インテル® VTune™ Amplifier XE で OpenMP* アプリケーションをプロファイルする

並列処理は、ハイパフォーマンス・コンピューティング (HPC) が必要とする性能を提供します。並列処理は、スーパースカラー、ベクトル命令、スレッド化と分散メモリーにおけるメッセージパッシングなどいくつかのレイヤーに渡って実行されます。OpenMP* は、とりわけ HPC 分野で利用されているスレッド抽象化技術です。多くの HPC アプリケーションは、OpenMP* と MPI の両方を使用するハイブリッド型の共有メモリー/分散プログラミング・モデルに移行しつつあります。このドキュメントでは、OpenMP* 並列プログラミング・モデルと OpenMP* ベースのアプリケーションのパフォーマンス・プロファイルに焦点を当てています。インテルが提供する強力なパフォーマンス・プロファイル・ツールであるインテル® VTune™ Amplifier XE は、OpenMP* コードのパフォーマンス上のボトルネックを検出する際に非常に役立ちます。本書では、OpenMP* アプリケーションをプロファイルする手順を示し、このツールで検出できる一般的なパフォーマンスの問題を説明します。

目次

OpenMP* アプリケーションを解析する 2

OpenMP* 並列領域の表現方法 3

シリアル領域 5

オーバーヘッドとスピン時間 6

シナリオ 1: バランスの取れた並列領域 7

シナリオ 2: バランスの取れていない並列領域 8

シナリオ 3: 粒度の問題 9

シナリオ 4: 同期オブジェクトと待機時間 10

まとめ 11

OpenMP* アプリケーションを解析する

並列領域の検出を含む OpenMP* アプリケーションのプロファイルには、インテル® VTune™ Amplifier XE 2013 Update 4 以降が最適です。最良の結果を得るため、インテル® Composer XE 2013 Update 2 以降でアプリケーションをビルドしてください。(ただし、インテル® Composer XE 2013 Update 5 は、OpenMP* 並列領域の検出に問題があるため使用しないでください。インテルの OpenMP* ランタイムの古いバージョンや、GCC* および Microsoft* の OpenMP* ランタイムをプロファイルすることはできますが、並列領域を検出できません。)

インテル® Composer XE 2013 SP1 で提供される OpenMP* ランタイムよりも古いバージョンを使用する場合は、**KMP_FORKJOIN_FRAMES** 環境変数を 1 に設定して必要な機能を有効にします。これは、[Project Properties] ダイアログボックスの [User-defined Environment Variables] で行います。
注: デフォルトでは、インテル® コンパイラーは OpenMP 領域名にソースファイル名を追加しません。領域名にソースファイル名を追加するには、**-parallel-source-info=2** オプションを指定してコンパイルします。

OpenMP* アプリケーションを解析するには、[Advanced Hotspots] などの一般的な解析を実行します (図 1 を参照)。

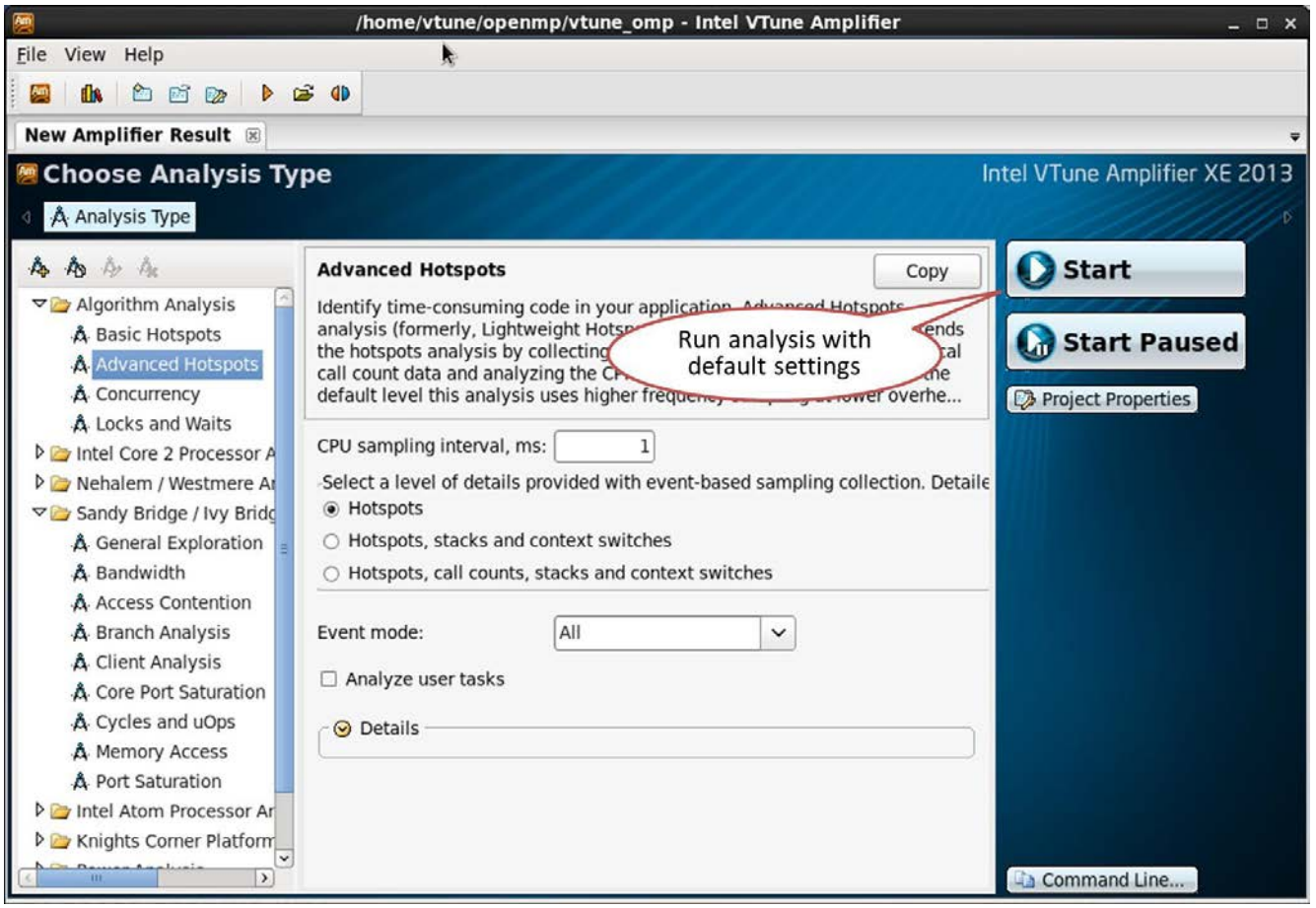

図 1. インテル® VTune™ Amplifier XE 解析の実行

OpenMP* 並列領域の表示

OpenMP* 並列領域は、インテル® VTune™ Amplifier XE でフレームドメインとして表示されます。グローバルフレームは、アプリケーションの経過時間のオーバーラップしない領域を表し、特定のスレッドに関連付けられません。

インテルの OpenMP* ランタイムを使用すると、並列領域を検出できます。並列領域を表示するには、[Bottom-up] ペインに切り替えて、[Frame Domain /Frame Type ...] でグループ化します (図 2 を参照)。

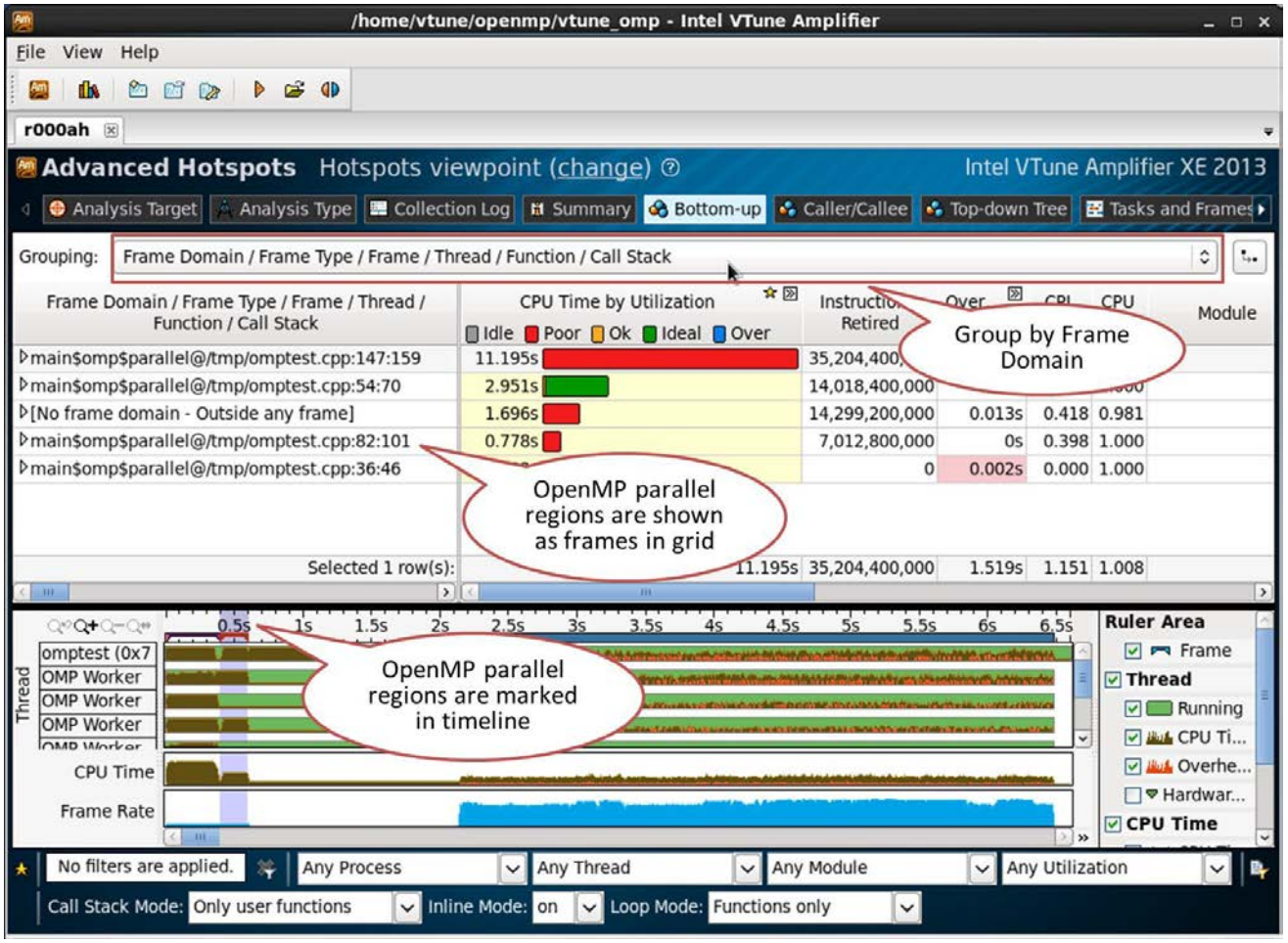

図 2. [Bottom-up] ペイン

[Tasks and Frames] ペインでも、OpenMP* 並列領域と対応するスレッドのアクティビティーを確認できます (図 3 を参照)。

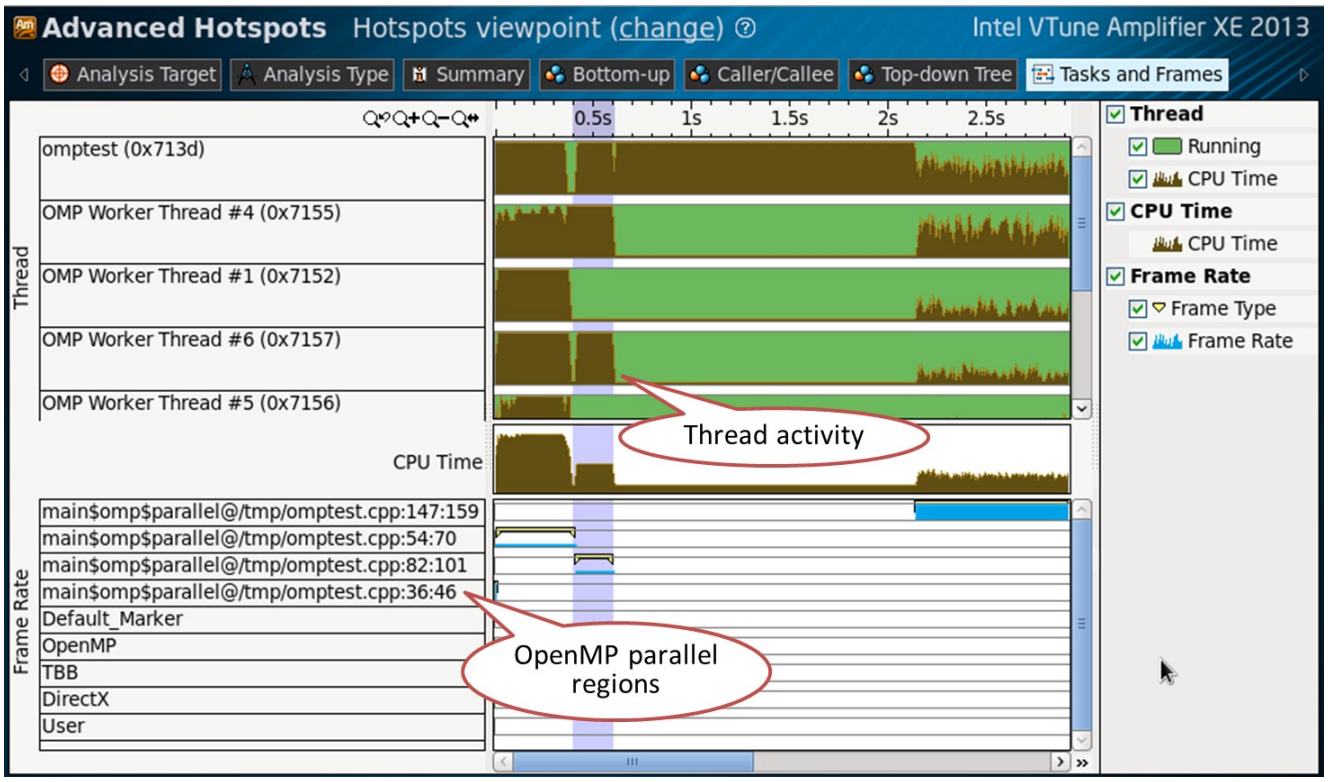

図 3. [Tasks and Frames] ペイン

各 OpenMP* 並列領域は、個別のフレームドメインとして表示され、ソースファイルと行番号の範囲によって識別されます。フレームドメインはコード位置を表し、その領域に対する各呼び出しがフレームになります。フレームは、ある fork ポイントから対応する join ポイントまでの時間を表します。フレームの数は、スレッドの数やチャンクサイズに依存しません。

次の簡単な疑似コードには、2 つの OpenMP* 並列領域が含まれています。インテル® VTune™ Amplifier XE のプロファイルではそれぞれがフレームドメインとなるため、2 つのフレームドメインが表示されます。

```
int main()
{
   #pragma omp parallel for    // フレームドメイン 1、フレームの数: 1
   for (int i=0; i < NUM_ITERATIONS; i++)
   {
      do_work();
   }

   for (int j=0; j<4; j++)
   {
      #pragma omp parallel for // フレームドメイン 2、フレームの数: 4
      for (int i=0; i < NUM_ITERATIONS; i++)
      {
          do_work();
      }
   }
}
```

1 つ目のフレームドメインは 1 回だけ呼び出され、fork と join が 1 回ずつ発生します。そのため、ループ本体が 16 スレッドによって並列実行される場合であっても、フレームドメイン 1 は 1 フレームだけになります。2 つ目の並列領域 (フレームドメイン 2) は、ループの各反復で呼び出され、各呼び出しで対応する fork と join が発生します。そのため、インテル® VTune™ Amplifier XE のプロファイルでフレームドメイン 2 には 4 つのフレームが表示されます。

シリアル領域

OpenMP* 並列領域外で費やされたすべての時間は、[No frame domain – Outside any frame] という名前の領域にまとめられます。この情報を基にコードのシリアル領域を予測できます (図 4 を参照)。

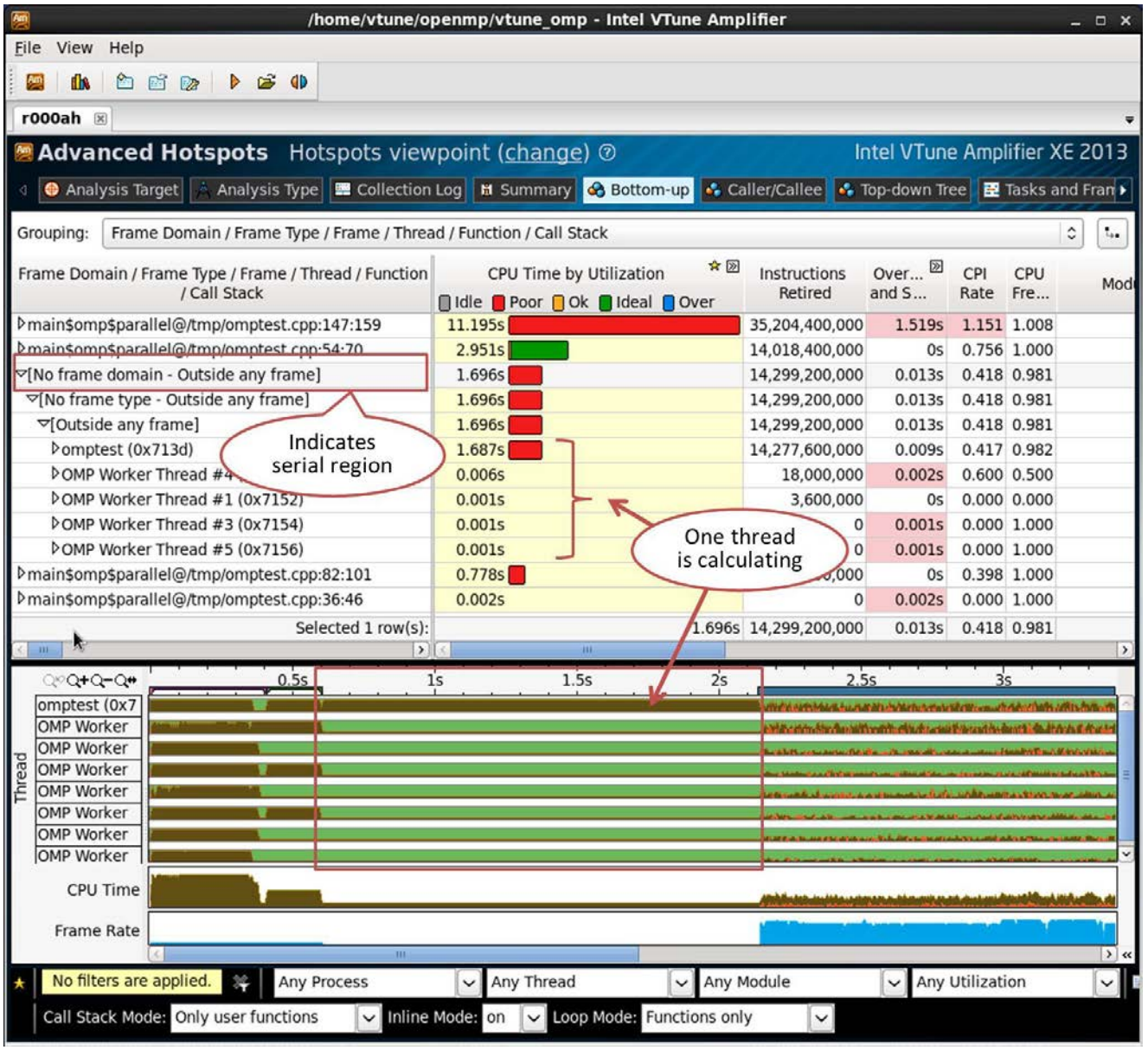

図 4. シリアル時間

オーバーヘッドとスピン時間

OpenMP* において**オーバーヘッド時間**とは、スレッド管理、ワーク分割、スケジュール、同期などに関連するランタイムの内部処理に費やされた時間を指します。ライブラリーの内部処理に費やされた時間が含まれ、有効な計算に費やされた時間は含まれません。**スピン時間**とは、 CPU がビジー状態だった間の待機時間です。通常、ソフトウェア・スレッドが待機中に、同期呼び出しによって CPU がポーリングされると発生します。つまり、同期ポイントで待機中にスピンすることから "スピン時間" と呼ばれます。OpenMP* スレッドは、OpenMP* barrier などのライブラリー内部の同期オブジェクトで "スピン" する可能性があります。

オーバーヘッドとスピン時間は、CPU 使用率に関連する関数名やコールスタックを確認して検出されます。一部の OpenMP* 内部関数はスレッドやタスクの管理などを行うため、それらの関数で費やされた時間はオーバーヘッドと見なされます。スピン時間も同様の方法で検出されます。また、一部の OpenMP* 内部関数はアクティブ待機中 (スピン待機中) に OpenMP* ランタイムによって実行されます。それらの関数で費やされた時間はスピン時間と見なされます (図 5 を参照)。オーバーヘッドとスピン時間は、インテル、GCC*、および Microsoft* の OpenMP* ランタイム、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) ランタイム、インテル® Cilk™ Plus ランタイムで検出可能です。

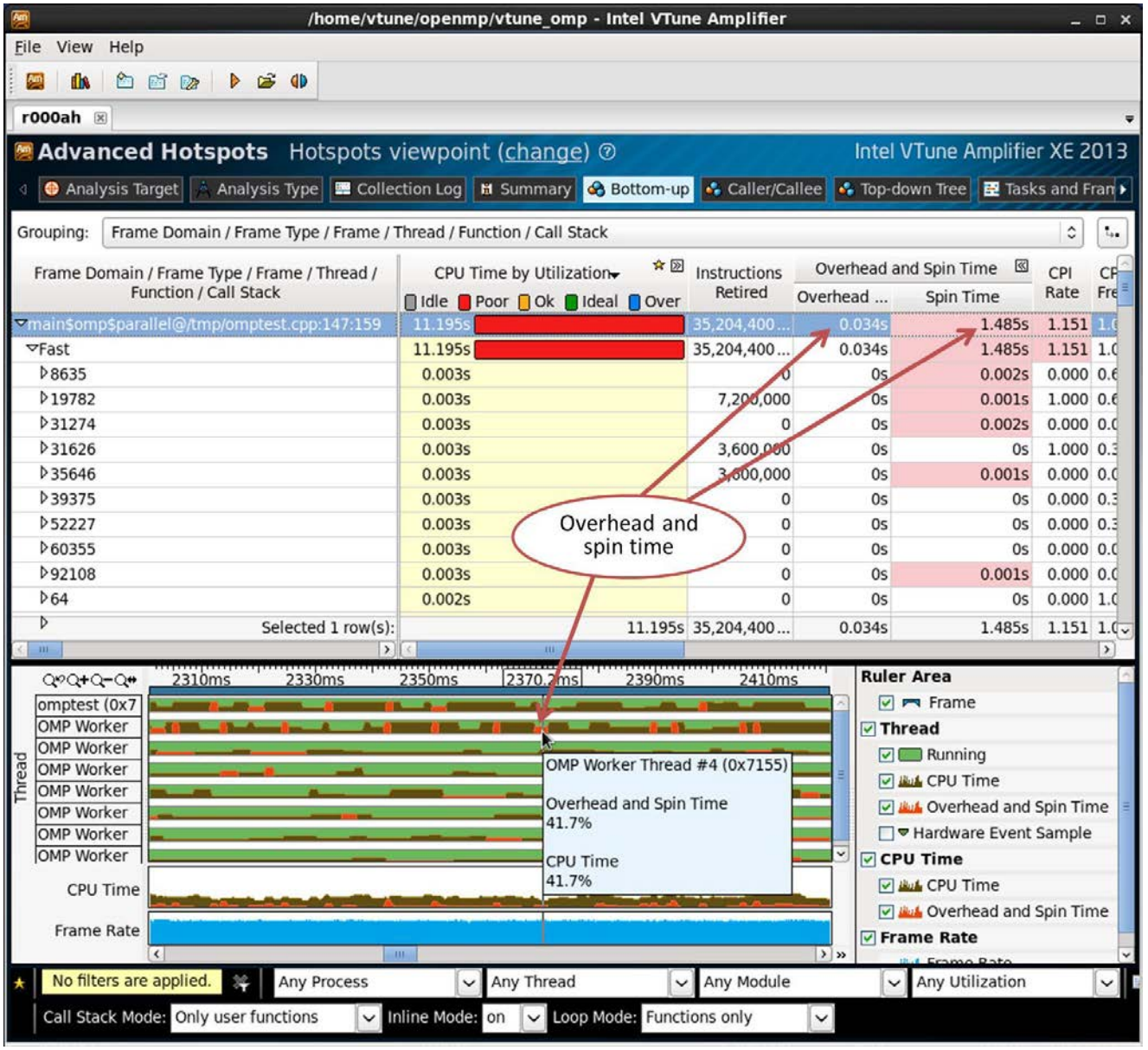

図 5. オーバーヘッドとスピン時間

シナリオ 1: バランスの取れた並列領域

サンプル・アプリケーションでは、omptest.cpp ファイルの 54 行目にある並列領域が良い例として挙げられます。下記の図 6 では、[Bottom-up] ペインで [Frame Domain/Frame Type/Frame/Thread/Function/Call Stack] でグループ化されており、フレームドメインにはフレームが 1 つだけあります。これは、この並列領域が 1 回だけ呼び出されたことを示します。データをスレッドレベルまで展開すると、8 つのスレッドが表示されます。テストに使用されたシステムは 4 コア 8 スレッドなので、これは良い結果と言えるでしょう。CPU 使用率が高く (CPU 時間を示すバーが緑色であることが分かります)、並列領域では 8 つのスレッドがすべてビジー状態で、ほぼ同量の計算を行っています。タイムラインでも、並列領域で 8 つのスレッドの CPU 使用率が高いことが確認できます (薄紫色でハイライトされている個所)。しかし、全く問題がないとは言い切れません。SIMD 命令を活用していなかったり、マイクロアーキテクチャー・レベルの問題が存在する可能性があります。この結果から分かることは、スレッド並列処理とロードバランスに関して OpenMP* の問題はないということです。

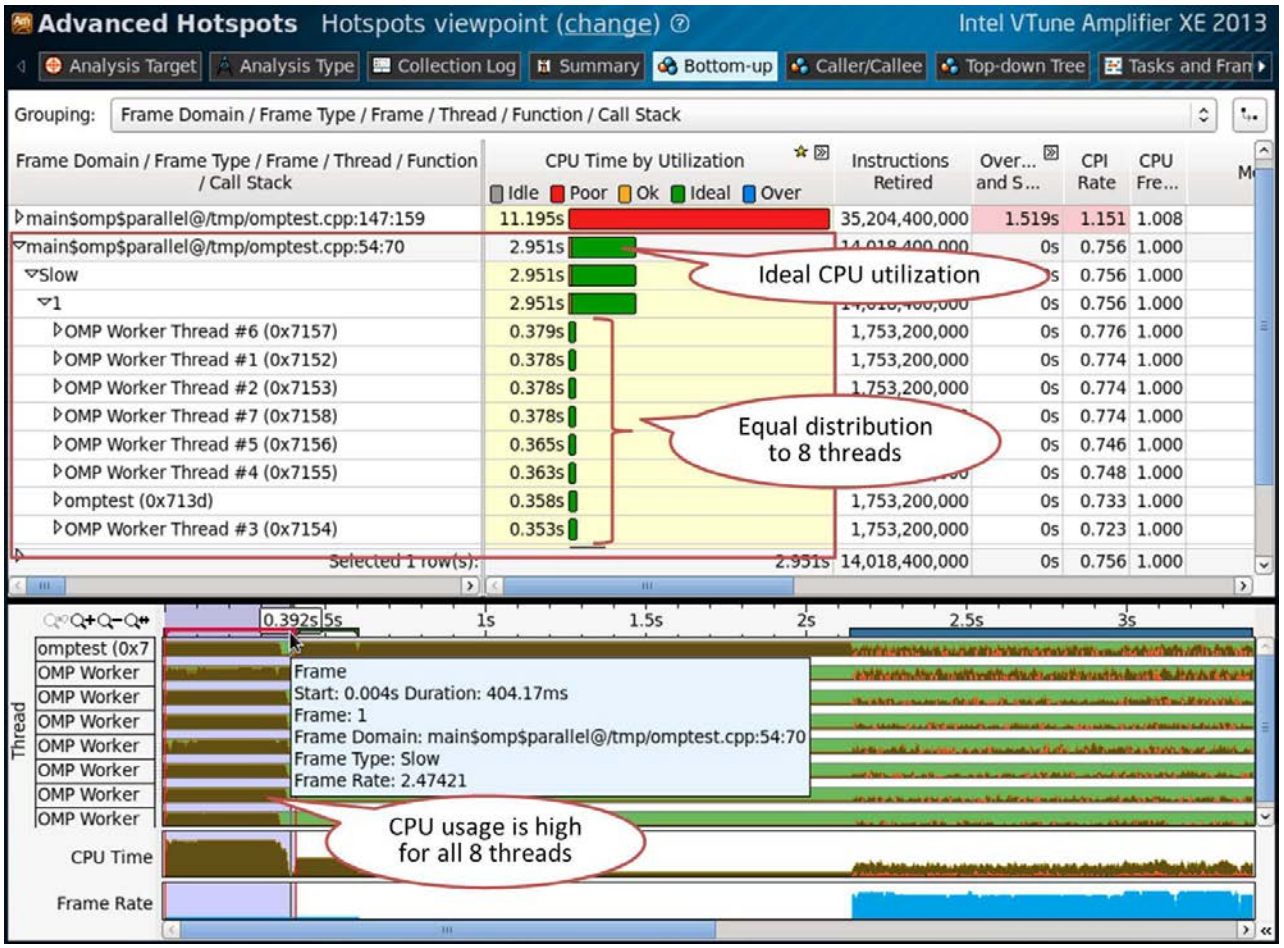

図 6. バランスの取れた並列領域

54 行目のコード:

```
#pragma omp parallel for schedule(static,1)                    // 54 行目
for (int index = 0 ; index < oloops ; index++)
{
   double *a, *b, *c, *at ;
   int ick ;

      a   =   ga   +
      index*84 ; c =
      gc           +
      index*84     ;
      fillmat (a) ;

      ick   =   work   (a,
      c,gmask) ;
      if  (ick
      > 0)
      {
         printf("error ick failed\n") ;
         exit(1) ;
      }
}
```

シナリオ 2: バランスの取れていない並列領域

82 行目の並列領域はバランスが取れていません。利用可能な 8 つのスレッドのうち 4 つしか使用しておらず、残りの 4 つは待機しています。これは、この領域の CPU 使用率が Poor (赤色) で示されていることからも明らかです (図 7 を参照)。

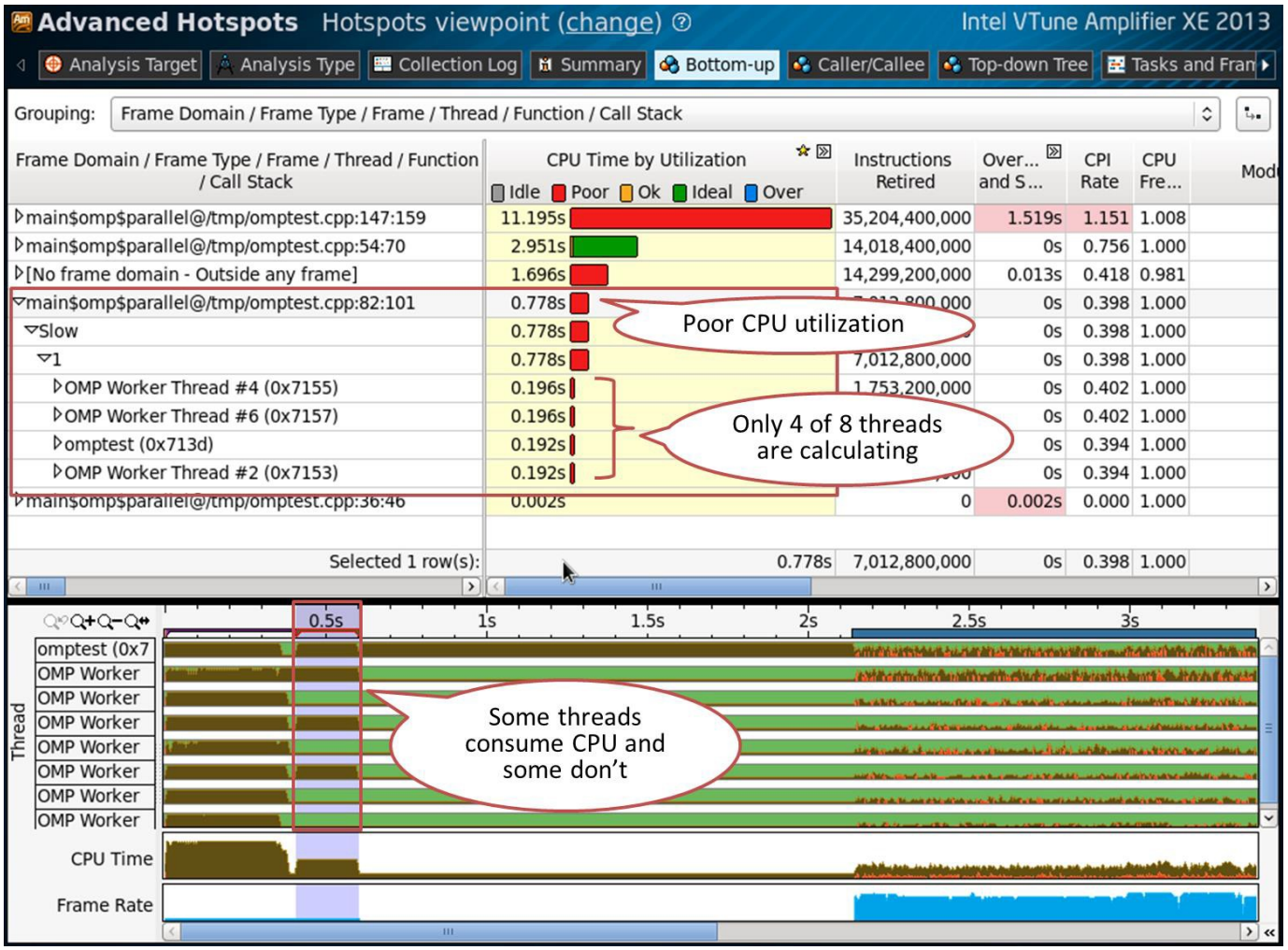

図 7. バランスの取れていない並列領域

82 行目のコード:

```
#pragma omp parallel for schedule(static,1)          // 82 行目
for (int index = 0 ; index < oloops ; index++)
{
   double *a, *b, *c, *at ;
   int ick ;

   if (index%2 == 0)
   {
      a    =    ga    +
      index*84  ;  c  =
      gc             +
      index*84       ;
      fillmat (a) ;

      ick   =   work   (a,   c,
      gmask) ;
      if   (ick
      > 0)
      {
         printf("error ick failed\n") ;
         exit(1) ;

      }
   }
}
```

シナリオ 3: 粒度の問題

前述の 2 つのシナリオでは、フレームドメインとフレームはそれぞれ 1 つでした。ここでは、147 行目の並列領域に複数のフレームがあります (図 8 を参照)。これは、並列領域が複数回呼び出されていることを示します。各フレームの CPU 時間はわずかで、これはマウスでタイムライン上のフレームをポイントしたときに表示されるフレームの統計情報からも明らかです。このことから、粒度が細かすぎることが分かります。つまり、OpenMP* 並列領域の呼び出しが多すぎる割に、各呼び出しに費やされる時間が短すぎます。これは、同期とスケジュールのオーバーヘッドを増加させ、CPU 使用率の低下につながります。

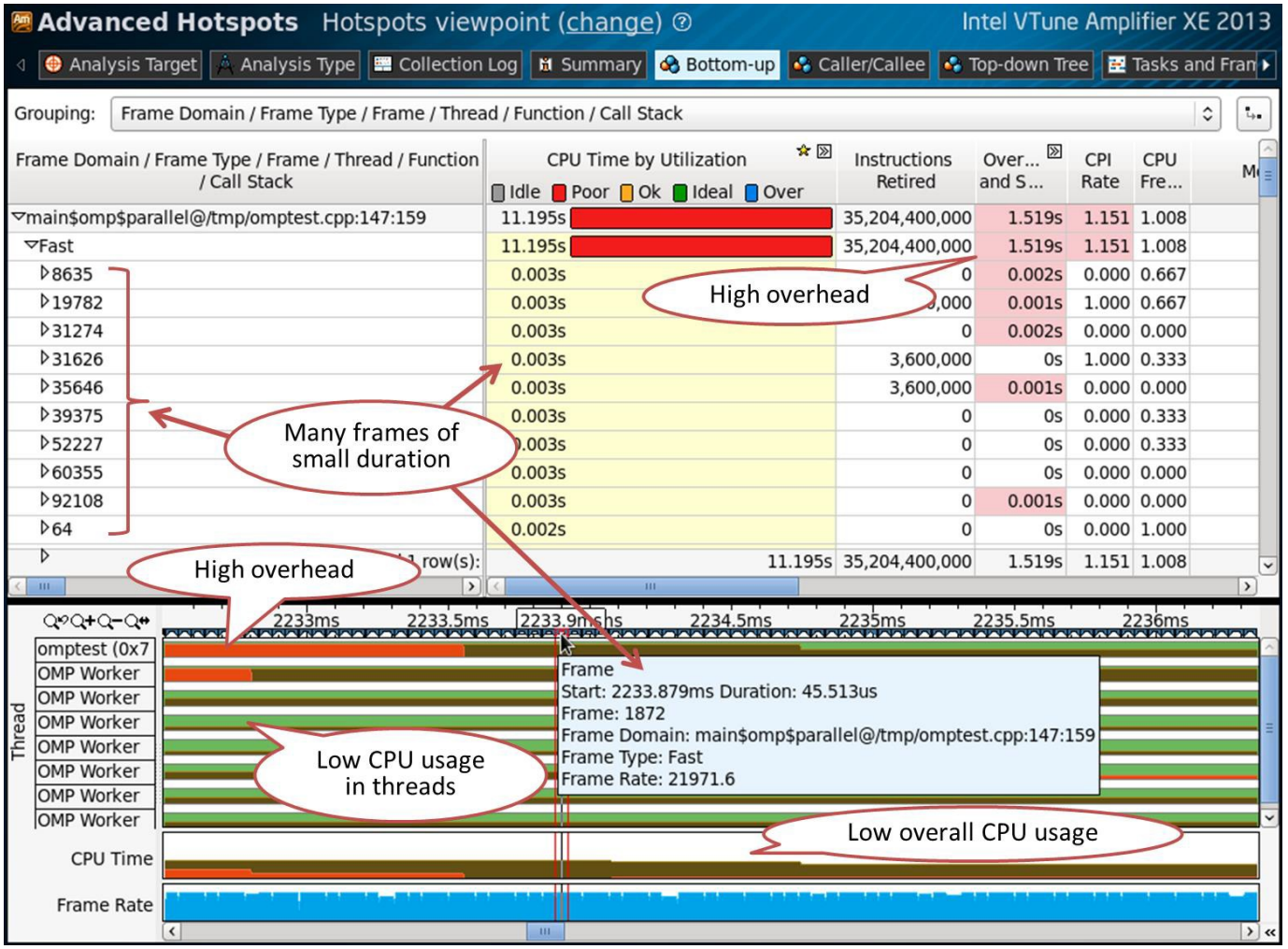

図 8. 粒度の問題

147 行目のコード:

```
for (q = 0 ; q < LOOPS ; q++)
{
   #pragma omp parallel for schedule(static,1) firstprivate(tcorrect)
   lastprivate(tcorrect)
      // 147 行目
   for (int index = 0 ; index < oloops ; index++)
   {
      double  *la,
      *lc;
      int
      lq,lmask ;

      la    =    ga    +
      index*84 ; lc =
      gc + index*84 ;
      lq = q ;
      lmask      =
      gmask ;
      ick    =    work1(ga,    gc,
      lq,lmask) ;
      if    (ick    ==    VLEN)
      tcorrect++ ;
   }
}
```

シナリオ 4: 同期オブジェクトと待機時間

同期オブジェクトの待機は、深刻なパフォーマンス・ボトルネックになることがあります。アプリケーションの同期と待機に関する詳細な情報を収集するには、[Locks and Waits] 解析を実行します。解析を実行する前に、解析の設定で [Analyze user tasks] チェックボックスと [Analyze Intel runtimes and user synchronization] チェックボックスをオンにします。

[Bottom-up] ペインは、デフォルトで同期オブジェクトを待機時間順に表示します (図 9 を参照)。インテル® VTune™ Amplifier XE は、OpenMP* ランタイム内で使用される "omp critical" 構文などの OpenMP* 同期プリミティブと OpenMP* barriers などの内部同期オブジェクトを識別することができます。そのため、待機に費やされた時間とその詳細 (短時間の待機が多いことや長時間の待機がいくつかあるかなど) が分かります。また、インテル® VTune™ Amplifier XE は、スレッドがビジーウェイト状態 (スピン時間) か、実際に待機状態に遷移しているかも示します。タイムラインに遷移状態 (黄色の縦線) が表示されるため、どのスレッドがどの同期オブジェクトで、どれだけ頻繁に競合し、どれだけ待機したかといった詳細を把握できます。

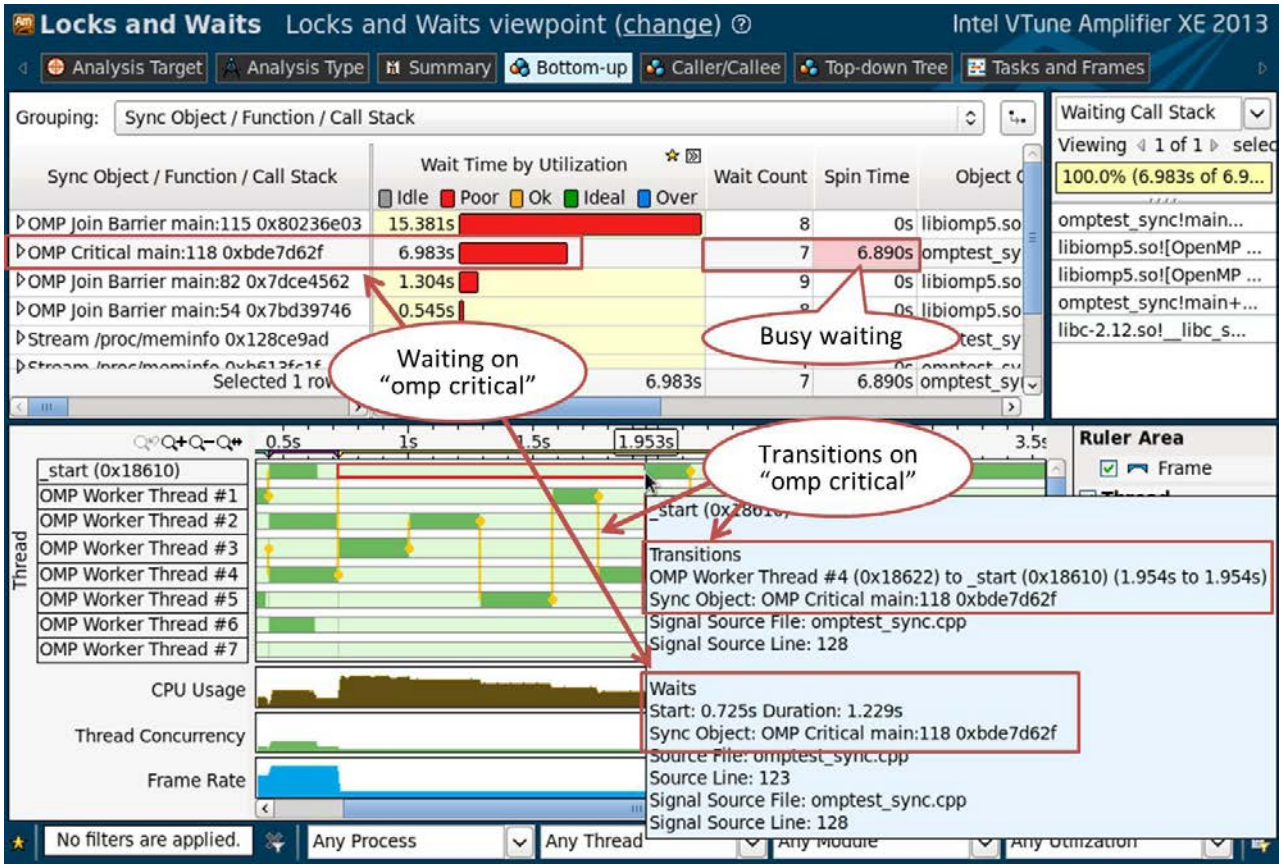

図 9. 同期と待機

118 行目のコード:

```
#pragma omp parallel for schedule(static,1)
for (int index = 0 ; index < oloops ; index++)
{
   #pragma omp critical (my_sync)            // 118 行目
   {
      double  *a,  *b,  *c,
      *at ;
      int
      ick ;

      a    =    ga    +
      index*84  ;  c  =
      gc + index*84 ;
      fillmat (a) ;

      ick    =    work    (a,
      c,gmask) ;
      if (ick >
      0)
      {
         printf("error ick failed\n") ;
         exit(1) ;
      }
   }
}
```

まとめ

インテル® VTune™ Amplifier XE により、OpenMP* ベースのアプリケーションの内部を掘り下げてみることができます。コードのシリアル/並列領域のバランスを予測し、各並列領域におけるプログラムの動作を確認できます。インテル® VTune™ Amplifier XE は、OpenMP* スレッド間のワーク分割や粒度に関する問題の検出、スレッドのスケジュールに起因するオーバーヘッドの予測、そして同期スキームの理解を支援します。CPU 使用率とスレッドのスケジュールに関する詳細な統計を特定の OpenMP* 並列領域に紐付けることで、アプリケーションの内部動作に関する情報を提供します。インテルの OpenMP* ランタイムを使用すると最も詳しい情報が得られますが、GCC* や Microsoft* の OpenMP* ランタイムを使用するコードをプロファイルすることも可能です。

Kirill Rogozhin
インテル コーポレーション
開発製品部門ソフトウェア & サービスグループ
テクニカル・コンサルティング・エンジニア