーもしくはメインメモリーから取得
  - ✓ 分岐予測ミス:
    プロセッサが分岐命令のパスを誤って予測
  - ✓ メモリアクセス:
    ロード/ストア回数などのメモリーアクセス

HWPGO の利用

- 機能変更や強化が積極的に実施されており、
  将来的に利用手順が変更される可能性があります
  - ✓ HWPGO チュートリアル
    ここでは分岐予測に関するハードウェアイベントを追加した HWPGO を実施
    Hardware-based Profile Guided Optimization (intel.com)

1. 事前準備: -fprofile-sample-generate を指定してコンパイル
   - ✓ 最適化に関連するオプションは併用できます

```
> icx -O3 -fprofile-sample-generate unpredictable.c nop.c -o unpredictable
```

HWPGO の利用

2. プログラムを実行して perf もしくは sep によるプロファイル情報を取得

```
> perf record -b -e BR_INST_RETIRED.NEAR_TAKEN:uppp,
BR_MISP_RETIRED.ALL_BRANCHES:upp -c 1000003 -- ./unpredictable
```

- ✓ BR_INST_RETIRED.NEAR_TAKEN、
  BR_MISP_RETIRED.ALL_BRANCHES は
  プロファイル情報として利用されるハードウェアイベント

HWPGO の利用

3. llvm-profgen を使いプロファイル情報をコンパイラーが参照できるように変換

```
> llvm-profgen ¥
  --format text ¥
  --output=misp.prof ¥
  --binary=unpredictable ¥
  --sample-period=1000003 ¥
  --perf-event=cpu_core/BR_MISP_RETIRED.ALL_BRANCHES/upp ¥
  --leading-ip-only ¥
  --perfdata=perf.data
```

```
> llvm-profgen ¥
  --format text ¥
  --output=freq.prof ¥
  --binary=unpredictable ¥
  --sample-period=1000003 ¥
  --perf-event=cpu_core/BR_INST_RETIRED.NEAR_TAKEN/uppp ¥
  --perfdata=perf.data
```

✓ インテル® コンパイラーに同梱されている llvm ツールを利用

llvm-profgen から得られたプロファイル情報

```
unpredictable:12756426100:0
 0: 0
 3.1: 201515726
 3.2: 201515726
 5: 201515726
 6: 116257832
 7: 116257832
 8: 119741696 nop:119741696
 9: 114806222
 11: 86644988 nop:86644988
 13: 201515726
 15: 0
 65517: 116257832
nop:199289924:206386684
 1: 199289924
```

```
unpredictable:168000504:0
 0: 0
 3.1: 0
 3.2: 0
 5: 84000252
 6: 0
 7: 0
 8: 0
 9: 0
 11: 0
 13: 0
 15: 0
 65517: 0
```

HWPGO の利用

4. オプション: 複数のコードパスや異なるデータセットを使用する場合は
   手順 2 と 3 を複数回実施
   - ✓ llvm-profgen merge コマンドによって生成された複数のファイルをマージ
5. -fprofile-sample-use を指定して再コンパイル

```
> icx -O3 -fprofile-sample-use=freq.prof ¥
-mllvm -unpredictable-hints-file=misp.prof ¥
unpredictable.c nop.c -o unpredictable.hwpgo
```

- ✓ 分岐予測に関する追加のフィードバックは
  -unpredictable-hints-file オプションを通して LLVM に指定
- ✓ freq.prof のみの指定はインストルメント方式の PGO の最適化に相当

HWPGO 適用する前と後の比較
perf stat を使用したイベント

HWPGO の実施前

```
xlsoftkk@xlsta750:~/repos/hwpgo-mispredict-example$ perf stat -e cycles:u,instruction

 Performance counter stats for './unpredictable':

       484,671,386      cpu_atom/cycles:u/
     3,226,244,867      cpu_core/cycles:u/
       662,616,145      cpu_atom/instructions/
     3,683,104,558      cpu_core/instructions/
       141,269,434      cpu_atom/br_inst_retired.all_branches:u/
       931,431,248      cpu_core/br_inst_retired.all_branches:u/
        12,160,248      cpu_atom/br_misp_retired.all_branches/
        84,767,838      cpu_core/br_misp_retired.all_branches/

       0.709172520 seconds time elapsed

       0.705270000 seconds user
       0.003995000 seconds sys
```

HWPGO の実施後

```
xlsoftkk@xlsta750:~/repos/hwpgo-mispredict-example$ perf stat -e cycles:u,instruction

 Performance counter stats for './unpredictable.hwpgo':

        55,786,769      cpu_atom/cycles:u/
       814,786,022      cpu_core/cycles:u/
       250,920,531      cpu_atom/instructions/
     4,060,112,009      cpu_core/instructions/
        18,579,000      cpu_atom/br_inst_retired.all_branches:u/
       608,679,745      cpu_core/br_inst_retired.all_branches:u/
           594,619      cpu_atom/br_misp_retired.all_branches/
            12,770      cpu_core/br_misp_retired.all_branches/

       0.184502821 seconds time elapsed

       0.180655000 seconds user
       0.003992000 seconds sys
```

最適化レポート機能

- 各フェーズを対象にコンパイラーが実行した最適化を説明
  - ✓ 成功した最適化
  - ✓ 何らかの理由で失敗した最適化
  - ✓ 成功した最適化と失敗した最適化に関する詳細情報
- 2025.0.0 にて機能が強化され以前のインテル® コンパイラーに類似したレポート形式の出力および指定方法が可能に

最適化レポートの生成

レベルごとの大まかな出力内容
<n> - 0: 無効
<n> - 1: 最適化の可否
<n> - 2: 失敗した最適化の理由
<n> - 3: より詳細な説明文

- -qopt-report=<n>

```
> ifx -O2 -xcore-avx2 -qopt-report=2 source.f90
```

  - ✓ レポートの詳細レベル: <n> = 1~3, デフォルトは 2
- 最適化レポートが記載されたテキスト形式の *.optrpt ファイルを生成
  - ✓ デフォルトの動作では *.optrpt ファイルはソースコードごとに生成されます
- 関数、ループ、OpenMP の並列領域、等の特定のグループごとに出力
  - ✓ レポートはソースファイル名とソース行に対応

例: matmul.f90 ファイルの 26行目に存在する ループ処理のベクトル化に対する最適化レポート

```
LOOP BEGIN at ./matmul.f90 (26, 13)
    remark #25566: blocked by 64
    remark #25563: Load hoisted out of the loop
    remark #15300: LOOP WAS VECTORIZED
    remark #15305: vectorization support: vector length 8
    remark #15475: --- begin vector loop cost summary ---
    remark #15476: scalar cost: 83.000000
    remark #15477: vector cost: 14.281250
    remark #15478: estimated potential speedup: 5.796875
    remark #15309: vectorization support: normalized vectorization overhead 0.125000
    remark #15488: --- end vector loop cost summary ---
    remark #15447: --- begin vector loop memory reference summary ---
    remark #15450: unmasked unaligned unit stride loads: 32
    remark #15451: unmasked unaligned unit stride stores: 16
    remark #15474: --- end vector loop memory reference summary ---
    remark #25583: Number of Array Refs Scalar Replaced In Loop: 12
LOOP END
```

特定の最適化フェーズにフィルター

- レポートが出力する項目は
  -qopt-report-phase=<キーワード> により制御

```
> ifx -O2 -xcore-avx2 -qopt-report=3 -qopt-report-phase=vec source.f90
```

  ✓ 特定の種類の最適化に関する情報のみ出力

- 複数のキーワードを指定可能

  ✓ デフォルトは all で動作

```
> ifx -qopt-report -qopt-report-phase=vec,openmp …
```

キーワード	対象のフェーズ
cg	コード生成
ipo	LTO 機能
loop	ループの最適化
openmp	OpenMP*
pgo	PGO 機能
vec	ベクトル化
all	全て

サニタイザー

- プログラムの実行時に解析を行い潜在する問題を検出
  - ✓ チェック機能はコンパイラーにより実行ファイルにインストルメントされる
  - ✓ 一般的な動的解析ツールと比較して軽量に動作するが再コンパイルが必要
- サニタイザー機能に関するチュートリアル
  - ✓ Find Bugs Quickly Using Sanitizers (intel.com)

<sanitizer>

※一部は icx/icpx のみ実装

- -fsanitize の指定
  - ✓ 検出時にソースファイル名などの追加情報を得るためにデバッグ情報を付与
  - ✓ 例: アドレスサニタイザーの有効化

```
> ifx -g -O0 -fsanitize=address source.f90
```

<sanitizer>	説明
address	メモリーアクセスにおける領域外参照や解放後の利用の検出
undefined	未定義動作の検出
memory	未初期化のメモリー参照の検出 ※ ifx では -check uninit オプションと同等
thread	スレッド間のデータ競合の検出

例: OpenMP* スレッドのデータ競合

```
> ifx -fiopenmp -fsanitize=thread -g -O2 *.f90
> ./a.out
```

```
26  !$OMP PARALLEL DO
27  do i=1, n
28    x = h * (DBLE(i)-0.5d0)
29    sum = sum + f(x)
30  end do
31  !$OMP END PARALLEL DO
```

```
WARNING: ThreadSanitizer: data race (pid=1185457)
  Write of size 8 at 0x7fffa30eabe8 by thread T15:
    #0 pi_calc_.DIR.OMP.PARALLEL.LOOP.232.split36 /home/xlsoftkk/work/pi_calc/pi-calc.f90:28:5 (a.out+0x4f039e) (BuildId:
a25fdbb986fd385726123d7a165267d46a0c4e57)
    #1 __kmp_invoke_microtask <null> (libiomp5.so+0x167d92) (BuildId: 0b92a2ad5b16d95f07791fdede39499bf1d23cea)

  Previous write of size 8 at 0x7fffa30eabe8 by main thread:
    #0 pi_calc_.DIR.OMP.PARALLEL.LOOP.232.split36 /home/xlsoftkk/work/pi_calc/pi-calc.f90:28:5 (a.out+0x4f039e) (BuildId:
a25fdbb986fd385726123d7a165267d46a0c4e57)
    #1 __kmp_invoke_microtask <null> (libiomp5.so+0x167d92) (BuildId: 0b92a2ad5b16d95f07791fdede39499bf1d23cea)
    #2 pi_calc_.void /home/xlsoftkk/work/pi_calc/main.f90 (a.out+0x4eee0c) (BuildId: a25fdbb986fd385726123d7a165267d46a0c4e57)
    #3 MAIN__ /home/xlsoftkk/work/pi_calc/main.f90:16:18 (a.out+0x4eee0c)
    #4 main <null> (a.out+0x4346ac) (BuildId: a25fdbb986fd385726123d7a165267d46a0c4e57)

  Location is stack of main thread.

  Location is global '??' at 0x7fffa30cb000 ([stack]+0x1fbe8)

  Thread T15 (tid=1185473, running) created by main thread at:
    #0 pthread_create <null> (a.out+0x43c51f) (BuildId: a25fdbb986fd385726123d7a165267d46a0c4e57)
    #1 __kmp_create_worker <null> (libiomp5.so+0x169587) (BuildId: 0b92a2ad5b16d95f07791fdede39499bf1d23cea)
    #2 pi_calc_.void /home/xlsoftkk/work/pi_calc/main.f90 (a.out+0x4eee0c) (BuildId: a25fdbb986fd385726123d7a165267d46a0c4e57)
    #3 MAIN__ /home/xlsoftkk/work/pi_calc/main.f90:16:18 (a.out+0x4eee0c)
    #4 main <null> (a.out+0x4346ac) (BuildId: a25fdbb986fd385726123d7a165267d46a0c4e57)

SUMMARY: ThreadSanitizer: data race /home/xlsoftkk/work/pi_calc/pi-calc.f90:28:5 in pi_calc_.DIR.OMP.PARALLEL.LOOP.232.split36
```

OpenMP* の利用

- -qopenmp を指定

```
> ifx -fiopenmp source.f90
```

- 参考: TARGET ディレクティブによる GPU へのオフロード機能
  - ✓ GPU のサポートはインテル製 GPU に制限
  - ✓ -fopenmp-targets、/Qopenmp-targets

    あわせて fopenmp-targets = spir64 を指示ください

```
> ifx -fiopenmp -fopenmp-targets=spir64 source.f90
```

- SIMD ディレクティブの有効化には -fiopenmp もしくは -qopenmp-simd を指定

新しいインテル® コンパイラーに移行

- 移行ガイド:
  - ✓ Porting Guide for ifort Users to ifx (intel.com)
  - ✓ Porting Guide for DPCPP or ICX (intel.com)
- ifx はパッケージバージョン 2023.0.0 以降を利用ください
  - ✓ 2022 以前の ifx はベータ版です
    できるだけ最新バージョンの利用をおすすめします

新しいインテル® コンパイラーに移行

- それぞれ対応したコマンドに変更
  - ✓ icc → icx、icpc → icpx、ifort → ifx
  - ✓ mpiifort、mpiicc 等のラッパーコマンドは、mpiicx、mpiifx に対応

icc	icx
icpc	icpx
ifort	ifx
mpiicc	mpiicx
mpiicpc	mpiicpx
mpiifort	mpiifx

新しいインテル® コンパイラーに移行

- icx/icpx はソースファイルの拡張子による動作を変更しません
  - ✓ C++ ファイル (.cpp) をコンパイルする場合は icpx を使用
- インテル実装のリンカー、アーカイバーは削除
  - ✓ makefile 等で xild、xiar を利用している場合は、ld、lld や ar 等のネイティブリンカーツールに切り替え

コンパイラー・オプションのマッピング

- コンパイラー・オプションの多くは
  新しいインテル® コンパイラーでも引き続き実装
  - ✓ 一方で同じコンパイラー・オプションでも動作や機能は異なる

    icc/icpc、ifort　インテルによるコンパイラー実装
    icx/icpx　LLVM/Clang に基づくインテルの拡張
    ifx　ifort フロントエンドと LLVM の接続、およびインテルの拡張
- 浮動小数点演算の動作モデルは Clang のデフォルトと異なる
  - ✓ icx:　-fp-model=fast
  - ✓ clang:　-fp-model=precise

コンパイラー・オプションのマッピング

- icx/icpx は Clang のコンパイラー・オプションを認識
  - ✓ Clang にコンパイラー・オプションを渡すには -Xclang <Clang オプション>
    LLVM にリンカー・オプションを渡す場合、-mllvm <オプション>
  - ✓ 再掲: インテル® コンパイラーが採用している Clang バージョンは
    コンパイラーリリースノートに記載

    Intel® oneAPI DPC++/C++ Compiler Release Notes (intel.com)

コンパイラー・オプションの対応
最適化オプション

- -x,-ax オプションの指定を推奨
  - ✓ デフォルトは多くの最適化に LLVM 標準の機能を使用
  - ✓ zmm レジスターの利用は引き続き -qopt-zmm-usage=high を追加で指定
- -fast オプションの動作は以前のインテル® コンパイラーと異なる
  - ✓ fast はプログラムの実行速度を引き上げるために複数のオプションを有効化

icc/icpc, ifort	-ipo, -O3, -no-prec-div,-static, -fp-model fast=2, -xHost
icx/icpx, ifx	-ipo, -O3, -static, -fp-model fast

削除されたコンパイラー・オプション

- 削除されたコンパイラー・オプションを使用すると警告

```
ifx: command line warning #10148: option '/Qparallel' not supported
```

```
icx: command line warning #10430: Unsupported command line options encountered
These options as listed are not supported.
For more information, use '-qnextgen-diag'.
option list:
        -parallel
```

- -qnextgen-diag により、削除されたコンパイラー・オプションを表示

```
> ifx -qnextgen-diag
...
options being removed:
...
```

自動並列化機能

- 自動並列化機能はコード中のループ処理を対象に OpenMP* 実装のマルチスレッド化された処理に変換
- 新しいコンパイラーでは自動並列化機能が削除され -parallel オプションはサポートされていません
  - ✓ OpenMP* 等を用いて手動でマルチスレッド化が必要

自動並列化機能

- 最適化レポートから自動並列化機能が有効化されたループ処理を確認
  - ✓ -qopt-report、-qopt-report-phase=par の組み合わせ

```
> icc -parallel -qopt-report -qopt-report-phase=par source.f90
```

```
LOOP BEGIN at ./int_sin.c(25,3)
   remark #17104: loop was not parallelized: existence of parallel dependence

   LOOP BEGIN at ./int_sin.c(33,5)
      remark #17109: LOOP WAS AUTO-PARALLELIZED
```

```
32  sum = INTEG_FUNC(interval_begin) * step / 2.0;
33  for (i = 1; i < N; i++)
34  {
35        x_i = i * step;
36        sum += INTEG_FUNC(x_i) * step;
37  }
```

自動並列化機能

- 自動並列化機能が指定した OpenMP* 指示句を表示
  - ✓ -qopt-report=3 以上が必要

```
> ifort -parallel -qopt-report=5 -qopt-report-phase=par source.f90
```

```
LOOP BEGIN at ./int_sin.c(33,5)
    remark #17109: LOOP WAS AUTO-PARALLELIZED
    remark #17101: parallel loop shared={ .2 } private={ } firstprivate={ step i j }
                   lastprivate={ } firstlastprivate={ } reduction={ sum }
```

- レポートは -x、-march を含む最適化の影響を受けます
  - ✓ 読みやすくするために一時的に-no-vec などの最適化を抑制するオプションを併用するとよいかもしれません

OpenMP* オプション

- OpenMP* 指示句をコンパイラーに認識させるには
  -fiopenmp もしくは -qopenmp を指定
  - ✓ どちらも同等の機能を有します

```
> ifx -fiopenmp source.f90
```

- -fopenmp も認識しますが icx/icpx では非推奨

icx/icpx における診断オプションと診断メッセージの番号付けの変更

- icc/icpc では warning、remark などのコンパイラーから出力される各メッセージは番号付きで表示
  - ✓ これらの warning、remark は -diag-type=diag-list 形式による制御が可能ですが icx/icpx では利用できません

```
> icc -diag-disable:161 sample.c
```

- icx/icpx では Clang が採用している warning、remark の表示および制御に準拠
  - ✓ Clang ドキュメント: Diagnostic flags 参照

    Diagnostic flags in Clang (clang.llvm.org)

プラグマ/ディレクティブのサポート

- 一部の実装を削除しておりプラグマ/ディレクティブを認識しない場合があります
  - ✓ Intel-Specific Pragma Reference (intel.com)
  - ✓ Compiler Directives (intel.com)
  - ✓ icx/icpx において-Wunknown-pragmas により
    ポートされていないプラグマの使用を警告
    これはデフォルトで有効

```
> icx -Wunknown-pragmas vec_add.c

vec_add.c(89,9): warning: unknown pragma ignored [-Wunknown-pragmas]
#pragma parallel
        ^
1 warning generated.
```

ifx 向け検証用のコンパイラー・オプションのセット

```
> ifx -g -O0 -warn all -check all -traceback -qno-openmp-simd
 -standard-semantics -fp-model=precise source.f90
```

- 最適化オプションの無効化とデバッグ用のオプションを追加
- コンパイラーの動作の違い、最適化オプションによる影響、に対する要因の切り分けに有効です

コンパイラー・オプション	効果
-O0, /Od	最適化の無効化
-warn all	コンパイル時の警告を全て出力
-check all	ランタイムチェックの有効化
-g	デバッグ用シンボルを作成する
-traceback	実行時エラーのトレースバックを出力 (-g もしくは /debug:full を追加してください)
-standard-semantics	Fortran 言語仕様に準拠した動作
-qno-openmp-simd	OpenMP* SIMD ディレクティブの無効化
-fp-model=precise,	精度に影響する最適化の抑止

浮動小数点演算の再現性

- 浮動小数点演算は近似値を求めることとなり演算結果には誤差を含みます
  - ✓ 期待される演算結果の精度はプログラムの要求に応じて決定
    運用可能な範囲で誤差を丸める
- "演算結果の再現性"、"計算時間の短縮"はトレードオフの関係
  - ✓ 例えば最適化を適用することで、いままでと異なる演算結果を出力することがあります
    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 | iSUS
- 一貫した演算結果を期待する場合、インテル® コンパイラーでは
  -fp-model オプションにより制御します

浮動小数点演算の再現性

- -fp-model=<type>

```
> ifx -O3 -xhost -fp-model=precise source .f90 -o run.out
```

- -fp-model の指定は数学ライブラリ関数の
  精度に影響するコンパイラー・オプションの動作もあわせて変更されます
- ifx の利用において整数の結果が大きく変わる場合、整数オーバーフローの
  扱いの変更に伴って発生している可能性があります
  - ✓ -fstrict-overflow オプションをお試しください

-fp-model=<type>

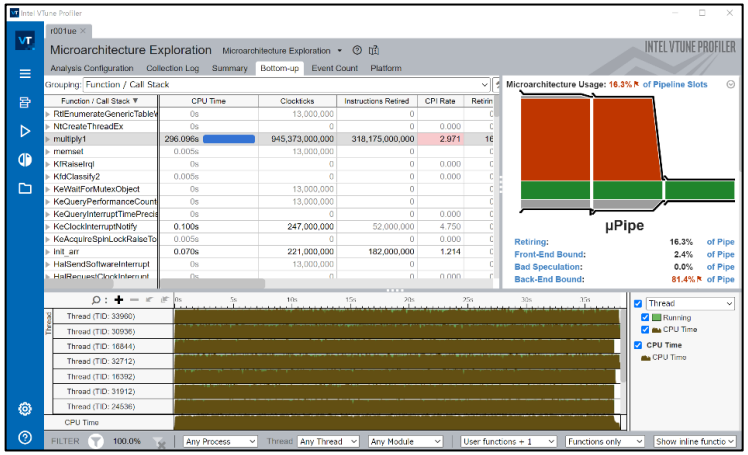
■ ※一部を抜粋

最適化	type	説明
積極的	fast[=1|2]	演算結果に影響がある最適化を許可 fast (fast=1) がインテル® コンパイラーにおけるデフォルト fast=2 が指定されると、さらにいくつかの追加の近似を許可
	precise	演算結果に影響しない最適化のみ許可
	consistent	FMA を無効化し異なるプロセッサーや最適化レベルでも 一貫した再現性のある結果が得られる最適化のみ許可
消極的 (厳密な演算)	strict	value-safe な最適化のみ許可し、浮動小数点例外のセマンティクスを有効化

インテル® VTune™ プロファイラー

インテル® VTune™ プロファイラー

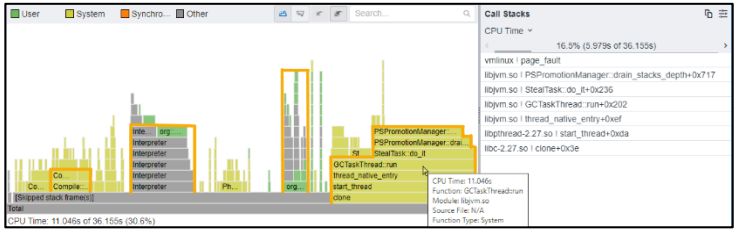
- インテル製のCPU、GPU、アクセラレーター上で実行される
  プログラムの最適化を支援するパフォーマンス・プロファイラー
  - ✓ GUI もしくは CUI から利用
- CPU 内部のハードウェア・カウンターから
  パフォーマンス・イベントを収集して表示
  - ✓ マイクロアーキテクチャー、
    メモリー、I/O などのハードウェアに
    関わる問題の調査



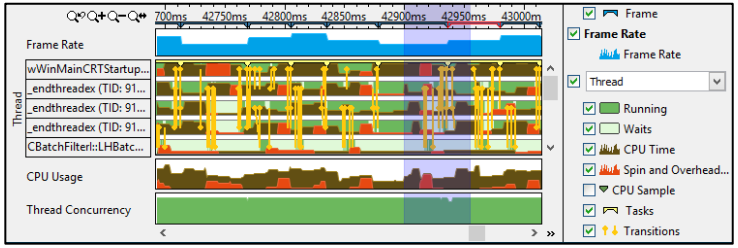
主な用途

- アルゴリズムの分析
  - ✓ マルチスレッド実行の効率
  - ✓ ホットスポットの特定
    各関数のコードパスと、その呼び出し先で
    消費した時間をフレームグラフで表示
- マイクロアーキテクチャーとメモリーの調査
  - ✓ プログラムの性能に影響を与える
    CPU 内部の実行状況
  - ✓ キャッシュミスや高帯域幅などの
    メモリーアクセス関連の問題
- システム全体の調査

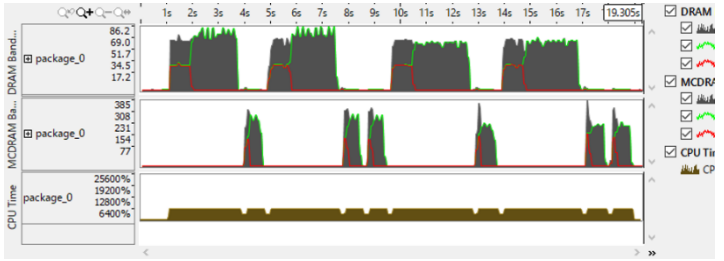
フレームグラフ



タイムライン



メモリーアクセス



インターフェース

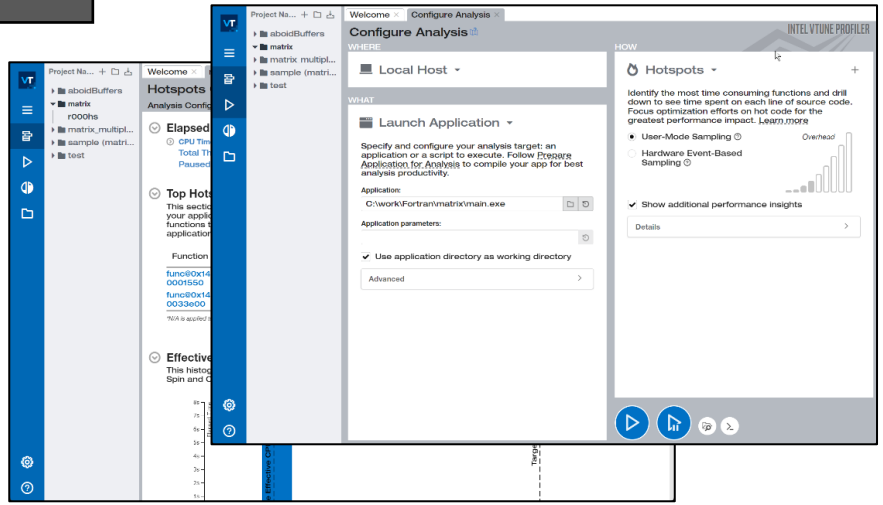
- CUI 環境の利用

```
> vtune -collect hotspots -- ./sample.out
```

```
> vtune -report hotspots -r r000hs
```

- GUI 環境の利用

```
> vtune-gui
```

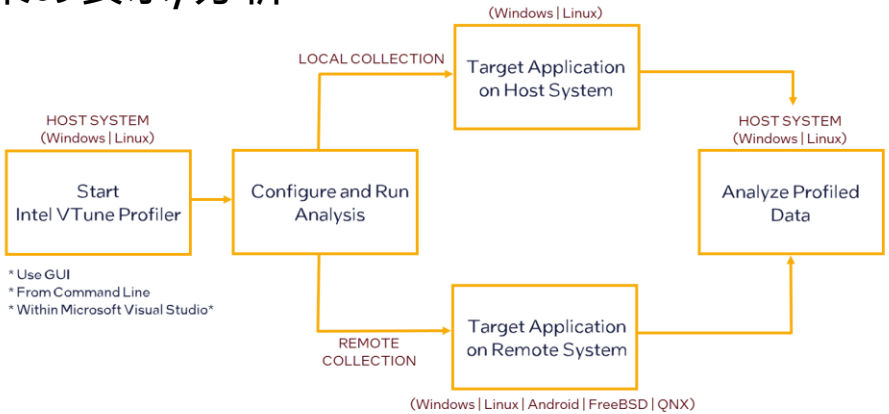


サンプリング モード

- 収集する情報にあわせて異なるサンプリング方式を使用します
- ユーザーモード サンプリング
  - ✓ User-Mode Sampling and Tracing Collection
    プロファイラーからターゲットに割り込みを入れてデータを収集します
- イベントベース サンプリング
  - ✓ Hardware Event-based Sampling Collection
    サンプリングドライバーを経由して CPU に搭載されているカウンターからイベントを収集します
  - ✓ サンプリング ドライバーは CPU 内部の PMU から提供される
    ハードウェア イベントを収集します
    PMU: Performance Monitoring Unit

基本のワークフロー

- インテル® VTune™ プロファイラーの起動
- 解析の設定
- ローカルもしくはターゲットでデータ収集の開始
- 結果の表示/分析



出典: Get Started with Intel® VTune™ Profiler (intel.com)

データ収集を行う基本コマンド

- 解析用のコマンド [Run Command Line Analysis (intel.com)]

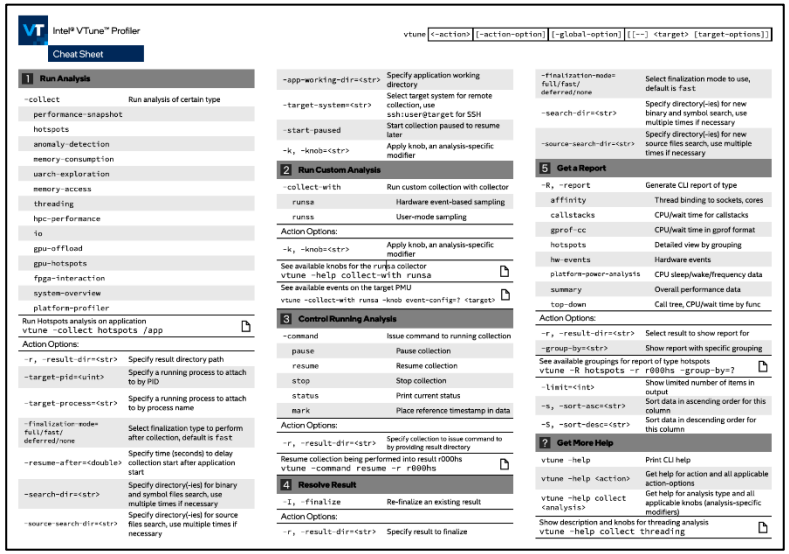
```
> vtune -collect <analysis_type> [-target-system=<system>] [--] <target>
```

✓ hotspots 解析タイプを実行

```
>vtune -collect hotspots -- test.out
```

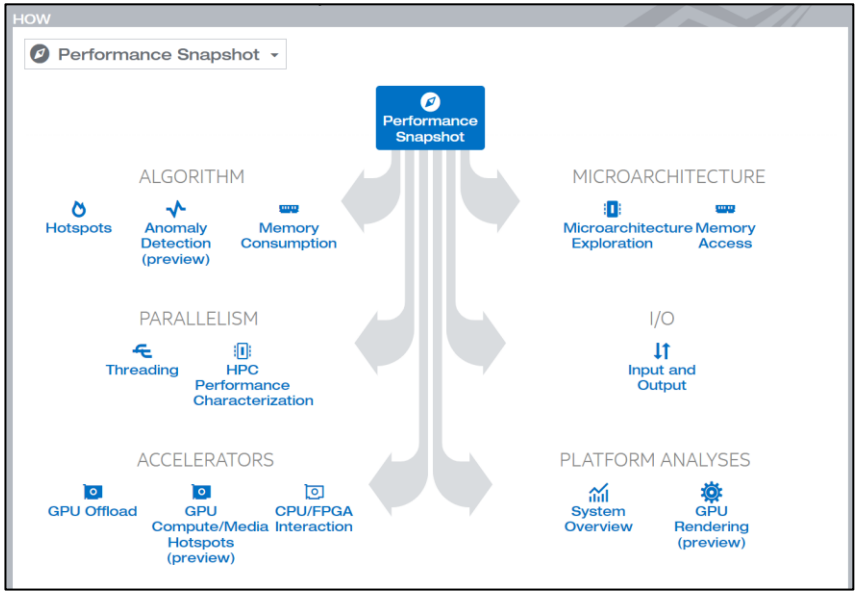
- GUI の  を選択すると、GUI の設定に対応した CUI コマンドを表示
- 参考: チートシート

  vtune-profiler-cheat-sheet.pdf (intel.com)



解析タイプ

- 得たい性能情報にあわせて
  解析タイプを選択
  - ✓ 例:
    プログラム内で時間を消費している関数
    マルチスレッド化された処理の並列実行効率
    CPU パイプラインの実行効率や命令の割合
    ロード/ストアやメモリー帯域幅の使用率



解析タイプ例

- 解析タイプのリストはヘルプから参照可能

```
> vtune --help collect
```

解析タイプ	主に収集する情報
hotspots	実行したプログラムが使用した CPU 時間 低オーバーヘッドな動作が特徴
threading	マルチスレッド化されたプログラムの効率 OpenMP* 並列領域を識別して潜在的な性能問題を評価
memory-access	キャッシュ効率およびメモリー帯域幅利用率および リモートメモリーアクセスの頻度
uarch-exploration	TMA を基にしたマイクロアーキテクチャー全般の性能
hpc-performance	CPU 使用率、ベクトル演算/FPU 使用率、メモリー帯域幅利用率の 3 つの観点から見た性能

解析コマンドの組み合わせ例

- イベントベースサンプリングの hotspots 解析タイプを実行 +
  コールスタックの収集と収集上限の指定 + 作業ディレクトリの指定

```
> vtune -collect hotspots -knob sampling-mode=hw ¥
-knob enable-stack-collection=true -knob stack-size=4096 ¥
--app-working-dir=/working_dir/path -- run.out <input_file>
```

- Threading 解析タイプを実行して結果を /result/path 以下に保存

```
> vtune -collect threading -result-dir=/result/path/r@@@{at} ¥
-- ./run.out <input_file>
```

- MPIランク0~3 を対象に hpc-performance 解析タイプを実行、
  結果を hpc_result に保存

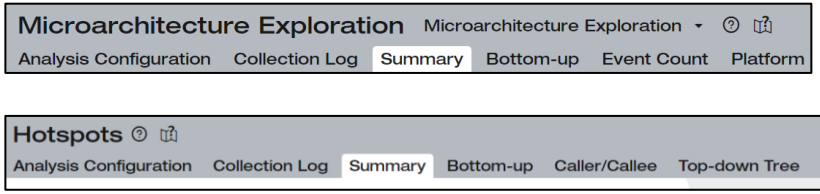
```
> mpirun -np 8 -gtool "vtune -collect hpc-performance ¥
-result-dir hpc_result:0-3" run.out <input_file>
```

結果

- 解析結果は r000hs、r001hs … といった連番のディレクトリを作成
  - ✓ -result-dir により任意のパス、名前に変更可能
- 解析の終了後に結果の概要を出力します。
  CUI から詳細を見るには -report コマンドを使用

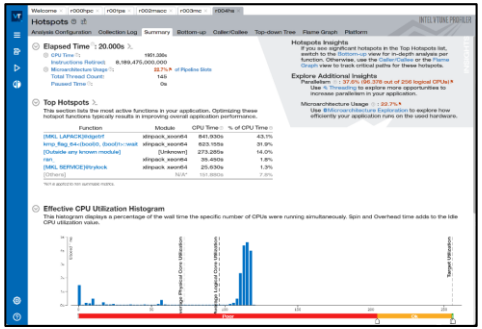
```
C:\Users\takeda\Documents\VTune\Samples\matrix>vtune -report hotspots -r r000hs
vtune: Using result path `C:\Users\takeda\Documents\VTune\Samples\matrix\r000hs'
vtune: Executing actions 75 % Generating a report                              Function                CPU Tim
----------------------  --------  ----------------------  ------------------  ----------------------  -------
multiply1               192.591s                192.591s                  0s                      0s  matrix.
NtCreateSection           0.305s                  0.305s                  0s                      0s  ntdll.d
func@0x71099770           0.217s                  0.217s                  0s                      0s  tmmon64
init_arr                  0.092s                  0.092s                  0s                      0s  matrix.
WaitForMultipleObjects    0.026s                      0s              0.026s                      0s  KERNELB
free_base                 0.012s                  0.012s                  0s                      0s  ucrtbas
vtune: Executing actions 100 % done
```

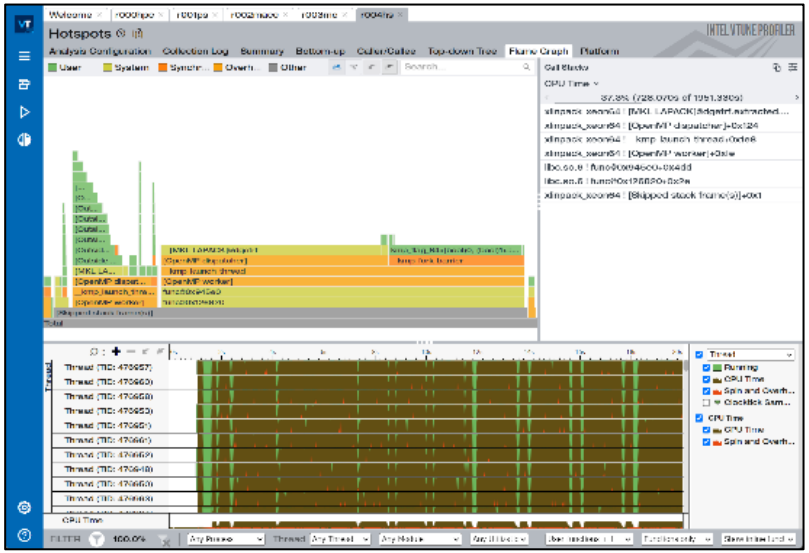
結果と GUI 各操作



- 結果は各ページごとに異なる視点から性能情報を提供
- 得られる情報は解析タイプや指定したオプションに依存します
  - ✓ フレームグラフを表示したい場合、コールスタックの収集が必要

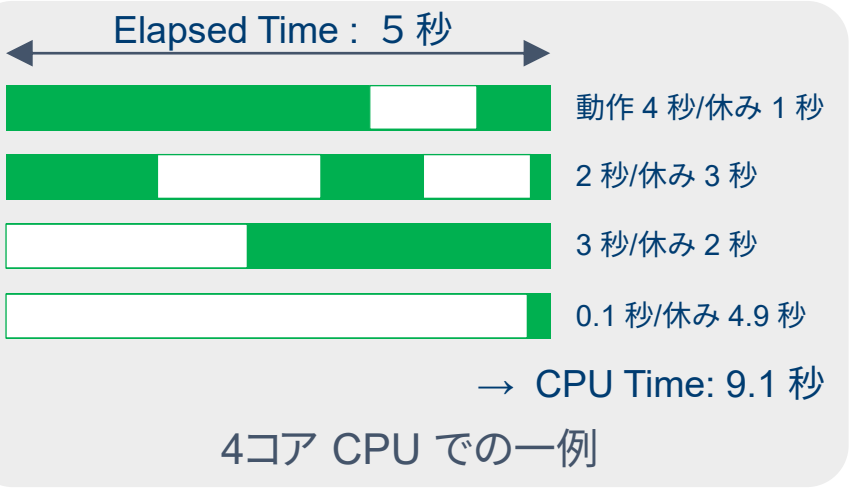
    ユーザーモードサンプリングの Hotspot 解析タイプではデフォルトで有効

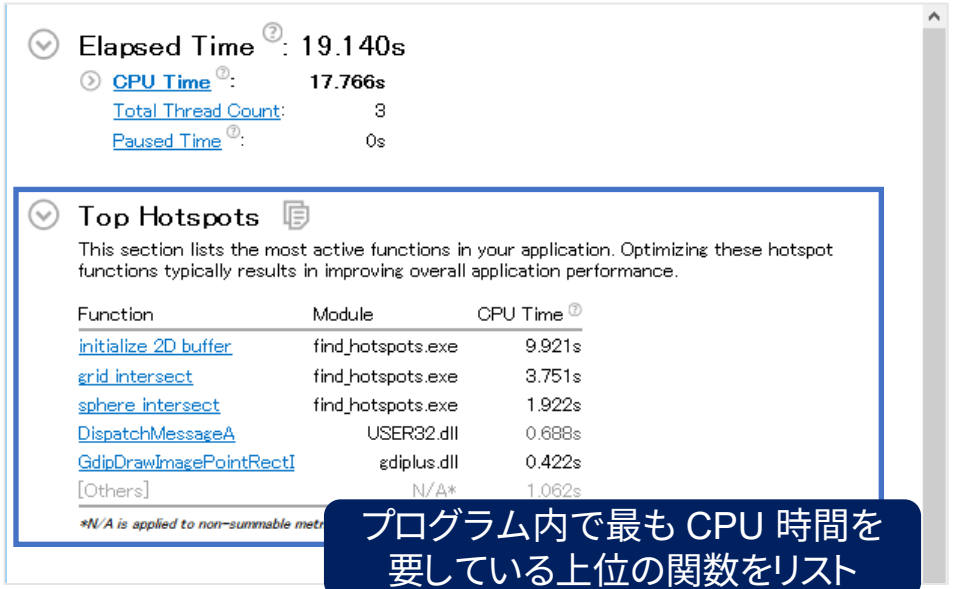




Hotspots 解析結果例

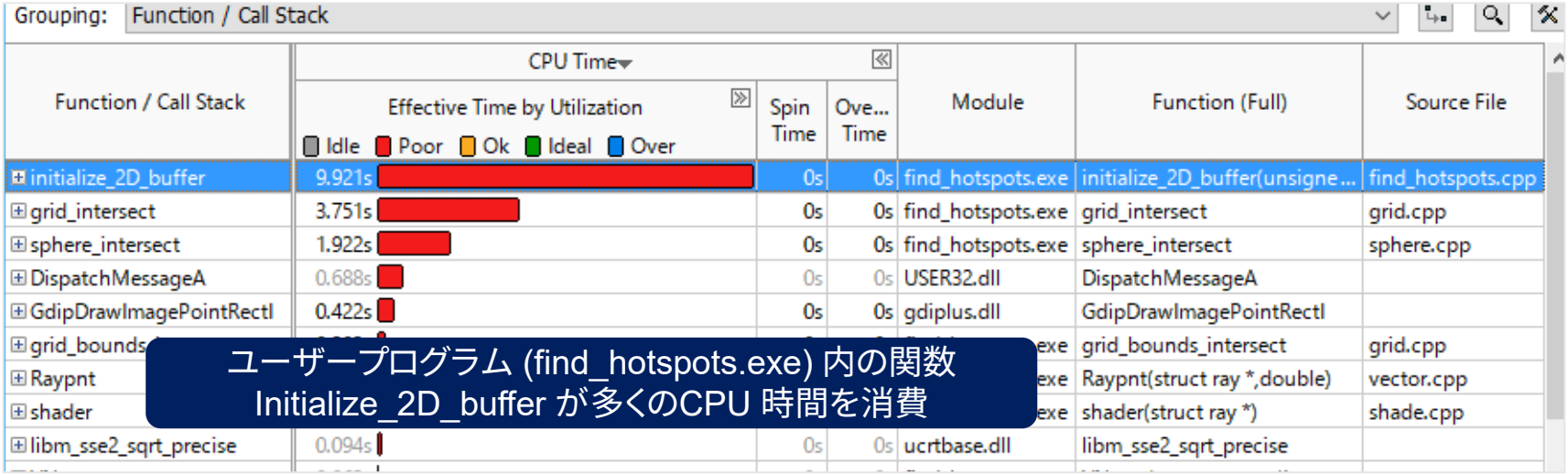
- Summary タブ
  - ✓ Elapsed Time: 経過時間 (実時間)
  - ✓ CPU Time: CPU の合計稼働時間



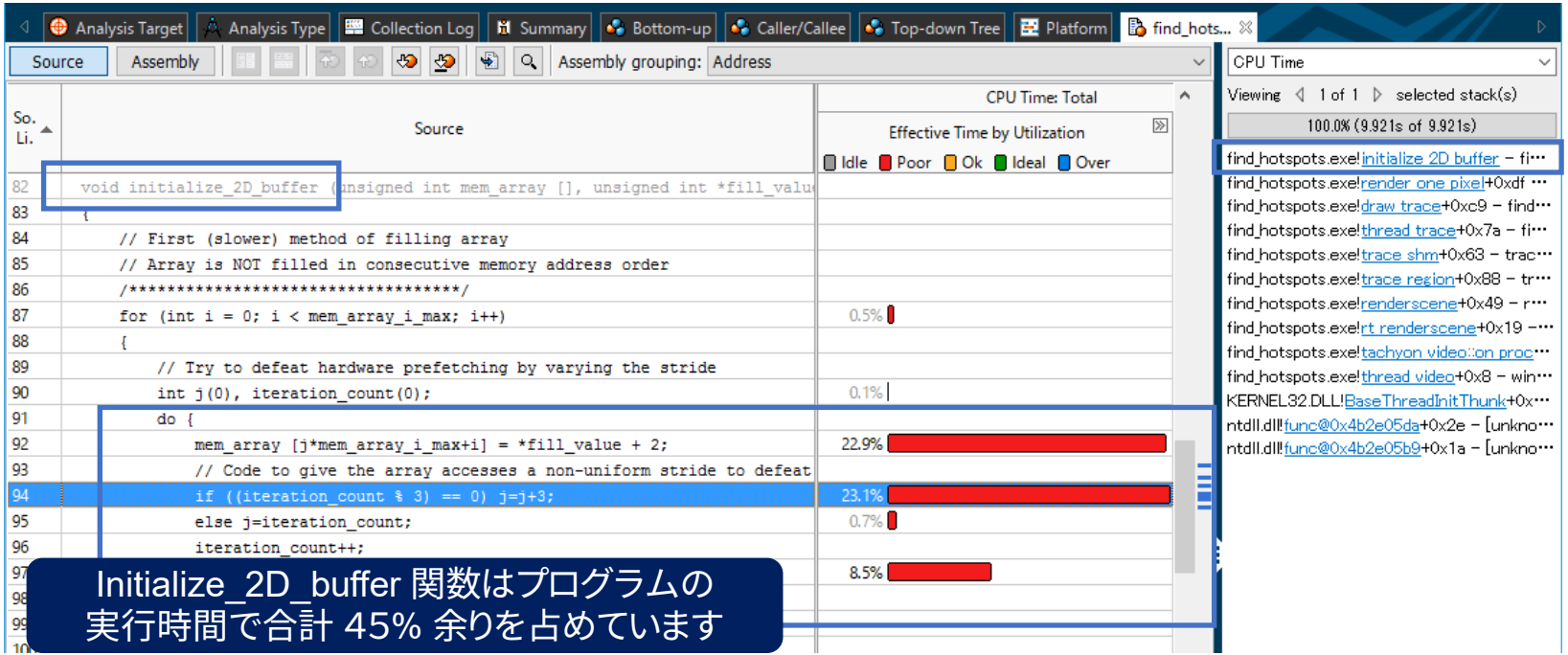


Hotspots 解析結果例

- Bottom-up (関数/項目ごとの表示) タブ

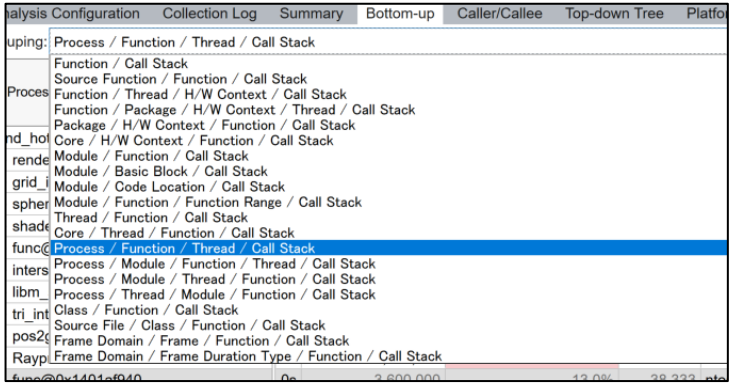


Hotspots 解析結果例

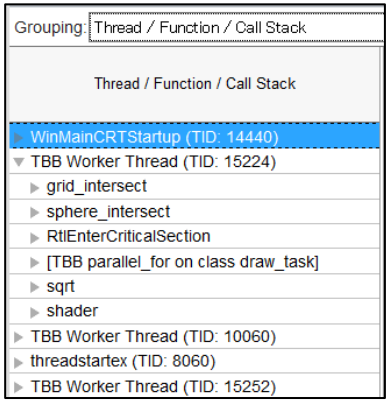


tips: Grouping の切り替え

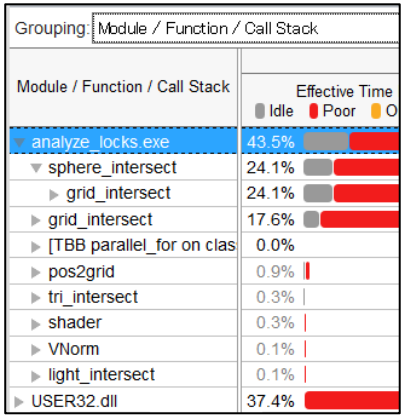
- Grouping は表示する性能情報の対応をスレッド毎、関数/ループ毎、プロセス毎などに切り替え
  - ✓ デフォルトで解析タイプに応じた Grouping が選択
    任意の Grouping の作成



Thread/Function/Call Stack
スレッドごとの関数の実行状況

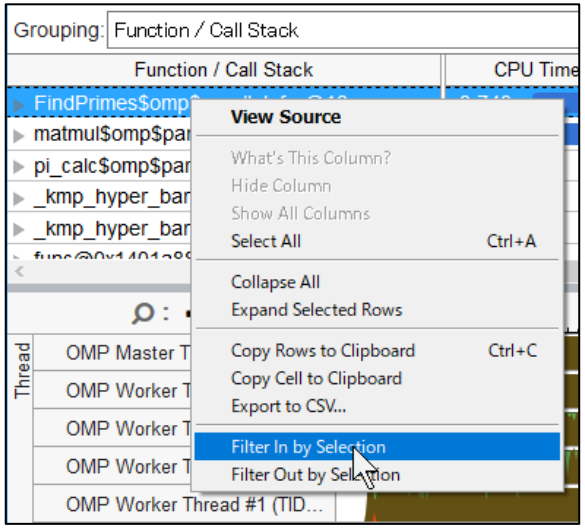


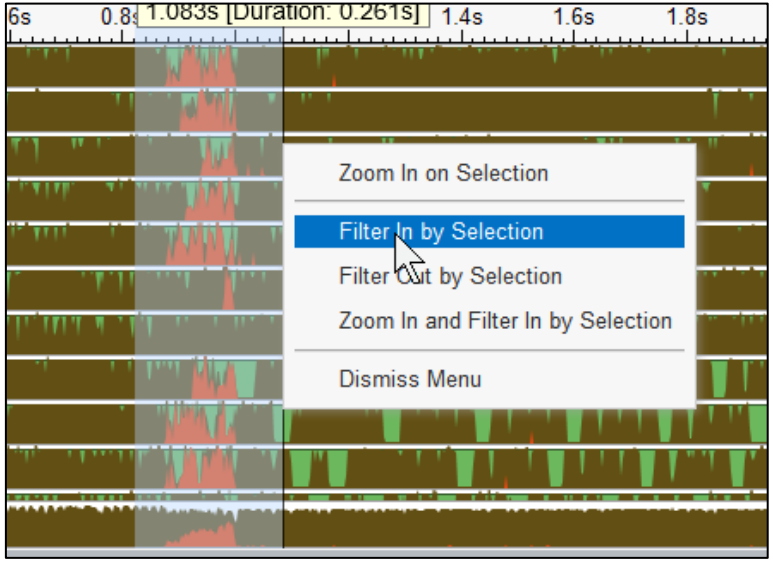
Module/Function/Call Stack
モジュール (実行ファイル、ライブラリー) ごとの関数の実行状況



tips: フィルタリング

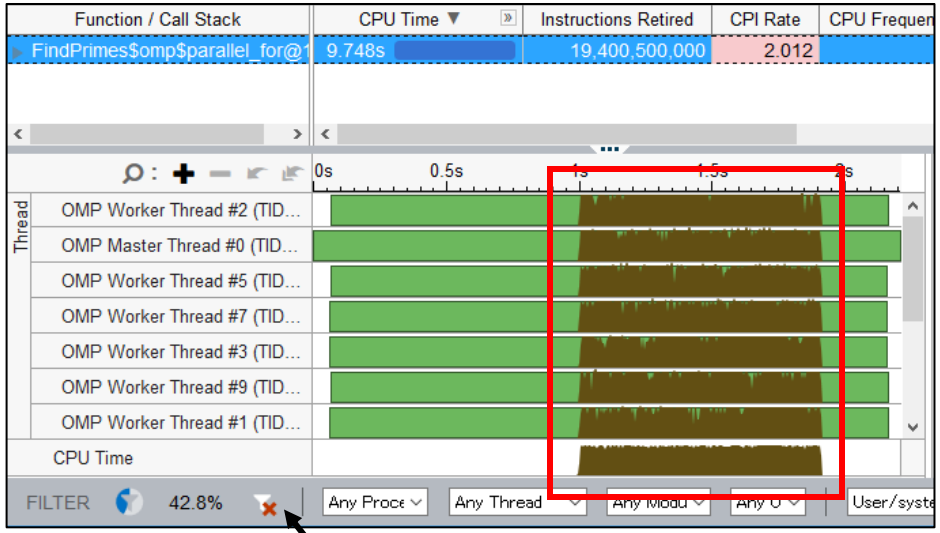
- オブジェクト、タイムラインをフィルタリングすることで、
  特定の情報だけ表示

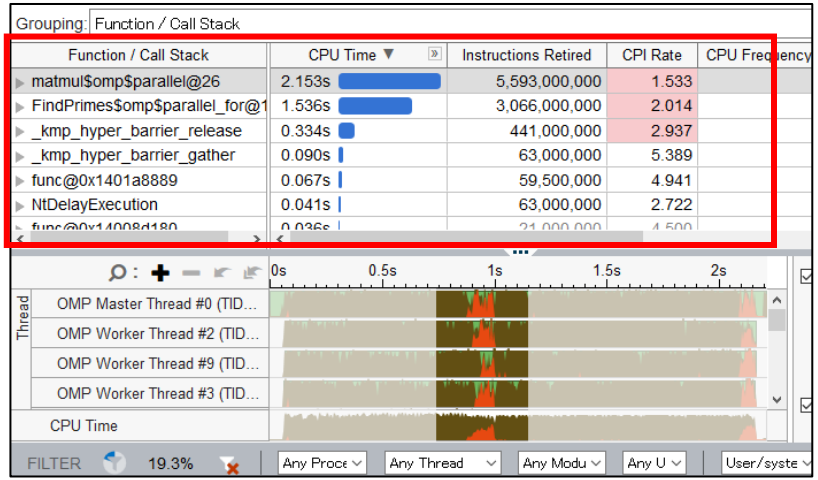




tips: フィルタリング

- タイムラインと 関数/項目のリストは連動してフィルターされます



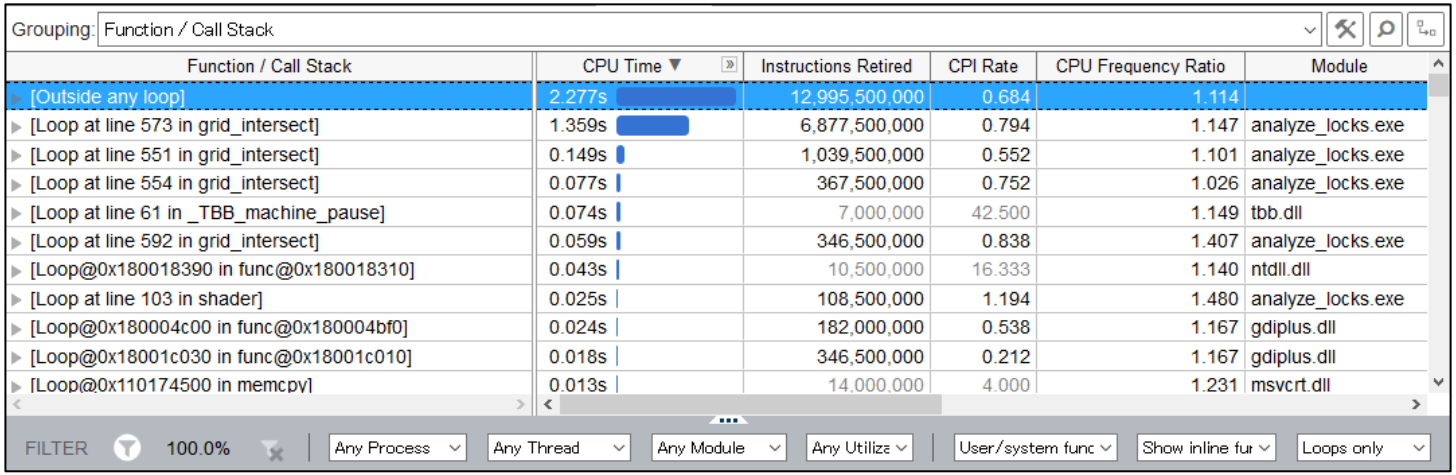


フィルタリングの解除

tips: フィルターツールバー

- いくつかの条件をもとに表示情報をフィルター

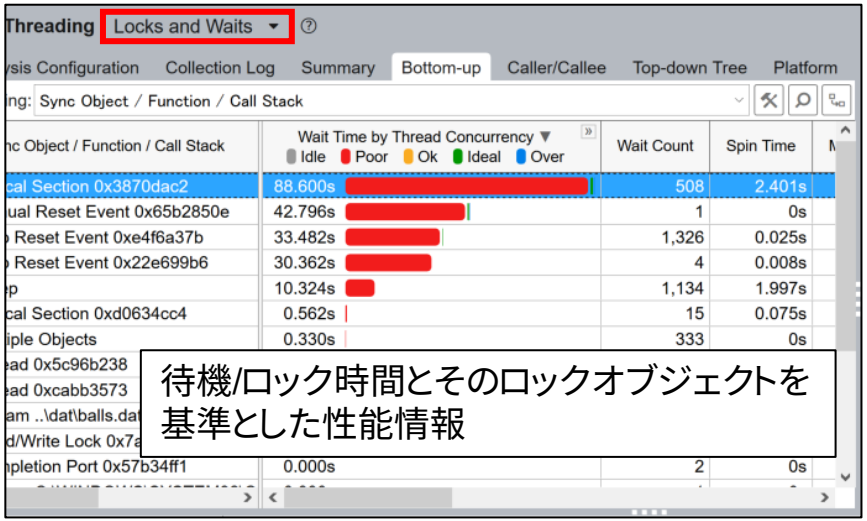
プロセス名　　スレッド番号　　モジュール名
(*.exe、*.out、*.dll、*.a)　　関数、ループ処理

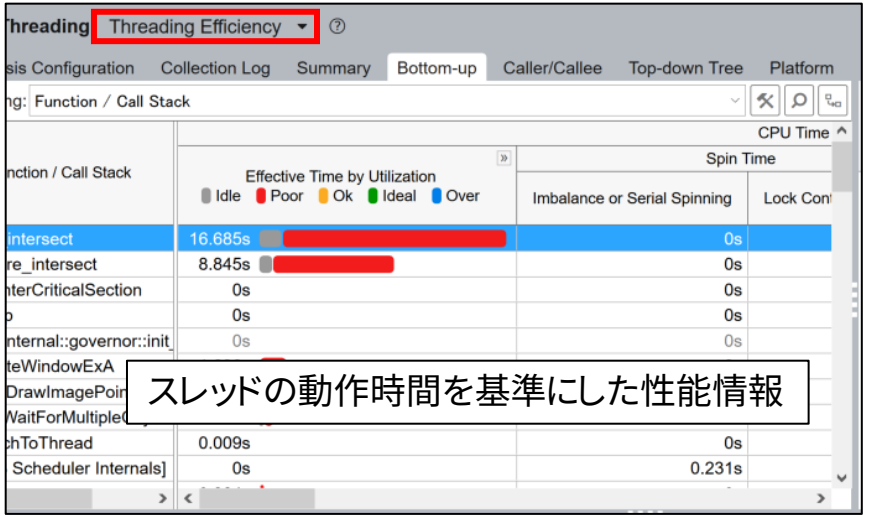


関数名ではなく
ループ処理単位で表示

tips: view の切り替え

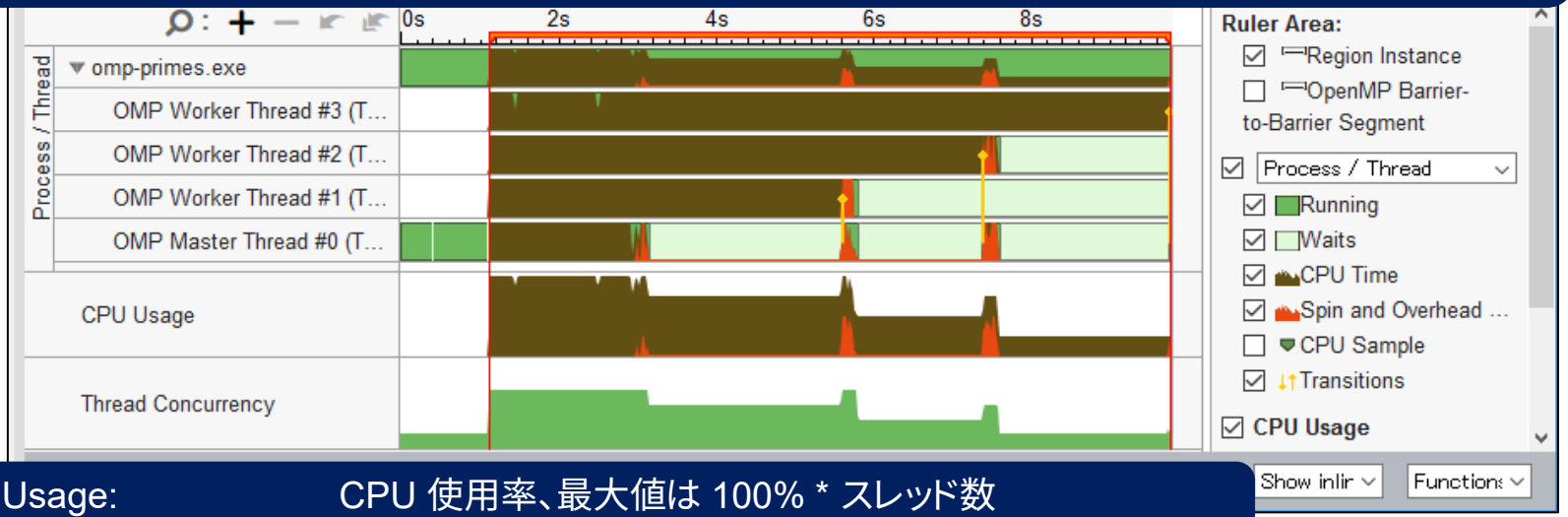
- View は特定の性能情報を基準とした表示に切り替え
  - ✓ 解析タイプによって表示可能な View の種類は異なる





tips: タイムライン

CPU Time (茶): CPU はユーザーコードの実行に費やされた
Spin and Overhead Time (橙): CPU は同期やスレッド API 側の処理に費やされた
Wait Time (白): スレッドは待機している (CPU が処理を行っていない)



CPU Usage: CPU 使用率、最大値は 100% * スレッド数
Thread Concurrency: 同時に動作したスレッド数

分析にあたって

- 解析タイプ毎に収集するデータは異なる
  - ✓ いくつか解析タイプを実行して性能情報を取得する必要があるかもしれません
- パフォーマンスへ影響しやすい性能情報を効率よく収集するために Performance Snapshot 解析タイプを利用ください
- MPI プログラムにおけるランク間通信に関連した性能情報の収集には Application Performance Snapshot (aps)を利用ください

参考: Application Performance Snapshot

- MPI プログラムにおける主要な性能情報を取得するツール

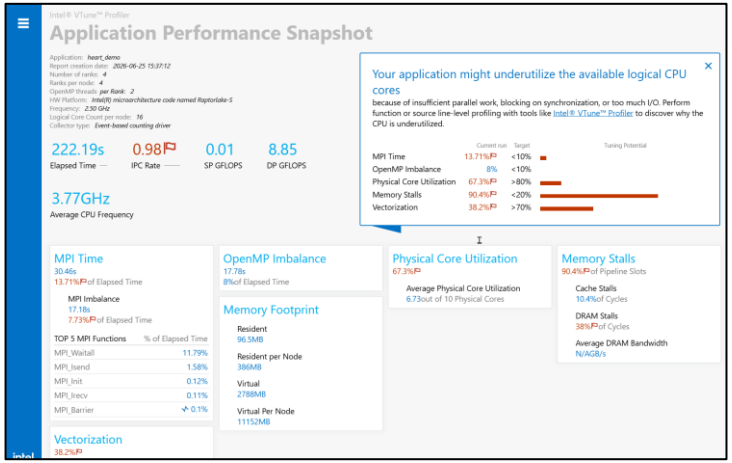
```
> mpirun -np 32 -ppn 16 aps ./run.out
```

```
> aps report ./aps_result_dir
```

- インテル® VTune™ プロファイラーは、スレッド、メモリ、ベクトル化等の効率に関する指標を用いてノード内の性能解析に焦点当てています
- ソースコードレベルに分析には Linaro MAP を利用ください

参考: Application Performance Snapshot

[Application Performance Snapshot User Guide for Linux* OS]

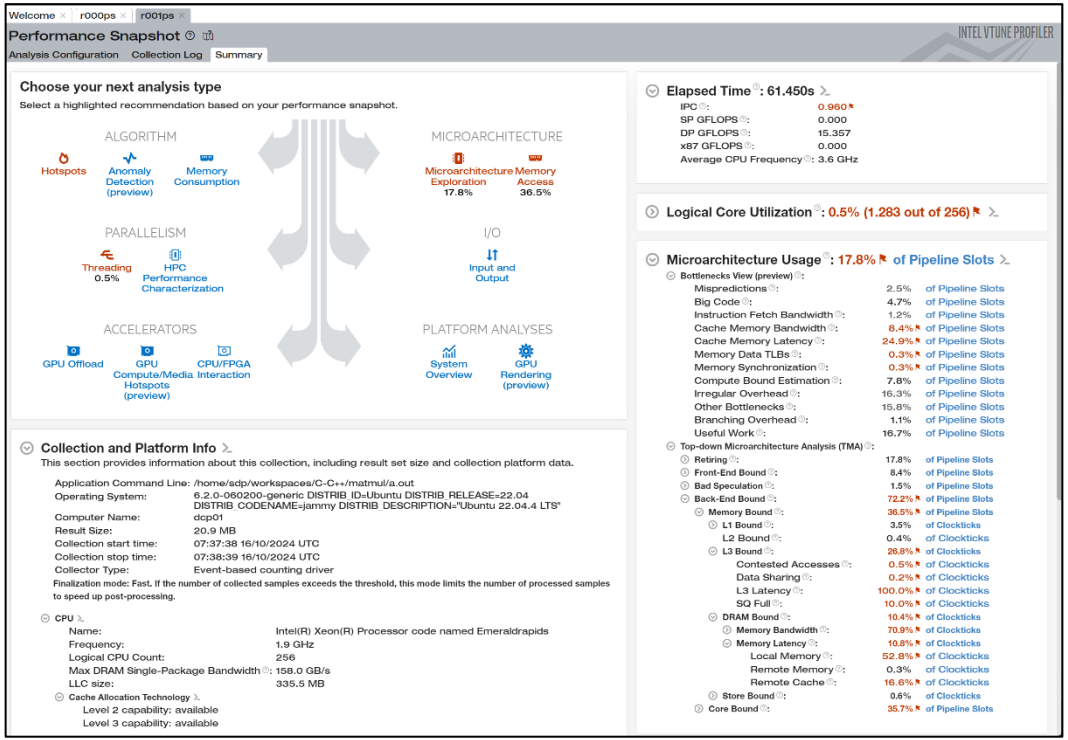


Function summary for all ranks							
Function	Time(sec)	Time(%)	Idle(sec)	Idle(%)	Volume(MB)	Volume(%)	Calls
MPI_Barrier	92.40	36.20	91.79	35.96	0.00	0.00	7788
MPI_Waitall	91.70	35.93	12.53	4.91	0.00	0.00	11733612
MPI_Init	37.57	14.72	0.00	0.00	0.00	0.00	44
MPI_Isend	15.97	6.26	0.00	0.00	50469.13	97.36	18400506
MPI_Finalize	10.92	4.28	0.00	0.00	0.00	0.00	44
MPI_Irecv	6.57	2.58	0.00	0.00	0.00	0.00	18400506
[filtered out 7 lines]							
TOTAL	255.24	100.00	104.39	40.90	51837.71	100.00	48543557

Data Transfers per Rank-to-Rank Communication					
Rank --> Rank	Time(sec)	Time(%)	Volume(MB)	Volume(%)	Transfers
0003 --> 0000	14.30	10.21	8835.73	7.33	370654
0002 --> 0003	14.15	10.10	8835.73	7.33	370622
0003 --> 0002	14.04	10.02	8142.73	6.75	336057
0003 --> 0001	14.02	10.01	7449.73	6.18	301492
0002 --> 0001	13.92	9.93	8142.73	6.75	336057
0002 --> 0000	13.89	9.91	7449.73	6.18	301524
0000 --> 0001	11.61	8.29	19577.18	16.24	847663
0001 --> 0000	11.39	8.13	17498.20	14.51	809130
0001 --> 0002	8.41	6.00	8835.72	7.33	370622
0001 --> 0003	8.14	5.81	7449.73	6.18	301492
0000 --> 0003	8.13	5.80	9528.71	7.90	337679
0000 --> 0002	8.10	5.78	8835.72	7.33	303114
TOTAL	140.11	100.00	120581.65	100.00	4986106
AVG	11.68	8.33	10048.47	8.33	415508

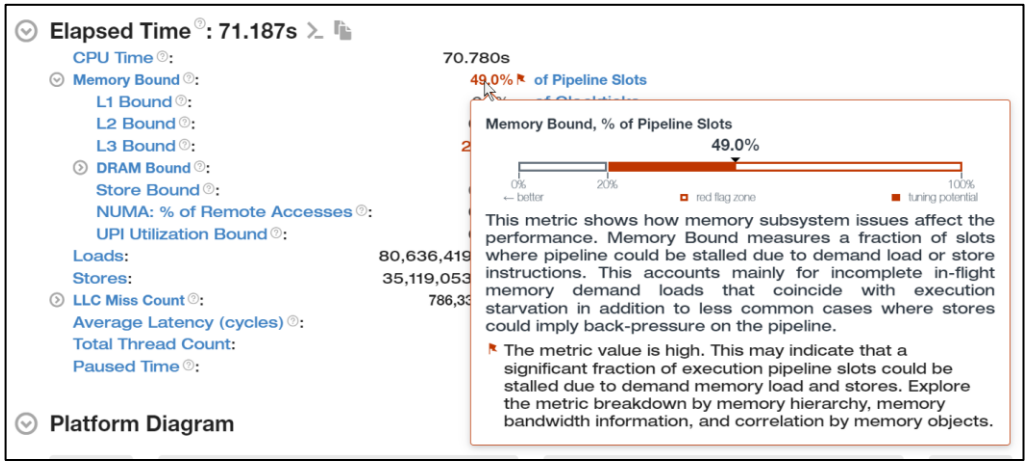
Performance Snapshot 解析タイプ

- 低オーバヘッドかつ、
  少ないデータ収集量で
  性能の概要を表示
  - ✓ パフォーマンスに影響を
    与える主な要素を表示
- より詳細な調査のために
  他の解析タイプを提案



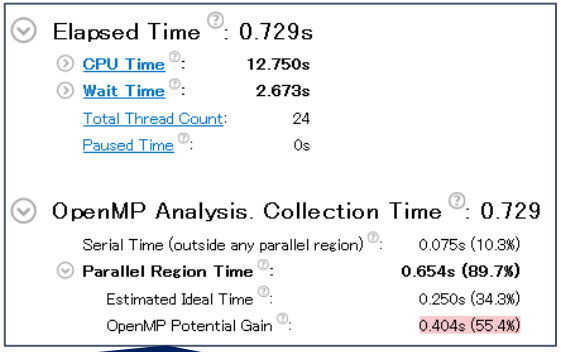
パフォーマンスへの影響

- パフォーマンスに影響がありそうな項目をハイライト
  - ✓ カーソルをあわせるとメトリックの詳細および閾値を確認可能

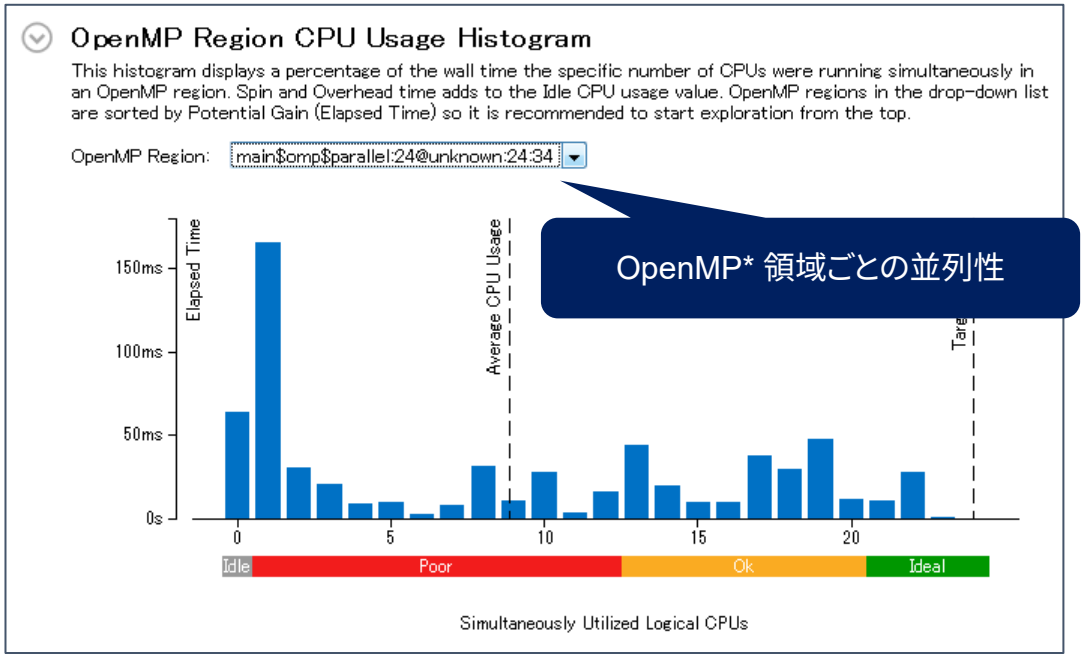


Threading 解析タイプ

- マルチスレッド化された計算処理の効率を調査
- 特に OpenMP* の利用について注目した情報を表示



シリアル/OpenMP* 領域での実行時間



OpenMP* 領域ごとの並列性

Threading 解析タイプ

- OpenMP* 領域における潜在的なパフォーマンスへの影響を表示
  - ✓ 例: ロード･インバランス、スレッド生成によるオーバーヘッド、スケジューリング、同期/ロック
  - ✓ ただしインテル コンパイラーでコンパイルしたプログラムに制限

Grouping を [OpenMP* Region > Function > Call Stack] などに設定

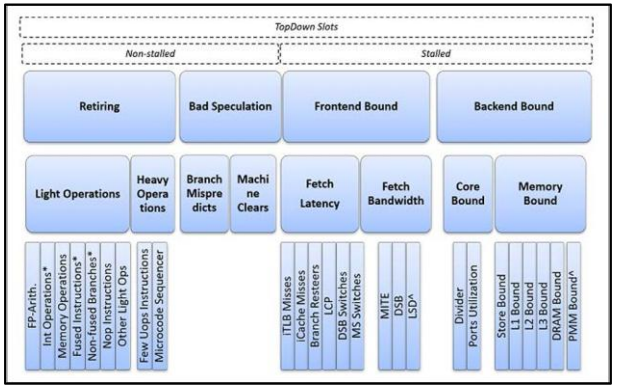
Grouping: OpenMP Region / Function / Call Stack

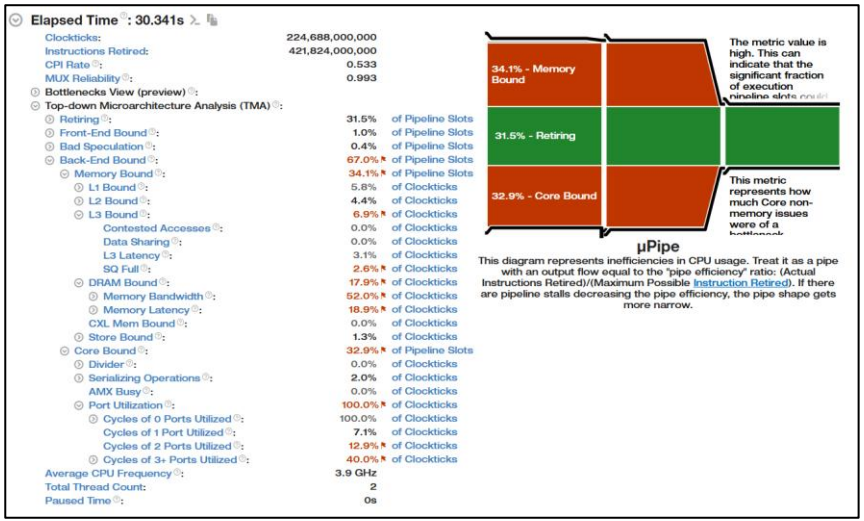
OpenMP Region / Function / Call Stack	OpenMP Potential Gain						Elapsed Time	Number of OpenMP threads	Instance Count	Idle
	Imbalance	Lock Contention	Creation	Scheduling	Reduction	Atomics				
primesompparallel:64@/home/xlsoftkk/workspaces/takeda/ips/omp/prime.cpp:62:62	8.914s	0s	0s	0s	0s	0s	18.873s	64	1	637.1
matmulompparallel:64@/home/xlsoftkk/workspaces/takeda/ips/omp/matmul.cpp:25:34	0.026s	0s	0s	0s	0s	0s	0.055s	64	1	0.4
[Serial - outside parallel regions]							2.451s			1.4
matmulompparallel:64@/home/xlsoftkk/workspaces/takeda/ips/omp/matmul.cpp:33:39							0.368s	64	1	22.4

項目ごとの時間を計上して表示します

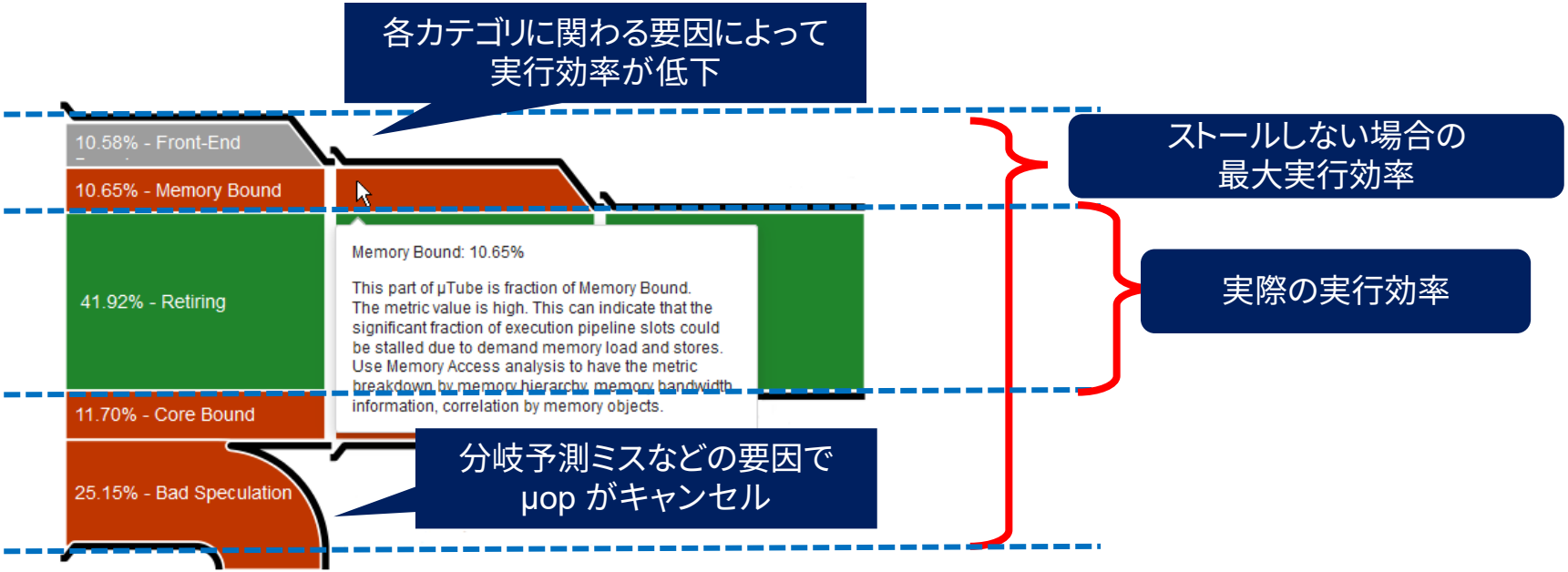
Microarchitecture Exploration 解析タイプ

- CPU 内部の実行状況を視覚化して実行効率を調査
  - ✓ CPU 使用率は高いが期待より動作が遅い場合に有効
  - ✓ Top-down Microarchitecture Analysis (intel.com) にもとづく解析



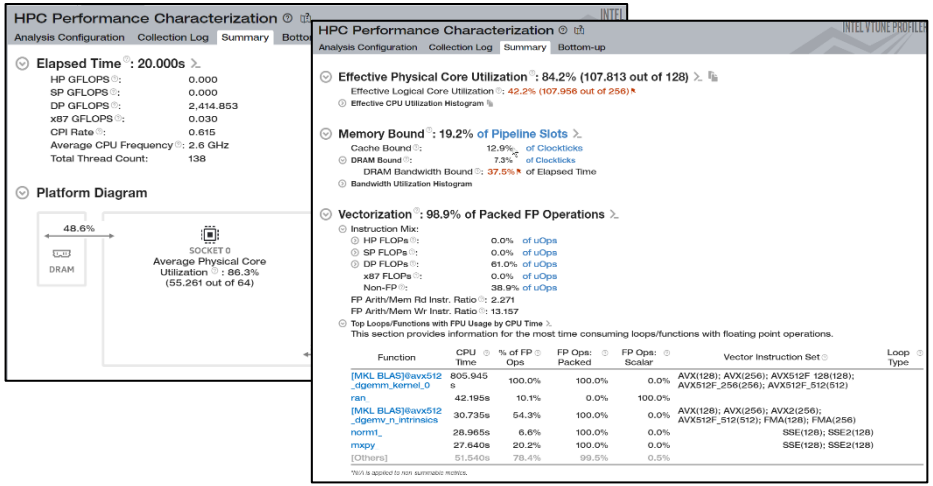


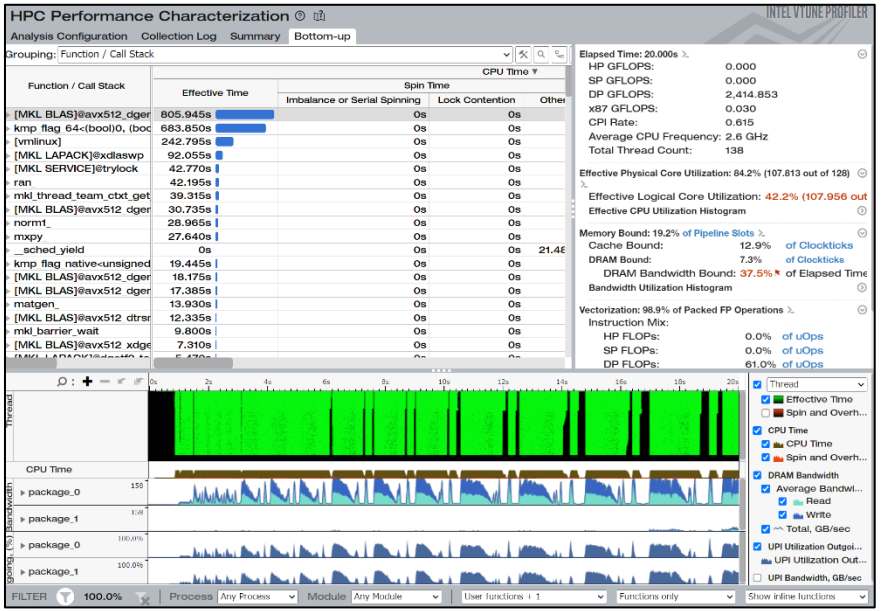
パイプライン ダイアグラム



HPC Performance Characterization 解析タイプ

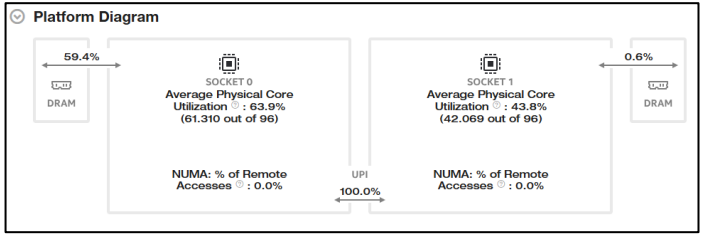
- 計算集約型なプログラムおける性能に影響しやすい項目をまとめて収集
  - ✓ SIMD 命令、メモリー帯域、物理コア利用率、FLOPS

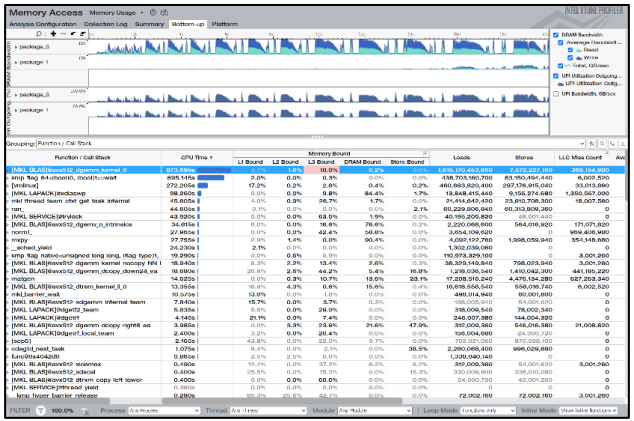


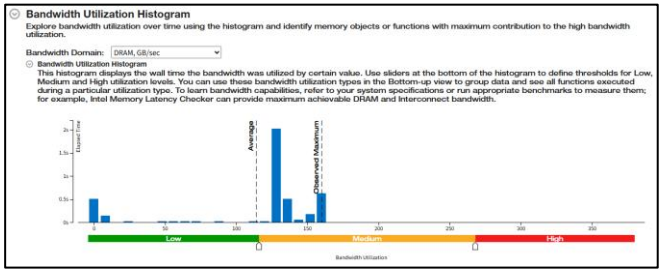


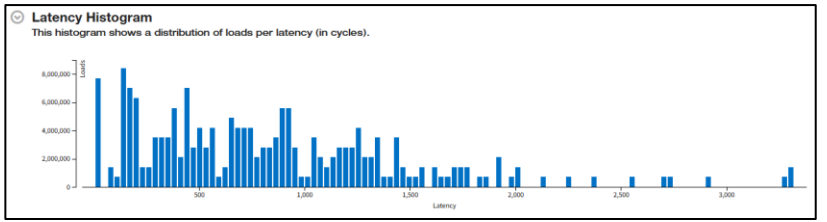
Memory Access 解析タイプ

- キャッシュメモリーの利用効率や DRAM/UPI の帯域幅を調査
  - ✓ NUMA 構成のシステムにおける リモートメモリーアクセスの調査に適しています









まとめ

- 新しいインテル® コンパイラーは LLVM を採用し新規開発されたコンパイラー
  - ✓ それぞれ利用するコマンドが変更
    icc → icx | icpc → icpx | ifort → ifx
  - ✓ 以前のインテル® コンパイラーで実装されていた
    コンパイラー・オプションの多くを引き継ぎつつ、新機能を提供
    その実装や一部機能の利用方法に変更があります
- プログラムの性能情報の収集には
  インテル® VTune™ プロファイラーをご利用ください
  - ✓ MPI プログラム向けには APS が有効です