

# インテル® VTune™ Amplifier XE 2016

## インテル® VTune™ Amplifier XE 2016

インテル® ソフトウェア開発ツール

### アプリケーションをチューニングしてスケーラブルなマルチコア・パフォーマンスを実現

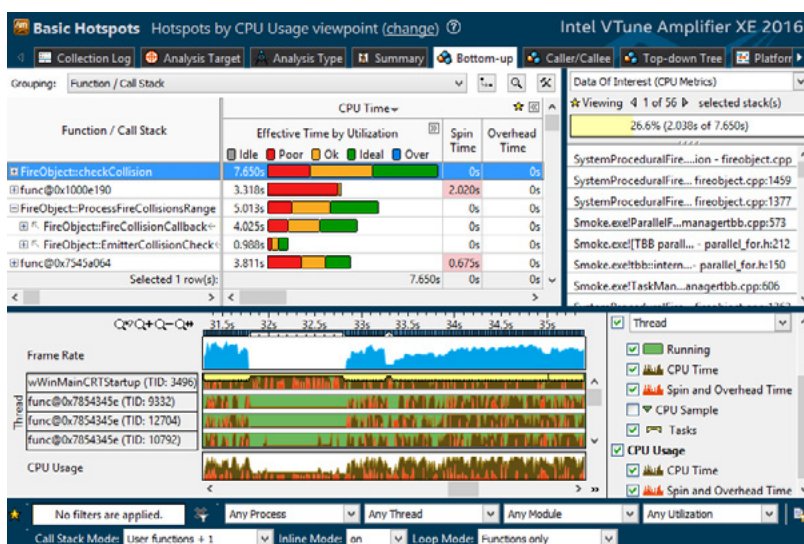
- 高速なコード – 低オーバーヘッドで正確なデータに基づいたチューニング
- 迅速に処理 – 簡単な解析により詳細なデータを取得
- より多くのデータ – CPU、GPU、帯域幅、スレッドなど
- ローカルおよびリモート収集

「インテル® VTune™ Amplifier XE は、複雑なコードを解析し、迅速にボトルネックを特定するのに役立ちました。ほかのインテル® ソフトウェア開発ツールと併用することで、以前のバージョンと比較して PIPESIM のパフォーマンスを 10 倍も向上することができました。」

**Schlumberger**  
シニア・サイエンティスト  
Rodney Lessard 氏

### 新機能

- OpenMP\* を素早くチューニング – 適切なデータにより効率良くチューニング
- 複数のランクからなるハイブリッド MPI/OpenMP\* を簡単にチューニング
- GPU および OpenCL\* プログラム・プロファイリング (Linux\* および Windows\*)
- マルチソケット帯域幅の最適化
- その他のさまざまな機能



## データを利用しないチューニングは単なる推測作業

初めてチューニングを行う場合でも、高度なパフォーマンスの最適化を行う場合でも、インテル® VTune™ Amplifier XE は広範なチューニングのニーズに応えるデータを提供します。hotspot、スレッド化、OpenCL\*、ロックと待機、DirectX\*、帯域幅などに関する豊富なパフォーマンス・データを収集します。

しかし、良いデータだけでは十分ではありません。そのデータを解析して、簡単に解釈できるツールが必要です。優れた解析ツールは、タイムラインやソースコードで結果をソートし、フィルターし、視覚化することで、シリアル時間とロード・インバランスを特定したり、遅い OpenMP\* インスタンスの原因を調査できます。

## 必要なデータを取得

- hotspot (統計コールツリー)、呼び出しカウント (統計)
- ロックと待機の解析によるスレッド・プロファイル
- キャッシュミス、帯域幅解析
- OpenCL\* プログラム・カーネル・トレースと GPU オフロード

「インテル® VTune™ Amplifier XE によって提供された情報に基づいてコードを最適化したところ、シングルコアでも大幅なパフォーマンスの向上 (約 2 倍) が得られました。」

**Mentor Graphics Corporation**  
 機械分析部門  
 R&D 副ディレクター  
 Alexey Andrianov 氏

## 簡単に使用可能

- 特別なコンパイラは不要: C、C++、C#、Fortran、Java\*、ASM
- Visual Studio\* (Windows\*)、Eclipse\* (Linux\*) 統合環境またはスタンドアロン (Windows\* および Linux\*)
- グラフィカル・インターフェイスとコマンドライン
- ローカルおよびリモートデータ収集
- 新機能: OS X\* から Windows\* および Linux\* データを解析

## 必要な情報を迅速に表示

- ソース / アセンブリで結果を表示
- OpenMP\* のスケーラビリティ解析、グラフィカル・フレーム解析
- 関係のないデータを非表示
- ビューポイントでデータをフィルタリング
- スレッドおよびタスク・アクティビティをタイムライン表示

## 低オーバーヘッドで細かいハードウェア・プロファイリング

インテル® プロセッサには、オンチップのパフォーマンス・モニタリング・ユニット (PMU) があります。インテル® プロセッサおよび互換プロセッサに対応した Basic Hotspots (基本的な hotspot) 解析に加えて、インテル® VTune™ Amplifier XE には、インテル® プロセッサ上の PMU を利用してデータを収集する低オーバーヘッドな Advanced Hotspots (高度な hotspot) 解析があります。システムワイドの解析は、ドライバーの解析に役立ちます。また、より短いサンプリング間隔の指定が可能になり (約 10 ミリ秒から約 1 ミリ秒に向上)、実行時間が短い小さな関数の hotspots も見つけることができます。

## 製品の詳細

機能	詳細																																				
<p><b>多くの CPU 時間を費やしているコードを素早く特定</b></p> <p>hotspot 解析は、多くの CPU 時間を費やしている関数のリストをソートして表示します。これは、チューニングで最も大きな効果が得られる部分です。[+] をクリックするとコールスタックが表示され、ダブルクリックするとソースを確認できます。</p>	 <table border="1"> <caption>CPU Time Analysis</caption> <thead> <tr> <th>Function / Call Stack</th> <th>Effective Time by Utilization</th> <th>Spin Time</th> <th>Overhead Time</th> </tr> </thead> <tbody> <tr> <td>std::basic_ifstream&lt;char,struct std::char_traits&lt;</td> <td>3.287s</td> <td>0s</td> <td>0s</td> </tr> <tr> <td>FireObject::ProcessFireCollisionsRange</td> <td>2.450s</td> <td>0s</td> <td>0s</td> </tr> <tr> <td>FireObject::FireCollisionCallback&lt; Parallel</td> <td>2.155s</td> <td>0s</td> <td>0s</td> </tr> <tr> <td>FireObject::EmitterCollisionCheck&lt; FireO</td> <td>0.295s</td> <td>0s</td> <td>0s</td> </tr> <tr> <td>CBaseDevice::Present</td> <td>2.200s</td> <td>0.180s</td> <td>0s</td> </tr> <tr> <td>D3DXCompileShader</td> <td>2.010s</td> <td>0s</td> <td>0s</td> </tr> </tbody> </table>	Function / Call Stack	Effective Time by Utilization	Spin Time	Overhead Time	std::basic_ifstream<char,struct std::char_traits<	3.287s	0s	0s	FireObject::ProcessFireCollisionsRange	2.450s	0s	0s	FireObject::FireCollisionCallback< Parallel	2.155s	0s	0s	FireObject::EmitterCollisionCheck< FireO	0.295s	0s	0s	CBaseDevice::Present	2.200s	0.180s	0s	D3DXCompileShader	2.010s	0s	0s								
Function / Call Stack	Effective Time by Utilization	Spin Time	Overhead Time																																		
std::basic_ifstream<char,struct std::char_traits<	3.287s	0s	0s																																		
FireObject::ProcessFireCollisionsRange	2.450s	0s	0s																																		
FireObject::FireCollisionCallback< Parallel	2.155s	0s	0s																																		
FireObject::EmitterCollisionCheck< FireO	0.295s	0s	0s																																		
CBaseDevice::Present	2.200s	0.180s	0s																																		
D3DXCompileShader	2.010s	0s	0s																																		
<p><b>結果をソースで確認</b></p> <p>関数リストをダブルクリックすると、関数で最も時間を費やしている箇所に移動します。</p>	 <table border="1"> <caption>Source CPU Time Analysis</caption> <thead> <tr> <th>Source Line</th> <th>Source</th> <th>Effective Time by Utilization</th> <th>Spin Time</th> <th>Overhead Time</th> </tr> </thead> <tbody> <tr> <td>1,456</td> <td>for( u32 j = rangeBegin; j &lt; range</td> <td>0.5%</td> <td>0.0%</td> <td>0.0%</td> </tr> <tr> <td>1,457</td> <td>{</td> <td>0.0%</td> <td>0.0%</td> <td>0.0%</td> </tr> <tr> <td>1,458</td> <td>FireObject *pfo = m_pFireObj</td> <td>0.4%</td> <td>0.0%</td> <td>0.0%</td> </tr> <tr> <td>1,459</td> <td>if( checkCollision(trp, trpp,</td> <td>5.4%</td> <td>0.0%</td> <td>0.0%</td> </tr> <tr> <td>1,460</td> <td>{</td> <td>0.0%</td> <td>0.0%</td> <td>0.0%</td> </tr> <tr> <td>1,461</td> <td>// if it passes this test</td> <td>0.0%</td> <td>0.0%</td> <td>0.0%</td> </tr> </tbody> </table>	Source Line	Source	Effective Time by Utilization	Spin Time	Overhead Time	1,456	for( u32 j = rangeBegin; j < range	0.5%	0.0%	0.0%	1,457	{	0.0%	0.0%	0.0%	1,458	FireObject *pfo = m_pFireObj	0.4%	0.0%	0.0%	1,459	if( checkCollision(trp, trpp,	5.4%	0.0%	0.0%	1,460	{	0.0%	0.0%	0.0%	1,461	// if it passes this test	0.0%	0.0%	0.0%	
Source Line	Source	Effective Time by Utilization	Spin Time	Overhead Time																																	
1,456	for( u32 j = rangeBegin; j < range	0.5%	0.0%	0.0%																																	
1,457	{	0.0%	0.0%	0.0%																																	
1,458	FireObject *pfo = m_pFireObj	0.4%	0.0%	0.0%																																	
1,459	if( checkCollision(trp, trpp,	5.4%	0.0%	0.0%																																	
1,460	{	0.0%	0.0%	0.0%																																	
1,461	// if it passes this test	0.0%	0.0%	0.0%																																	
<p><b>ロックと待機の解析によりスレッドをチューニング</b></p> <p>並列プログラムでパフォーマンスが低下する一般的な原因 — ロックの待機に長時間が費やされ、コアが十分に活用されない — を素早く特定します。</p>	 <table border="1"> <caption>Wait Time by Thread Concurrency</caption> <thead> <tr> <th>Sync Object / Function / Call Stack</th> <th>Wait Time</th> <th>Wait Count</th> <th>Object Type</th> </tr> </thead> <tbody> <tr> <td>Manual Reset Event 0xf04628bd</td> <td>71.808s</td> <td>1,072</td> <td>Manual Reset Event</td> </tr> <tr> <td>Auto Reset Event 0xcc18b37c</td> <td>41.789s</td> <td>2,540</td> <td>Auto Reset Event</td> </tr> <tr> <td>Thread Pool</td> <td>38.303s</td> <td>1</td> <td>Constant</td> </tr> <tr> <td>Sleep</td> <td>38.212s</td> <td>3,815</td> <td>Constant</td> </tr> <tr> <td>Manual Reset Event 0xba2e95f3</td> <td>35.302s</td> <td>505</td> <td>Manual Reset Event</td> </tr> <tr> <td>Auto Reset Event 0x38cd6d85</td> <td>0.737s</td> <td>298</td> <td>Auto Reset Event</td> </tr> </tbody> </table>	Sync Object / Function / Call Stack	Wait Time	Wait Count	Object Type	Manual Reset Event 0xf04628bd	71.808s	1,072	Manual Reset Event	Auto Reset Event 0xcc18b37c	41.789s	2,540	Auto Reset Event	Thread Pool	38.303s	1	Constant	Sleep	38.212s	3,815	Constant	Manual Reset Event 0xba2e95f3	35.302s	505	Manual Reset Event	Auto Reset Event 0x38cd6d85	0.737s	298	Auto Reset Event								
Sync Object / Function / Call Stack	Wait Time	Wait Count	Object Type																																		
Manual Reset Event 0xf04628bd	71.808s	1,072	Manual Reset Event																																		
Auto Reset Event 0xcc18b37c	41.789s	2,540	Auto Reset Event																																		
Thread Pool	38.303s	1	Constant																																		
Sleep	38.212s	3,815	Constant																																		
Manual Reset Event 0xba2e95f3	35.302s	505	Manual Reset Event																																		
Auto Reset Event 0x38cd6d85	0.737s	298	Auto Reset Event																																		
<p><b>OpenMP* を素早くチューニング — 適切なデータにより効率良くチューニング</b></p> <p>OpenMP* コードの効率が悪い原因を影響の大きい順に表示します。低オーバーヘッドで正確なデータが得られます。</p> <p><b>複数のランクからなるハイブリッド MPI/OpenMP* を簡単にチューニング</b></p> <p>インテル® Trace Analyzer &amp; Collector で選択された複数の MPI ランクをプロファイリングします。OpenMP* パフォーマンス向上の可能性が大きい順に表示できます。</p>	 <table border="1"> <caption>OpenMP Potential Gain</caption> <thead> <tr> <th>OpenMP Region / Function / Call Stack</th> <th>Imbalance</th> <th>Lock Contention</th> <th>Creation</th> <th>Scheduling</th> <th>Reduction</th> <th>Other</th> </tr> </thead> <tbody> <tr> <td>conj_grad_\$omp\$parallel:24@</td> <td>3.944s</td> <td>0s</td> <td>0.000s</td> <td>0.002s</td> <td>0.000s</td> <td>0.010s</td> </tr> <tr> <td>MAIN_\$omp\$parallel:24@/h</td> <td>0.086s</td> <td>0s</td> <td>0s</td> <td>0s</td> <td>0s</td> <td>0.000s</td> </tr> <tr> <td>[Serial - outside any region]</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>0s</td> </tr> </tbody> </table>	OpenMP Region / Function / Call Stack	Imbalance	Lock Contention	Creation	Scheduling	Reduction	Other	conj_grad_\$omp\$parallel:24@	3.944s	0s	0.000s	0.002s	0.000s	0.010s	MAIN_\$omp\$parallel:24@/h	0.086s	0s	0s	0s	0s	0.000s	[Serial - outside any region]						0s								
OpenMP Region / Function / Call Stack	Imbalance	Lock Contention	Creation	Scheduling	Reduction	Other																															
conj_grad_\$omp\$parallel:24@	3.944s	0s	0.000s	0.002s	0.000s	0.010s																															
MAIN_\$omp\$parallel:24@/h	0.086s	0s	0s	0s	0s	0.000s																															
[Serial - outside any region]						0s																															
<p><b>チューニングの可能性を素早く特定</b></p> <p>潜在的なチューニングの可能性がある場合、セルはピンクでハイライト表示されます。カーソルを移動するとアドバイスが表示される特にキャッシュ、帯域幅などの高度な最適化で役立ちます。</p>	 <table border="1"> <caption>Unfilled Pipeline Slots (Stalls)</caption> <thead> <tr> <th>Function / Call Stack</th> <th>Clockticks</th> <th>Instructions Retired</th> <th>CPI Rate</th> <th>Back-end Bound</th> <th>Front-end Bound</th> </tr> </thead> <tbody> <tr> <td>FireObject::checkCollision</td> <td>16,639,342,359</td> <td>9,528,342,771</td> <td>1.746</td> <td>High</td> <td>Low</td> </tr> <tr> <td>FireObject::ProcessFireCollisionsRange</td> <td>7,789,603,182</td> <td>6,154,123,581</td> <td>1.266</td> <td>High</td> <td>Low</td> </tr> <tr> <td>FireObject::FireCollisionCallback</td> <td>5,696,103,361</td> <td></td> <td></td> <td>Low</td> <td>High</td> </tr> <tr> <td>ParallelForBody::operator()&lt; [TBB</td> <td>5,692,521,630</td> <td></td> <td></td> <td>Low</td> <td>High</td> </tr> <tr> <td>[TBB parallel_for on class ParallelF</td> <td>3,581,731</td> <td></td> <td></td> <td>Low</td> <td>High</td> </tr> </tbody> </table> <p>Selected 1 row(s): 16,639,342,359</p> <p>Threshold: (((1 - ((IDQ_UOPS_NOT_DELIVERED.CORE + L</p>	Function / Call Stack	Clockticks	Instructions Retired	CPI Rate	Back-end Bound	Front-end Bound	FireObject::checkCollision	16,639,342,359	9,528,342,771	1.746	High	Low	FireObject::ProcessFireCollisionsRange	7,789,603,182	6,154,123,581	1.266	High	Low	FireObject::FireCollisionCallback	5,696,103,361			Low	High	ParallelForBody::operator()< [TBB	5,692,521,630			Low	High	[TBB parallel_for on class ParallelF	3,581,731			Low	High
Function / Call Stack	Clockticks	Instructions Retired	CPI Rate	Back-end Bound	Front-end Bound																																
FireObject::checkCollision	16,639,342,359	9,528,342,771	1.746	High	Low																																
FireObject::ProcessFireCollisionsRange	7,789,603,182	6,154,123,581	1.266	High	Low																																
FireObject::FireCollisionCallback	5,696,103,361			Low	High																																
ParallelForBody::operator()< [TBB	5,692,521,630			Low	High																																
[TBB parallel_for on class ParallelF	3,581,731			Low	High																																

## 動作環境

プロセッサ	インテル® プロセッサ / コプロセッサおよび互換プロセッサ / コプロセッサ
言語	C、C++、C#、Fortran、Java*、ASM ほか。Microsoft* コンパイラー、GCC、インテル® コンパイラー、そのほか標準に準拠するコンパイラーで動作します。
コンパイラー	Microsoft* コンパイラー、GCC、インテル® コンパイラー、そのほか標準に準拠するコンパイラーで動作
開発環境	スタンドアロンまたは Microsoft* Visual Studio* および Eclipse* 統合環境
ホスト・オペレーティング・システム	Windows*、Linux* をサポート。Windows* または Linux* データ用の OS X* ビューアーもあります。
ターゲット・オペレーティング・システム	Windows* および Linux* をサポート
スレッド化モデル	インテルによる OpenMP* 実装、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)、インテル® Cilk™ Plus。スレッドの詳細な情報を提供します。
拡張スレッド・パフォーマンス解析	インテルによる OpenMP* 実装
MPI 並列処理	インテル® Trace Analyzer & Collector の MPI プロファイラーとの統合
GPU	最新のインテル® プロセッサでの OpenCL* プログラムおよびメディア・アプリケーションのチューニング

## インテル® Parallel Studio XE スイートに含まれるコンポーネント

インテル® VTune™ Amplifier XE は、並列ソフトウェア開発向けの統合ソフトウェア開発スイートであるインテル® Parallel Studio XE またはスタンドアロンで使用できます。

### 関連情報

#### インテル® VTune™ Amplifier XE

[software.intel.com/en-us/intel-vtune-amplifier-xe](http://software.intel.com/en-us/intel-vtune-amplifier-xe)

#### 30 日間の評価版

[software.intel.com/en-us/intel-vtune-amplifier-xe/try-buy](http://software.intel.com/en-us/intel-vtune-amplifier-xe/try-buy)



本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品適格性、あらゆる特許権、著作権、その他知的財産権の非侵害性への保証を含む) に関してもいかなる責任も負いません。インテルによる書面での合意がない限り、インテル製品は、その欠陥や故障によって人身事故が発生するようなアプリケーションでの使用を想定した設計は行われていません。

インテル製品は、予告なく仕様や説明が変更されることがあります。機能または命令の一覧で「留保」または「未定義」と記されているものがありますが、その「機能が存在しない」あるいは「性質が留保付である」という状態を設計の前提にしないでください。これらの項目は、インテルが将来のために留保しているものです。インテルが将来これらの項目を定義したことにより、衝突が生じたり互換性が失われたりしても、インテルは一切責任を負いません。この情報は予告なく変更されることがあります。この情報だけに基づいて設計を最終的なものとししないでください。

本資料で説明されている製品には、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。最新の仕様をご希望の場合や製品をご注文の場合は、お近くのインテルの営業所または販売代理店にお問い合わせください。本資料で紹介されている資料番号付きのドキュメントや、インテルのその他の資料を入手するには、1-800-548-4725 (アメリカ合衆国) までご連絡いただくか、インテルの Web サイト ([www.intel.com](http://www.intel.com)) を参照してください。

© 2015 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Cilk、VTune は、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。OpenCL および OpenCL ロゴは、Apple Inc. の商標であり、Khronos の使用許諾を受けて使用しています。\* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。