4. パフォーマンス解析ツールを使用した最適化

Revision 1.2

エクセルソフト株式会社
黒澤 一平

XLSOFT

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
TA/TC	インテル® Trace Analyzer/ Collector
VTune	インテル® VTune Amplifier XE
<Install dir>	各製品のインストールディレクトリー 例:<TA/TC Install dir> = /opt/intel/itac/8.0.3.007/
mpiicc	mpicc のインテル® C コンパイラー版
mpiicpc	mpic++, mpicxx のインテル® C++ コンパイラー版
mpiifort	mpif77,mpif90 のインテル® Fortran コンパイラー版
mpirun mpiexec.hydra	最新バージョンのインテル® MPI は Hydra を使用 (MPD は明示的に設定しないかぎり使用されない)

インテル® VTune Amplifier XE

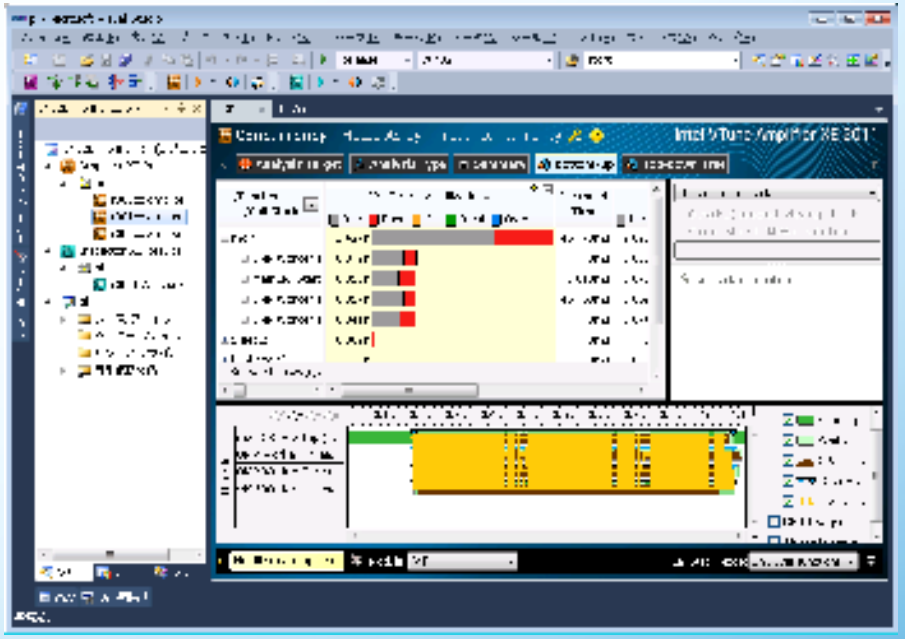
- パフォーマンス向上のための解析ツール
  ソースコードの修正や、特別なビルドなしに該当箇所を特定

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
プロセッサーで生じたイベントを表示

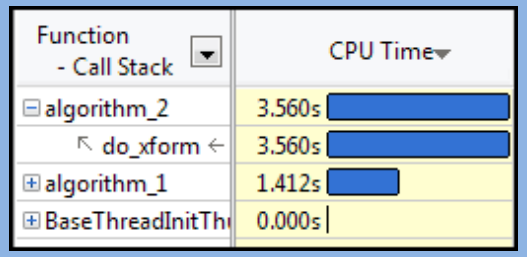


Visual Studio* 2010 に統合した
インテル® VTune Amplifier XE

インテル® VTune Amplifier XEの使用例

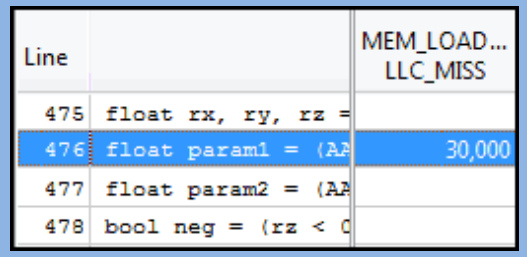
アプリケーションの問題を検出

時間を要している箇所



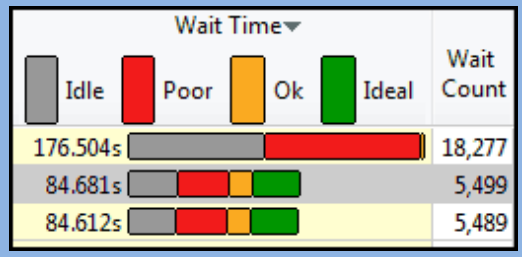
・時間を要している関数を検出
・コールスタックを確認
・ソースレベルでの表示

キャッシュミス



・関数をキャッシュミスの回数でソート
・ソースレベルでキャッシュミスを確認

待機による遅延



・待機時間ごとにロックを確認する
・待機中の CPU 利用率を表示

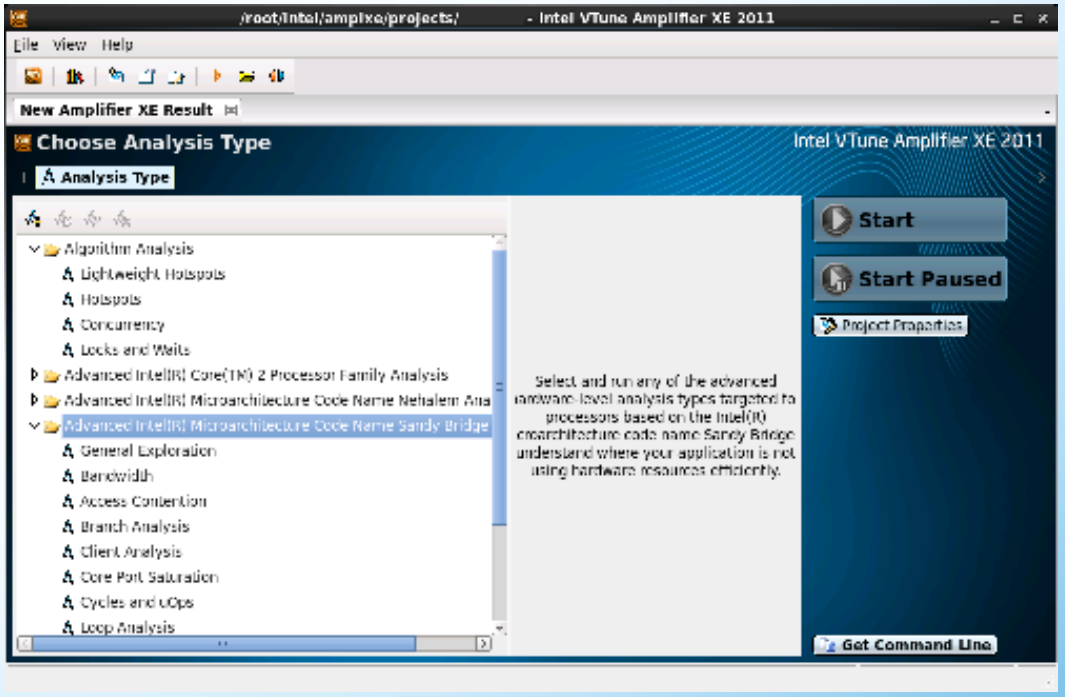
- 低いオーバーヘッド
- 特別なコンパイルや組み込み関数は不要
- GUI とコマンドラインに対応
- ハイブリッド MPI プログラムの解析

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

core2-general-exploration
nehalem-general-exploration
snb-general-exploration



GUI からの指定は、初めての利用者でも使用できるようになっている

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

環境変数を設定する

–x86_64 インテル® VTune Amplifier XE

source <VTune Install dir>/amplxe-vars.sh

バージョン 2011 の場合には

source /opt/intel/vtune_amplifier_xe/amplxe-vars.sh

実行時に VTune を指定する

–シングルノード(ワークステーション)向け解析

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

–クラスター向け解析

mpiexec -n 1024 amplxe-cl -collect *<Analysis Type>* -r *<name>* -- *<program>*

※ <name> は結果を保存するディレクトリー名

※ <Analysis Type は前頁参照>

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

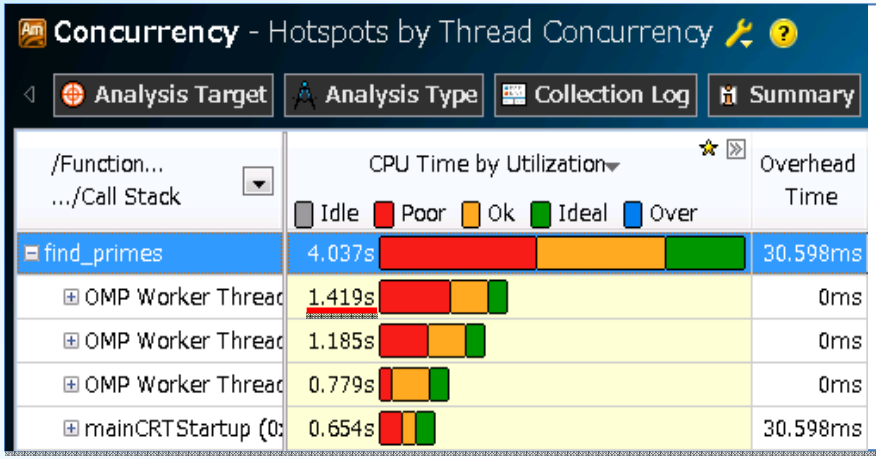
- インテル® VTune Amplifier XE は、一部のサンプリング方式では Performance Monitoring Unit (PMU) からデータを取得する

> PMUを使用する解析タイプ例
> *〈Analysis Type〉* =lightweight-hotspots, or snb-general-exploration

- PMU は 1 プロセッサーに 1 つのみ搭載されている
- 複数起動されたインテル® VTune Amplifier XE から PMU に同時アクセスすることができないため、以下のように 1ノードに対して1プロセスが稼動するように設定して使用する

```
mpiexec -host host001 -n 1 amplxe-cl -collect 〈Analysis Type〉 -r 〈name〉
    -- 〈program〉: -host host001 -n 15 〈program〉:  -host host 002 -n 1
    amplxe-cl -collect 〈Analysis Type〉 -r 〈name〉 -- 〈program〉: -host
    host002 -n 15 …
```

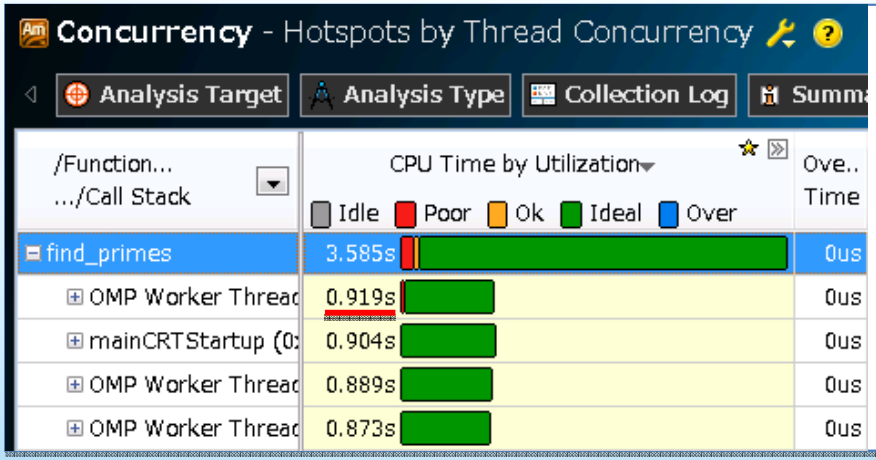
Concurrencyを使用した解析例



並列性 (Concurrency) を解析し、さまざまな表示をすることができる

例では関数単位の並列性を表示したことで、ワークロード(仕事の分配量)が不均一なことが分かった

ワークロードを均一化



ワークロードを均一化した結果、処理時間が短縮した

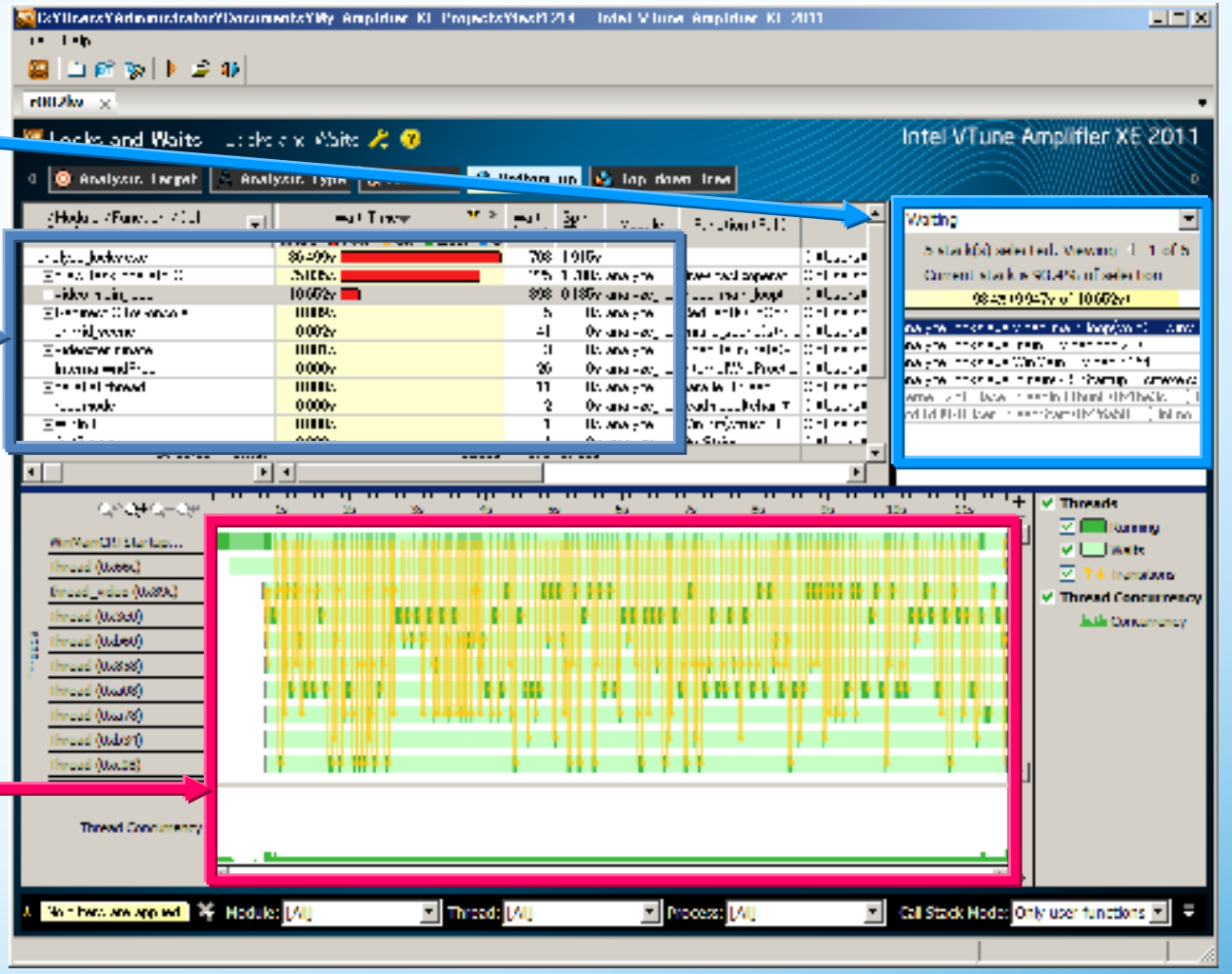
今回の結果では処理に要する時間が 1.42 秒から 0.92 秒に短縮されている

インテル VTune Amplifier XE のインターフェース

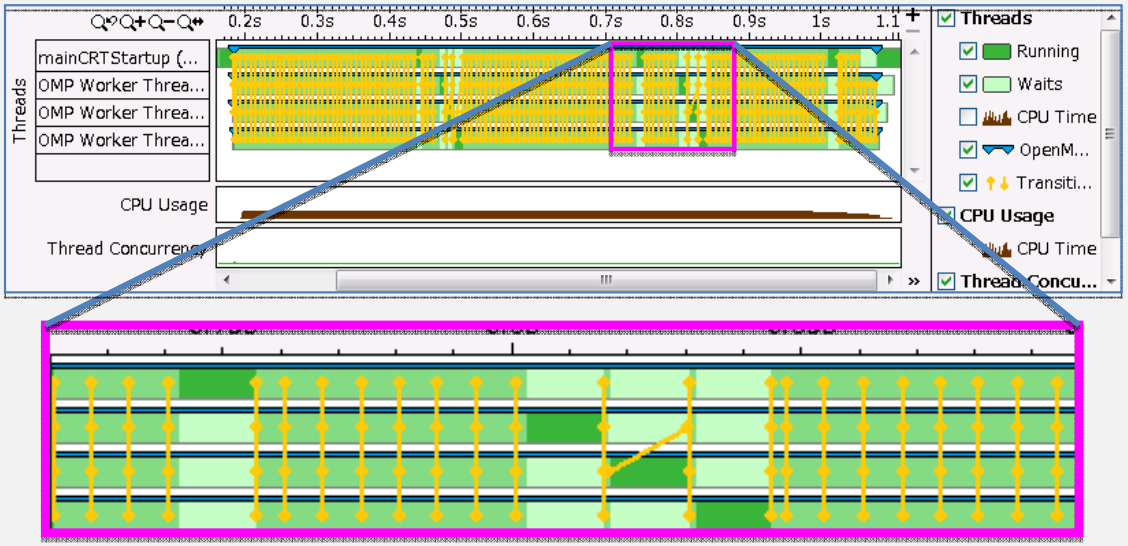
コールスタックの表示

関数-スレッド等の
グラフと帳票表示

各スレッド毎の動作、
通信状況の時系列表示

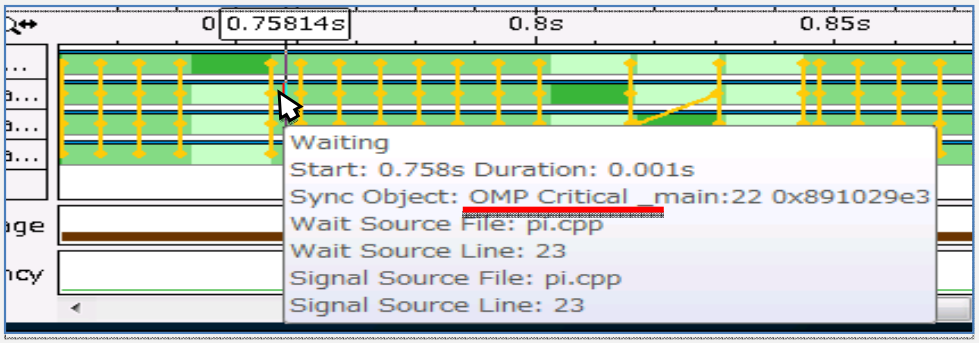


タイムラインビュー



取得したイベントや並列性の情報を時系列表示

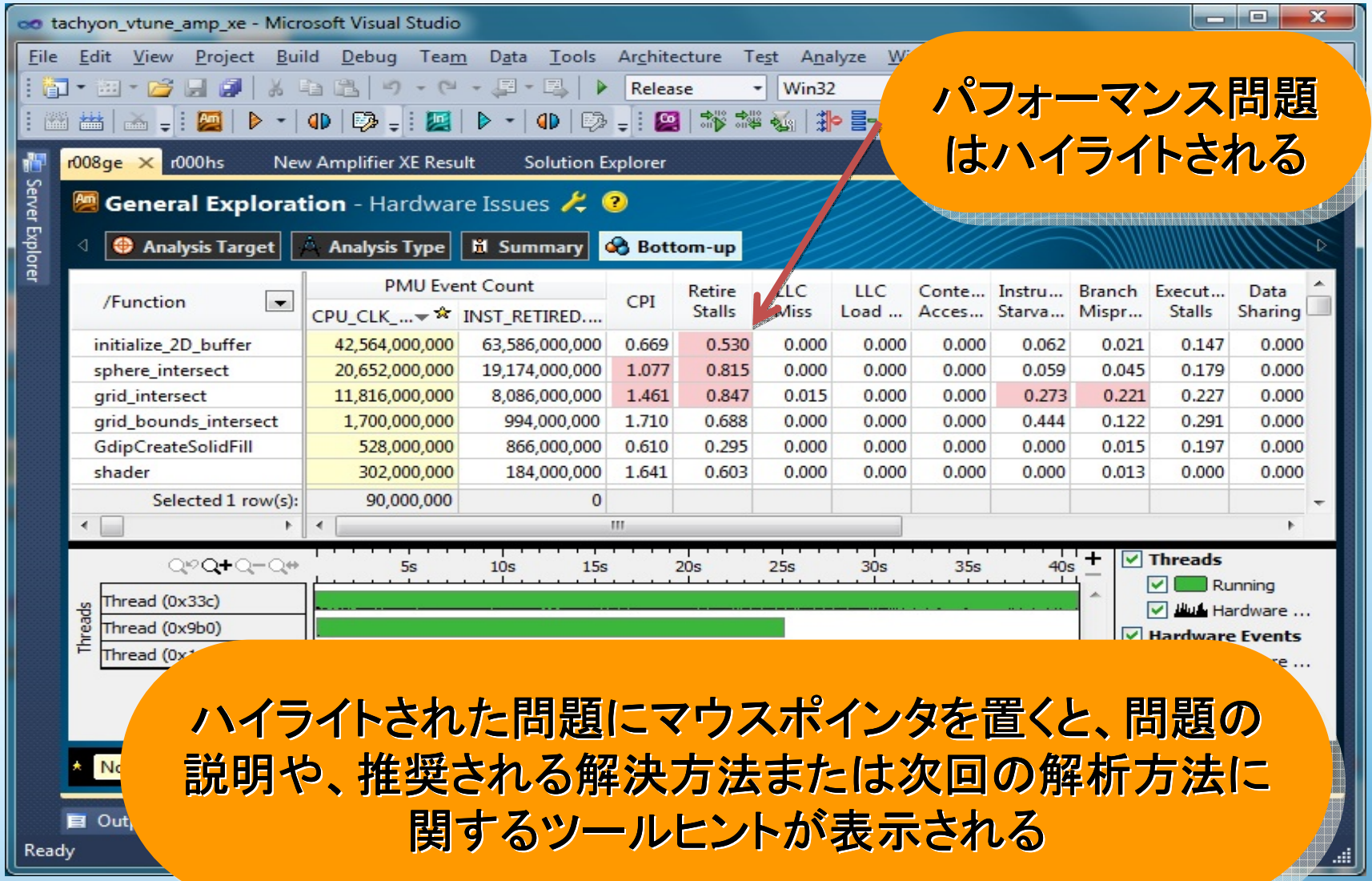
拡大表示することで通信による遅延やスレッドの稼働順番を把握することができる



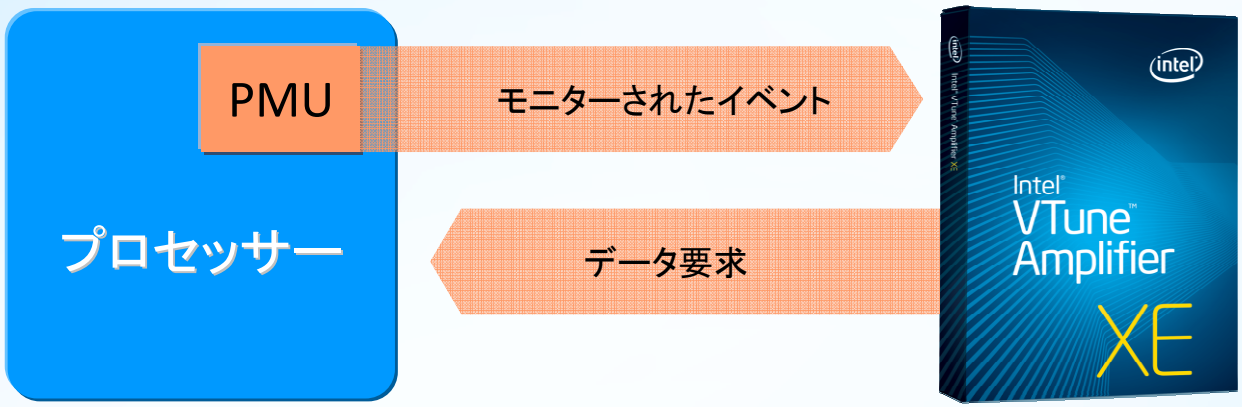
マウスカーソルを合わせることで詳細情報を表示できる

ここではクリティカル・セクションが遅延を招いていることが分かる

snb_general-exploration を使用した解析例 (Sandy Bridge 向けのイベントベースサンプリング)



Performance Monitoring Unit (PMU)



- インテルプロセッサーに実装されたPerformance Monitoring Unit (PMU) に、モニターされたイベントが保存されている。
- VTune は設定したタイミングでPMUからデータを取得することでイベント・ベース・サンプリングを行う。

イベント・ベース・サンプリング

- プロセッサーの各イベントを統計的に解析することで、キャッシュミスや分岐予測ミスなどのパフォーマンス問題を識別することができる。

基本的なイベント例

	実行された命令	キャッシュミス
Core 2	CPU_CLK_UNHALTED.CORE	MEM_LOAD_RETIRED.L2_LINE_MISS
Core i7 2010	CPU_CLK_UNHALTED. THREAD_P	MEM_LOAD_RETIRED. LLC_MISS
Core i7 2011	CPU_CLK_UNHALTED. THREAD_P	MEM_LOAD_UOPS_MISC_RETIRED. LLC_MISS_PS

ロード遅延に関するイベント例

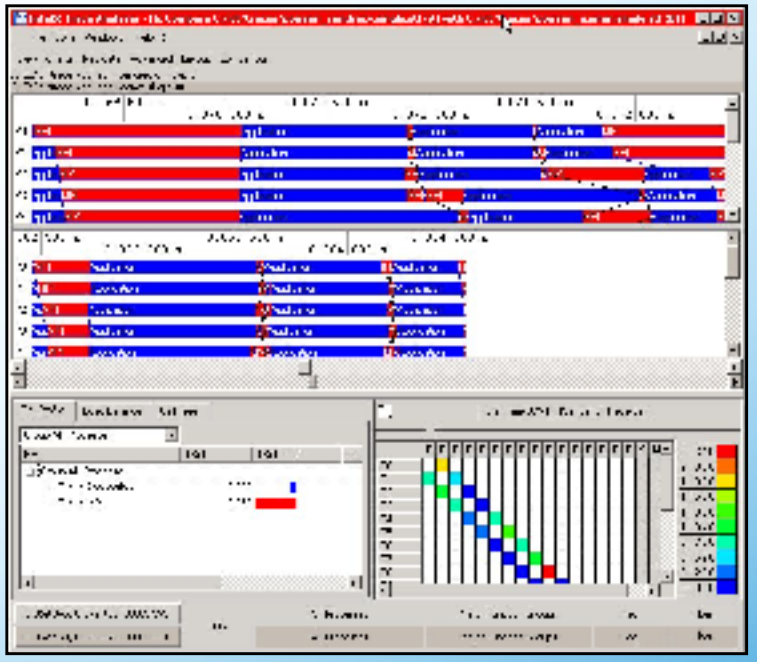
MEM_LOAD_UOPS_RETIRED.L1_HIT_PS
MEM_LOAD_UOPS_RETIRED.L2_HIT_PS
MEM_LOAD_UOPS_RETIRED.L3_HIT_PS
MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT_PS
MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS
MEM_LOAD_UOPS_MISC_RETIRED.LLC_MISS_PS
MEM_LOAD_UOPS_RETIRED.HIT_LFB_PS

前ページのように、イベント名はプロセッサーによって異なる
まずは、各プロセッサー向けにプリセットされた <Analysis Type>を使用することが推奨されている
必要に応じて、より詳細なイベントを追加設定し使用することができる

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



XLSOFT

インテル® Trace Analyzer/Collector の環境変数を設定

環境変数はスクリプトにより自動設定される

- x86_64 インテル® コンパイラー

  source <composer Install dir>/bin/compilervars.sh intel64

  または

  source <icc Install dir>/bin/iccvars.sh intel64

  source <ifort Install dir>/bin/ifortvars.sh intel64

- x86_64インテル® MPI

  source <Intel MPI Install dir>/bin64/mpivars.sh

- インテル® Trace Analyzer/Collector

  source <TA/TC Install dir>/bin/itacvars.sh

Info.
- composer は icc と ifort の両方の環境設定が行なわれる
- Cシェルの場合には .sh の代わりに .csh を使用

インテル® Trace Analyzer/ Collector の動作テスト1

- コンパイル

```
>mpiicc   test.c -o test.exe -trace
>mpiifort test.f -o test.exe -trace
```

- 実行 (stf ファイルの生成)

```
>mpirun -n 8 ./test.exe
>mpiexec -n 8 ./test.exe
```

- インテル Trace Analyzer の起動

```
> <TA/TC Install dir>/bin/traceanalyzer text.exe.stf
```

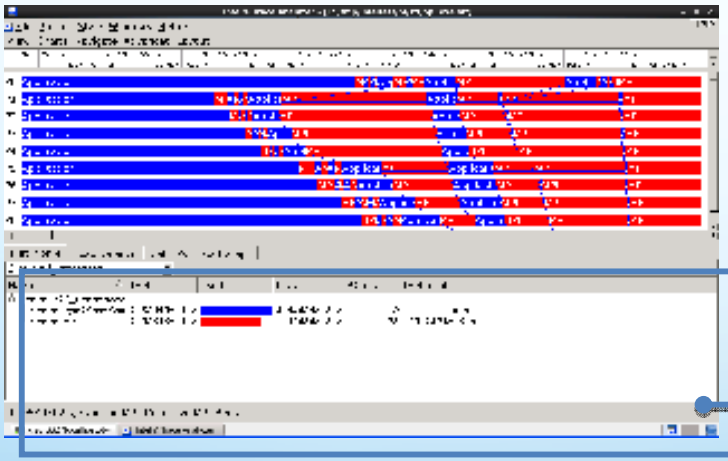
または、起動後に GUI からstfファイルを開く

```
> <TA/TC Install dir>/bin/traceanalyzer
```

(/opt/intel/itac/8.0.3.007/bin/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スドラッグして拡大
- タイムラインの表示を変更することで Function Prorile 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

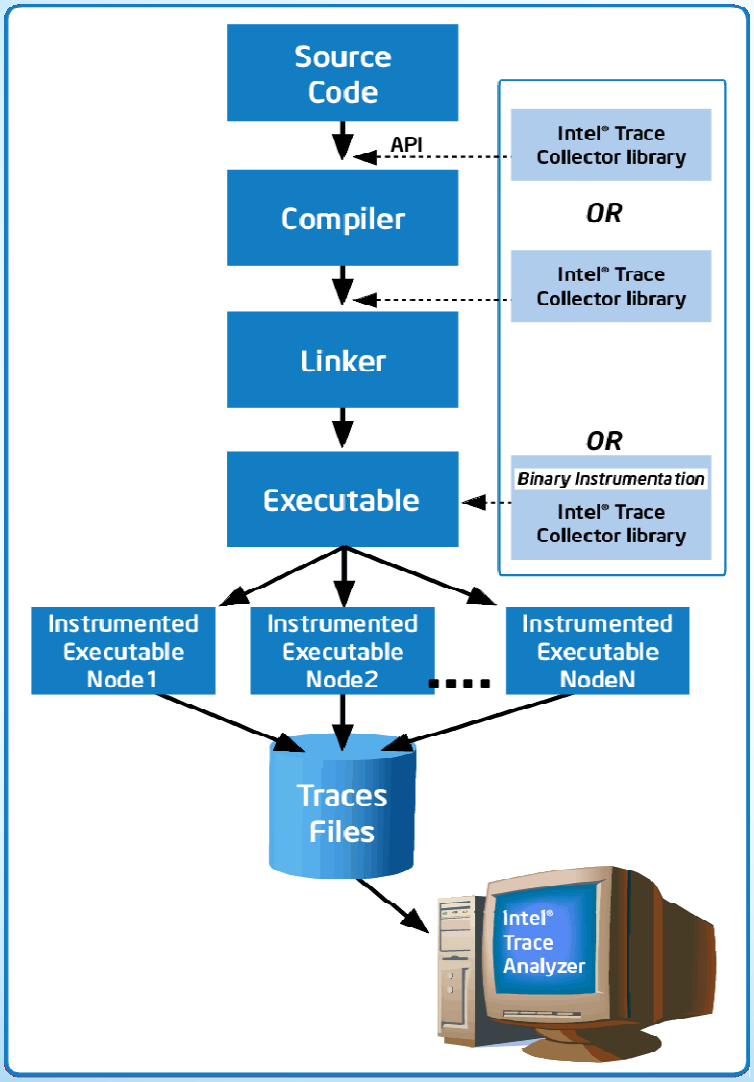


Function Profile

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

インテル® Trace Analyzer とインテル® Trace Collector



TC インテル® Tracer Collector

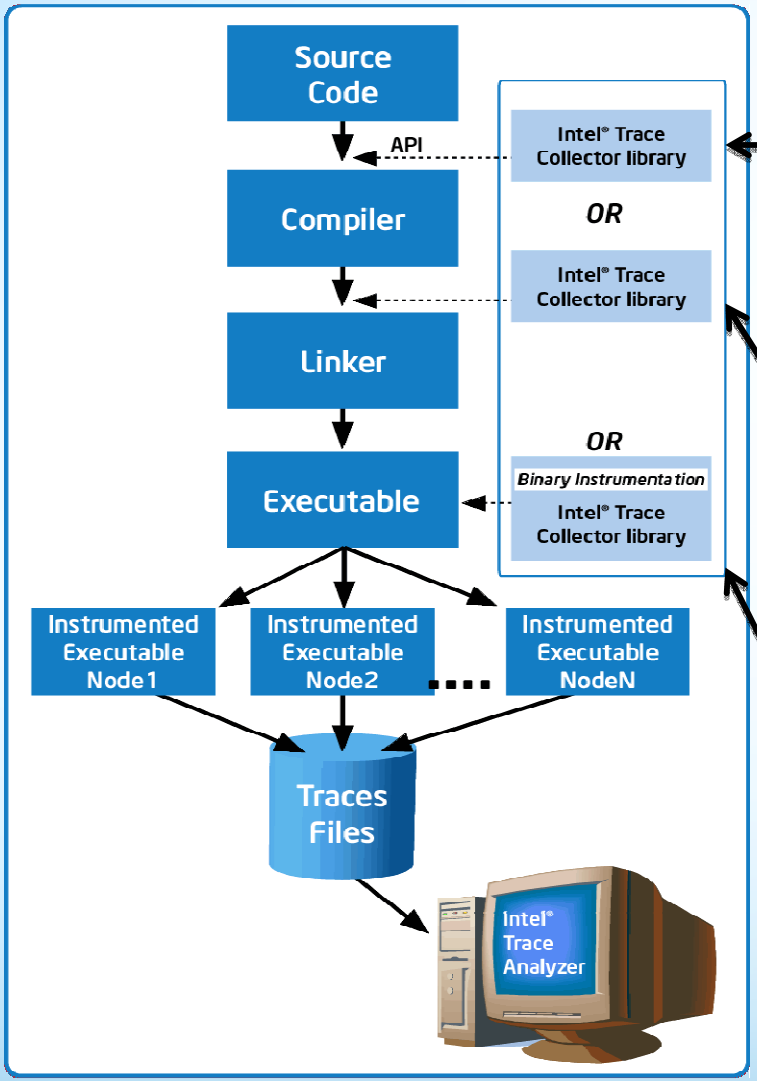
- MPI アプリケーションの解析を主目的とした情報収集ツール
- API、コンパイル時、実行時の任意のタイミングで実装することができ、解析情報は動的に収集される

TA インテル® Tracer 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

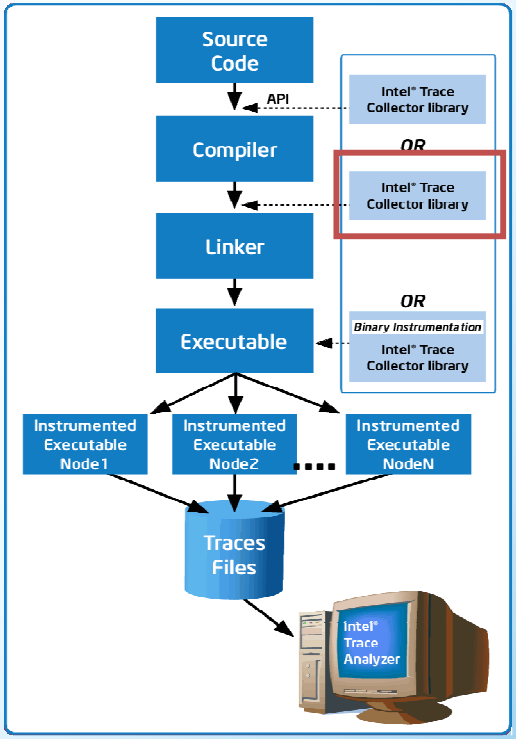
コンパイル時にTCに対応させるオプションを指定することができる。コンパイル時または後から環境変数によって動作を設定することができる。

・実行時

起動時にTCを実装することができる。設定はオプションまたは環境変数を使用することができ、実装は全自動で行われる。

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

- コンパイル フォーマット

```
>mpiicc   sample.c -L${VT_LIB_DIR} -IVT ${VT_ADD_LIBS}
>mpiifort sample.f -L${VT_LIB_DIR} -IVT ${VT_ADD_LIBS}

>mpiicc sample.c -L${VT_LIB_DIR} -IVT ${VT_ADD_LIBS}
                 A               B     C
```

A	-L${VT_LIB_DIR}	TC のスタティックライブラリーパスの指定
B	-IVT	MPIトレース用のライブラリー。他 VTnull、VTfs、VTmc、VTcs 選択可
C	${VT_ADD_LIBS}	他の TC ライブラリーの指定 ※ハイフンは不要

```
-trace = -L${VT_LIB_DIR} -IVT ${VT_ADD_LIBS}
> mpiicc sample.c -trace
```

コンパイル時のインテル® Trace Collector 実装 3

- TCライブラリーの選択

```
>mpiicc sample.c -L${VT_LIB_DIR} -lVT ${VT_ADD_LIBS}
```

-lVT 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
-lVTnull	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
-lVT	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
-lVTfs	libVTfs.so	トレースファイルを随時生成する、Fale-safe 方式
-lVTmc	libVTmc.so	正確性チェックを追加 (エラー検出)
-lVTcs	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

コンパイル時のインテル® Trace Collector 実装 4

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

```
>mpiicc -g sample.c -o sample.exe -trace
>mpirun -genv VT_PCTRACE on -n 8 ./sample.exe
```

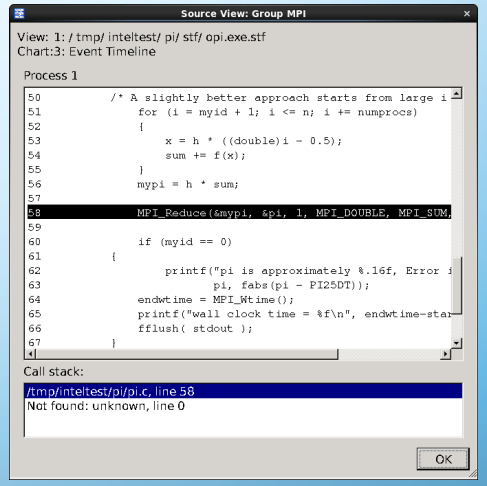
2) TA を起動して、sample.stf を開く

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



実習: MPI アプリケーションの正当性チェック

実習: MPI アプリケーションの正当性チェック 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 -g sample.c -L${VT_LIB_DIR} -lVTmc ${VT_ADD_LIBS}
```

-lVTmc	Message Checking ライブラリーを指定

```
>mpirun -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s -genv
VT_DEADLOCK_WARNING 25s -n 8 ./sample.exe
```

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)

XLSOFT

実習: MPI アプリケーションの正当性チェック 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}

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

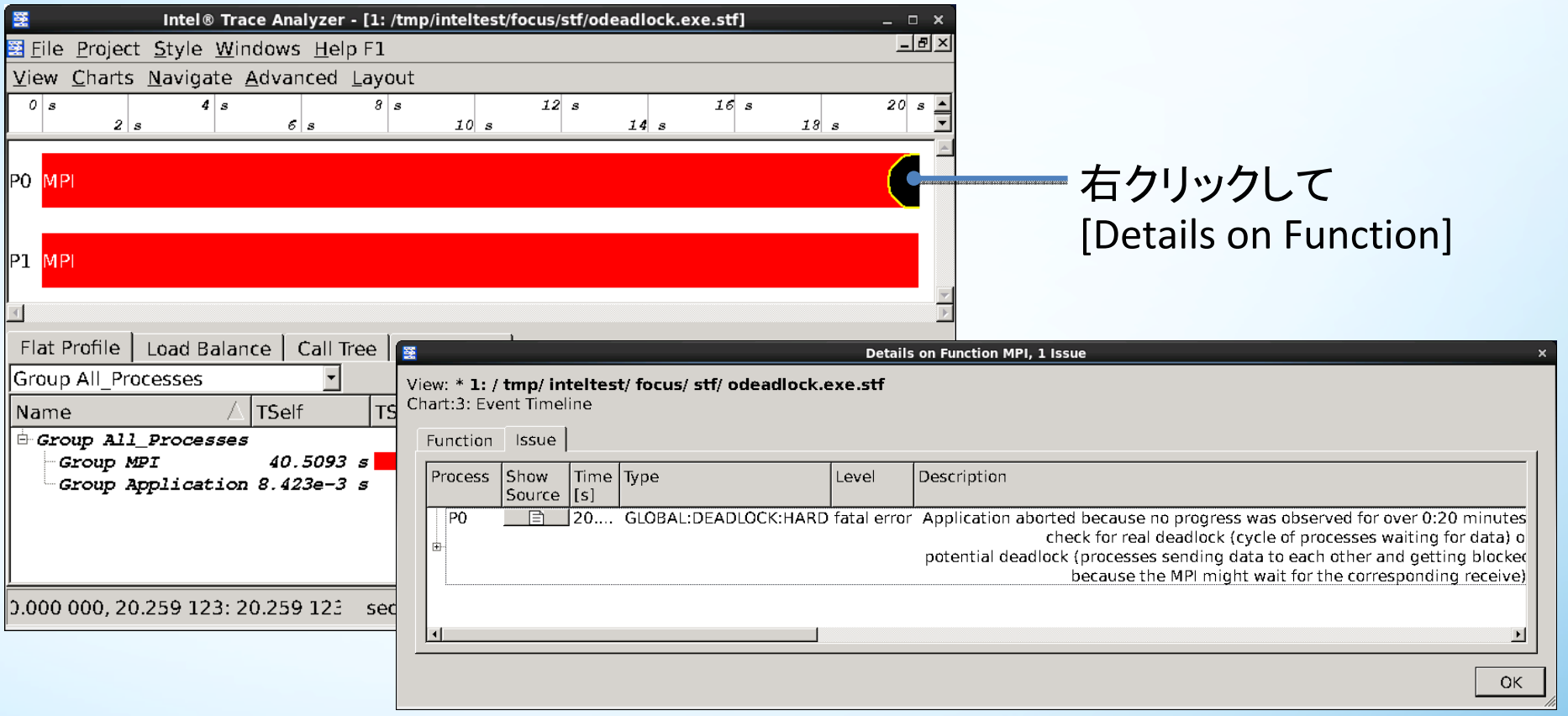
```
[0] ERROR: no progress observed in any process for over 0:20 minutes, aborting application
[0] WARNING: starting emergency trace file writing

[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=0x7fff1cdb2474, 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=0x7fffa4bd4ce4, count=800000, datatype=MPI_INT, source=0,
tag=999, comm=MPI_COMM_WORLD, *status=0x7fffa4ee20e4)
[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

実習: MPI アプリケーションの正当性チェック 3

TA による表示

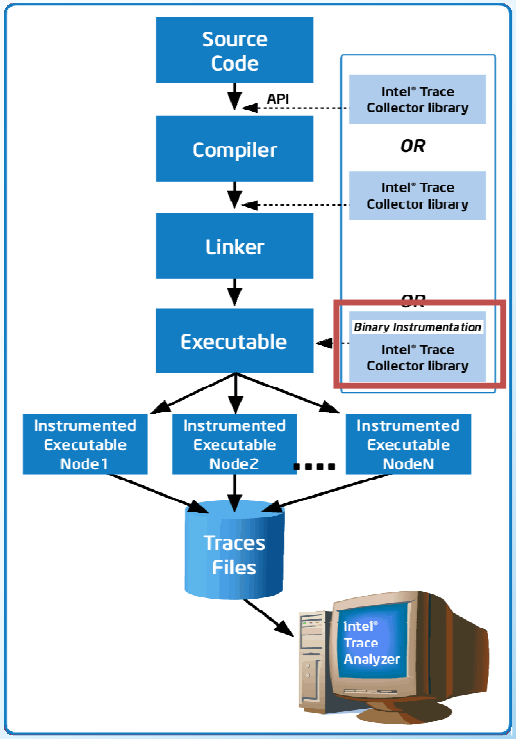


右クリックして
[Details on Function]

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。

実行時の実装: itcpin の利用 1

- 実行時に実装する目的
  - 実行時実装の場合、実行ファイルさえあればトレースファイルを生成することができる
  - コンパイル時は特に設定が不要であり、情報収集の設定を複数変更しながらトレースデータを取得する際などに、コンパイルの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 → 通常時に自動実装されトレースファイルが生成

実行時の実装: itcpin の利用 2

itcpin を利用して、実行時に TC を自動実装する

- 書式

  mpirun <mpi options> itcpin [<TC options>] -- <app command line>

  ※ "--" は TC オプションとアプリケーションコマンドラインを区分けする記号

- 実行例

  >mpiicc sample.c -o sample.exe

  >mpirun -n 10 itcpin --run -- ./sample.exe

  さらに、同じファイルを使用してデッドロックを検出

  >mpiexec -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s -genv VT_DEADLOCK_WARNING 25s -n 8 itcpin --insert libVTmc.so --run -- ./sample.exe

実行時の実装: itcpin の利用 3

- TCオプション例

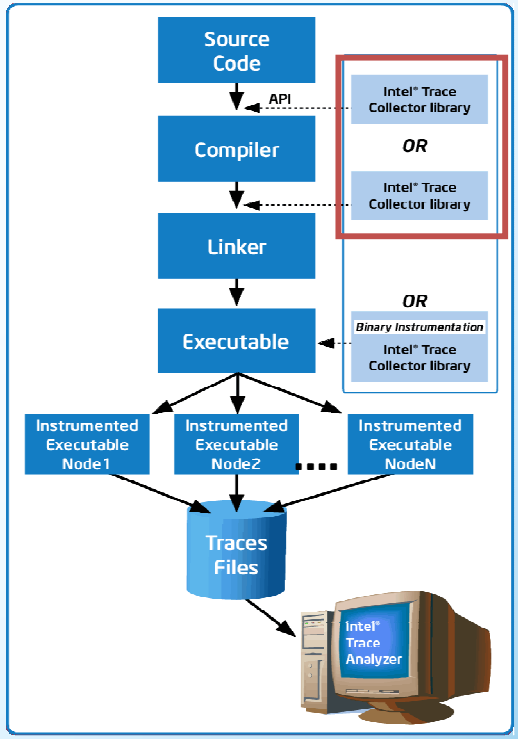
  mpirun <mpi options> itcpin [<TC options>] -- <app command line>

TC オプション	機能
--run	itcpin は指定された実行ファイルを実行しトレースファイルを生成する。このオプションが指定されない場合、実行ファイルを解析して設定可能な機能の一覧を表示する。
--insert	TCのトレース用のライブラリーを選択して、トレース方法を変更する。指定しない場合、libVT.so が適用される 例: --insert libVTfs.so その他 libVTnull.so、libVTmc.so、libVTcs.so を指定可能

TCオプションの詳細と他のオプションは、インテル® Trace Collector のマニュアル、
<TA/TC Install dir>/doc/ITC_Reference_Guide.pdf の
[3.4 Tracing of Binaries and Binary Instrumentation] に記載

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、Intel® Trace Analyzer Reference Guide (ITA_Reference_Guide.pdf) の [1.7 For the Impatient] を実施

- トレース済みのサンプルファイル

  <TA/TC Install dir>/examples/traces/

  poisson_sendrecv.single.stf:　MPI_sendrecv を用いたブロック通信

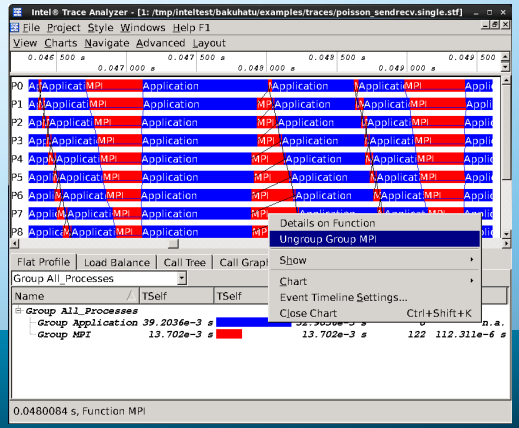
  poisson_icomm.single.stf:　ブロック無し通信への改良版

- 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

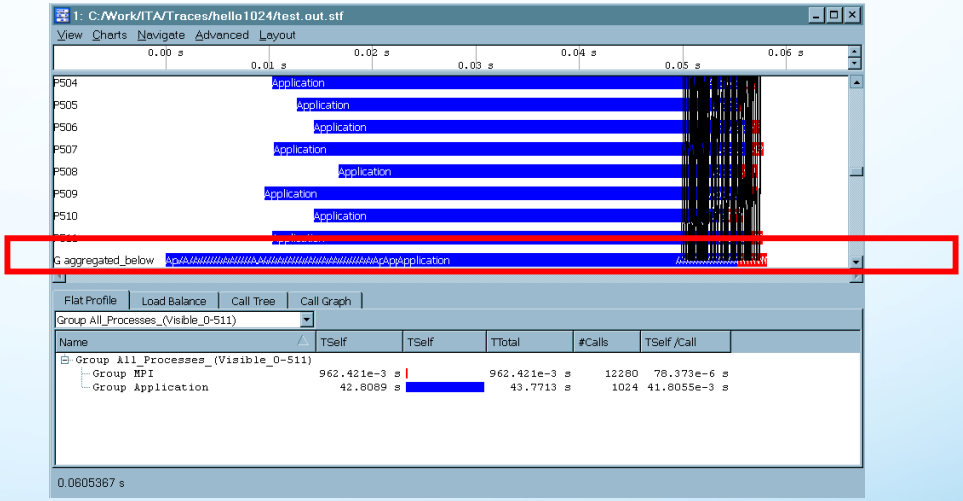
- タイムラインを表示 [Charts]-[Event Timeline]
  - グラフを拡大してアプリケーション (青色部分)とMPI (赤色部分)の関係を確認する
    - MPIタイムの比率が大きすぎないか
    - 各プロセスが均一か

- 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

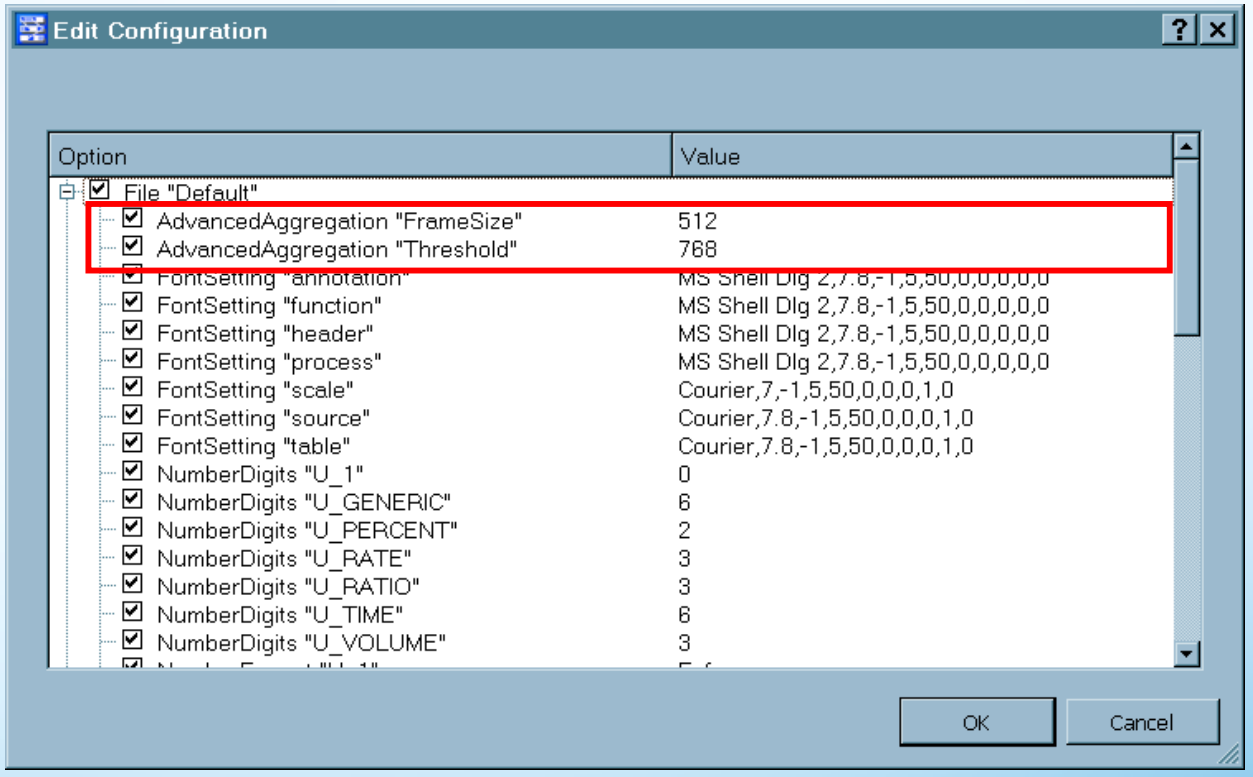
- インテル® Trace Analyzer 8.0.3 より実装された Advanced Aggregation 機能は、大量のデータを特定の事象ごとに集約して表示
  - プロセス数に応じてデータを集約する
  - 1000プロセス以上のデータを扱うことができ、初期値では 768 以上のプロセスは自動的に集約表示される



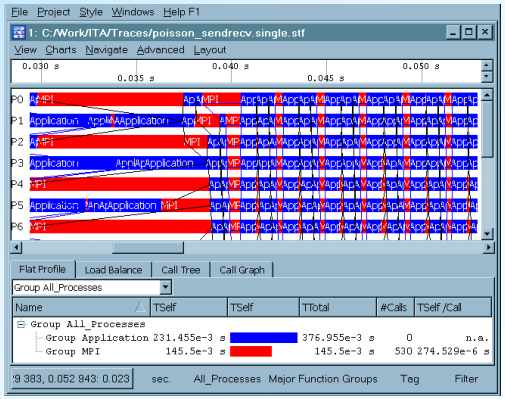
集約されたプロセスは纏められ、
1プロセスのように表示される

Advanced Aggregation: データの集約 2

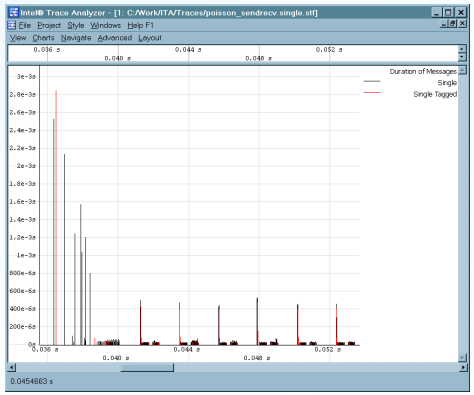
- 自動集約される閾値は 変更可能
  [File]-[Edit Configuration] にて表示されるダイアログで設定



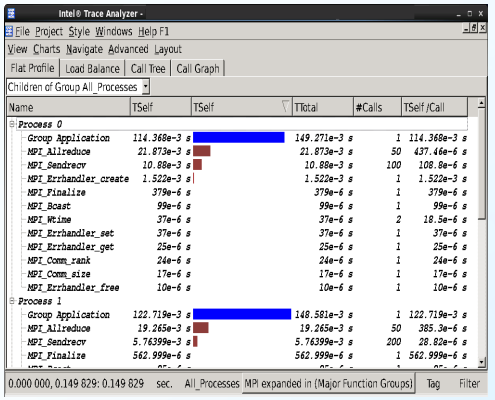
様々な表示、機能



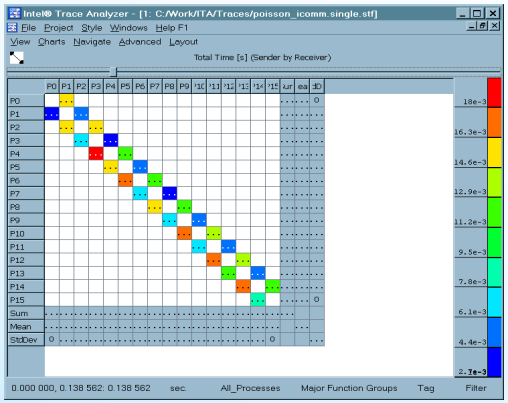
Event Timeline



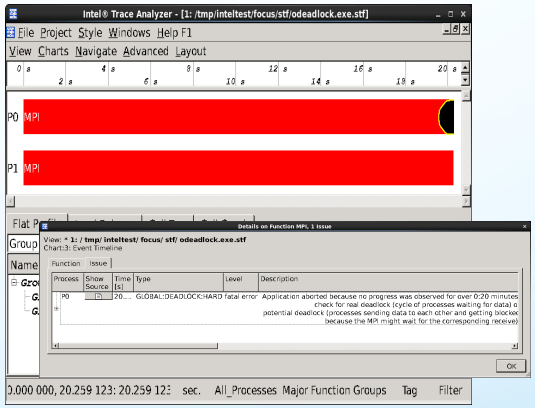
Qualitative Timeline



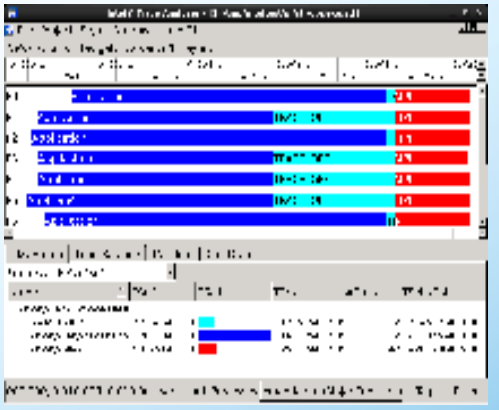
Flat Profile



Message Profile



MPI Error Details



Filtering

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ー向けには、インテル製マイクロプロセッサー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ーに対して、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ー固有の最適化は、インテル製マイクロプロセッサー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ー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

XLSOFT