

# 製品概要

パフォーマンス・プロファイラー  
インテル® VTune™ プロファイラー



# 最新のハードウェア向けにソフトウェア・パフォーマンスを最適化

## インテル® VTune™ プロファイラーでパフォーマンス・ボトルネックを素早く見つける

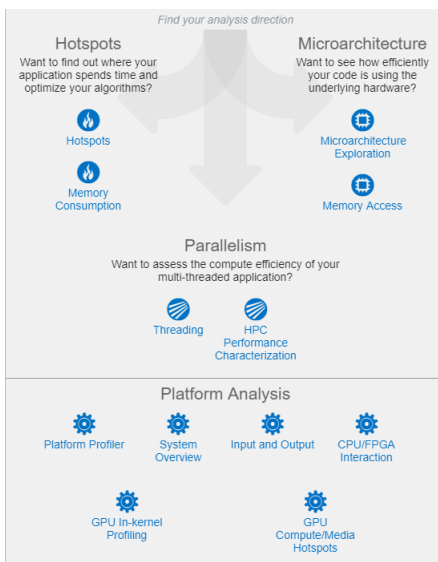


図 1. 豊富なパフォーマンス・データのセットを収集して解析

## 広範なパフォーマンス・データを収集

単純なアプリケーションのチューニングを行う場合でも、スレッド化された MPI アプリケーションの高度なパフォーマンス最適化を行う場合でも、インテル® VTune™ プロファイラーで必要なデータを取得可能です。hotspot、スレッド化、ロックと待機、DirectX\*、OpenMP\*、インテル® スレディング・ビルディング・ブロック (インテル® TBB)、帯域幅、キャッシュ、メモリアクセス、ストレージ・レイテンシー、OpenCL\* アプリケーションなどに関する豊富なパフォーマンス・データのセットを収集します (図 1)。

インテル® プロセッサーで実行する C、C++、C#、Fortran、Python\*、Go\*、Java\*、OpenCL\*、およびこれらの言語が混在するコードを正確にプロファイルします。単一言語のプロファイラーとは異なり、インテル® VTune™ プロファイラーは言語が混在したコードを解析できます。

- より多くのデータを表示 - CPU、FPU、GPU、スレッド化、メモリアクセスなど
- 迅速に結果を表示 - 簡単な解析により詳細なデータを取得
- 高速なコードを開発 - 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
- ワークフローを選択 - ローカル/リモート収集またはコマンドライン/グラフィカル・インターフェイスを選択可能

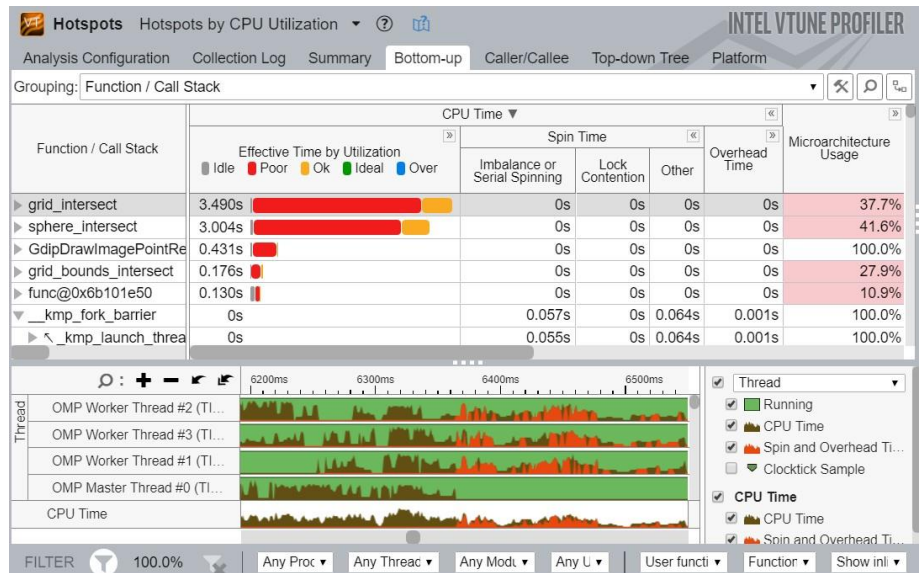


図 2. hotspot 解析は、アプリケーションで最も時間が消費されている場所を表示します。また、パフォーマンス向上の可能性と、ロード・インバランス、ロック競合、フォーク、スケジュール、リダクションのような一般的なパフォーマンス低下の原因を示す、スレッド・パフォーマンスの詳細な解析結果も提供します。

## 強力なデータ解析で時間を節約

優れたデータがあっても、それを役立てることができなければ意味がありません。データを考察するにはマイニングが必要です。ハイレベルのサマリーと強力な解析により、タイムラインやソースコードで結果をソート、フィルター、視覚化して時間を節約できます。

## 2020 の新機能

- **メモリー解析:** インテル® Optane™ DC パーシステント・メモリー向けに設計および最適化
- **プラットフォーム・プロファイラー:** システムメトリックを素早く分かりやすく表示
- **I/O 解析:** PCIe\* デバイスメトリック
- **HPC 解析:** ベクトル化メトリック、プロセスおよびスレッド・アフィニティー、Lustre\* 並列ファイル I/O メトリック (プレビュー機能)
- **スナップショット:** 通信パターン診断と Open MPI\*
- **GPU および OpenCL\* アプリケーション解析:** インラインフィルターと命令カウント
- **Linux\*:** ドライバーを追加せずに解析タイプを追加
- **パフォーマンス解析クックブック:** ユーザーフレンドリーなレシピ
- **ユーザー・インターフェイスの改良:** 初めて使用するユーザー向けにツアーを用意
- **より分かりやすい製品名に変更** (旧称: インテル® VTune™ Amplifier)

## 柔軟なオプション: スタンドアロンまたは包括的な開発ツールスイートに統合

インテル® VTune™ プロファイラーは、スタンドアロン製品または統合ソフトウェア開発ツールスイートの一部として利用できます。

- **インテル® Parallel Studio XE** は、エンタープライズ、クラウド、HPC、AI 向けのハイパフォーマンスでスケーラブルな並列アプリケーションの開発を支援する開発スイートです。スイートには、コンパイラー、ライブラリー、その他の解析ツールも含まれます。
- **インテル® System Studio Professional Edition** および **Ultimate Edition** は、スマート・コネクテッド・デバイスのシステムと IoT 開発に使用されます。スイートには、コンパイラー、ライブラリー、解析ツール、クラウドコネクタも含まれており、400 を超えるセンサーを利用できます。

インテル® VTune™ プロファイラーを補完する解析ツール:

- **インテル® Advisor** は、ベクトル化、スレッド化、フローグラフの最適化を支援します。ルーブリック解析は改善の余地が最も大きいループを見つけます。

## 製品詳細

- **インテル® Trace Analyzer & Collector** は、MPI アプリケーションを検証し、スレッド化により最も大きな利点が得られるループをインテル® VTune™ プロファイラーに知らせます。

## 必要なデータを取得

- **hotspot** (統計コールツリー)
- **スレッド・プロファイル** (ロックと待機の解析)
- **メモリーアクセス**、キャッシュミス、帯域幅、NUMA 解析
- **FLOPS と FPU 利用率**
- **ストレージアクセス** (ソースマップ)、レイテンシー・ヒストグラム、I/O 待機
- **OpenCL\*** プログラムカーネルのトレースと GPU オフロード

## 簡単に使用可能

- 特別なコンパイラーは不要: C、C++、C#、Fortran、Java\*、Python\*、Go\*、ASM
- Microsoft\* Visual Studio\* IDE 統合
- 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、マルチランク対応 (MPI アプリケーション)
- Linux\*、Windows\*、FreeBSD\*、Android\*、および一部の組み込み OS でデータを収集
- Linux\*、Windows\*、macOS\* ホストで結果を解析

## 必要な情報を迅速に表示

- ソース/アセンブリーで結果を表示
- OpenMP\* スケーラビリティ解析とグラフィカル・フレーム解析
- **メモリー解析:** データ構造のチューニングと NUMA レイテンシーの最適化
- **ストレージ解析:** I/O ボトルネックを特定
- **タイムラインとビューポイントでデータをフィルターして関係のないデータを非表示**
- スレッドおよびタスク・アクティビティーをタイムライン表示

## 低オーバーヘッドで詳細なハードウェア・プロファイル

インテル® プロセッサーと互換プロセッサーに対応した基本的な hotspot 解析に加えて、インテル® VTune™ プロファイラーには、インテル® プロセッサー上のオンチップのパフォーマンス・モニタリング・ユニット (PMU) を利用してデータを収集する低オーバーヘッドの高度な hotspot 解析があります。キャッシュミス、分岐予測ミス、帯域幅などの重要なパフォーマンス問題も見つけることができます。

## 多くの CPU 時間を費やしているコードを素早く特定

hotspots 解析は、多くの CPU 時間を費やしている関数のソートされたリストを表示します。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が表示されます。

Function / Call Stack	CPU Time					Spin Time	Overhead Time
	Effective Time by Utilization						
	Idle	Poor	Ok	Ideal	Over		
▼ FireObject::checkCollision	3.348s					0s	0s
▶ [Loop at line 1453 in FireObject::P	2.771s					0s	0s
▶ [Loop at line 1491 in FireObject::P	0.578s					0s	0s
▶ [Loop at line 1453 in FireObject::Proces	1.052s					0s	0s
▶ rand	0.696s					0s	0s
▶ ParticleEmitter::FirePatch::initParticle	0.520s					0s	0s

## 製品詳細 (続き)

<p><b>ソースで結果を確認</b></p> <p>関数リストでダブルクリックすると、関数の最も多くの CPU 時間を費やしている場所が表示されます。C、C++、Fortran、アセンブリ、Java*、Python*、Go*、OpenCL* カーネルで利用できます。ソース行レベルのプロファイル情報を確認できます。</p>	<table border="1"> <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(ttp, tttp,</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(ttp, tttp,	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(ttp, tttp,	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>適切なデータにより OpenMP* と Intel® TBB を簡単にチューニング</b></p> <p>低オーバーヘッドで正確なデータを利用して、スレッド化の効率が悪い原因を影響の大きい順に確認できます。</p> <p><b>複数のランクからなるハイブリッド MPI/OpenMP* を最適化</b></p> <p>OpenMP* パフォーマンス・ゲインの大きい順に結果がソートされます。</p>	<table border="1"> <thead> <tr> <th>Process / OpenMP Region / Function / Thread / Call Stack</th> <th>Effective Time by Utilization</th> <th>Spin Time</th> </tr> </thead> <tbody> <tr> <td>▶ heart_demo (rank 15)</td> <td>99.641s</td> <td>21.705s</td> </tr> <tr> <td>▼ heart_demo (rank 17)</td> <td>99.569s</td> <td>21.650s</td> </tr> <tr> <td>▶ [Serial - outside any region]</td> <td>10.336s</td> <td>15.719s</td> </tr> <tr> <td>▶ make_rk_step\$omp\$parallel:</td> <td>21.290s</td> <td>1.418s</td> </tr> <tr> <td>▶ make_rk_step\$omp\$parallel:</td> <td>21.183s</td> <td>1.431s</td> </tr> <tr> <td>▶ make_rk_step\$omp\$parallel:</td> <td>21.239s</td> <td>1.320s</td> </tr> </tbody> </table>	Process / OpenMP Region / Function / Thread / Call Stack	Effective Time by Utilization	Spin Time	▶ heart_demo (rank 15)	99.641s	21.705s	▼ heart_demo (rank 17)	99.569s	21.650s	▶ [Serial - outside any region]	10.336s	15.719s	▶ make_rk_step\$omp\$parallel:	21.290s	1.418s	▶ make_rk_step\$omp\$parallel:	21.183s	1.431s	▶ make_rk_step\$omp\$parallel:	21.239s	1.320s														
Process / OpenMP Region / Function / Thread / Call Stack	Effective Time by Utilization	Spin Time																																		
▶ heart_demo (rank 15)	99.641s	21.705s																																		
▼ heart_demo (rank 17)	99.569s	21.650s																																		
▶ [Serial - outside any region]	10.336s	15.719s																																		
▶ make_rk_step\$omp\$parallel:	21.290s	1.418s																																		
▶ make_rk_step\$omp\$parallel:	21.183s	1.431s																																		
▶ make_rk_step\$omp\$parallel:	21.239s	1.320s																																		
<p><b>ボトルネックとなっているメモリー・オブジェクトを特定</b></p> <p>典型的な hotspot 解析では、最も多くの時間を費やしているコードが示されます。メモリーアクセス解析は、異なる観点を提供します。パフォーマンスの問題を引き起こし、帯域幅を消費しているメモリー・オブジェクトが示されるため、パフォーマンスを向上する方法について新しい情報が得られます。</p>	<table border="1"> <thead> <tr> <th>Memory Object</th> <th>Total Latency</th> <th>Loads</th> <th>Stores</th> <th>LLC Miss Count</th> </tr> </thead> <tbody> <tr> <td>alloc_test.cpp:157 ( 30 MB )</td> <td>65.6%</td> <td>4,239,327,176</td> <td>4,475,334,256</td> <td>0</td> </tr> <tr> <td>alloc_test.cpp:135 ( 305 MB )</td> <td>6.8%</td> <td>411,212,336</td> <td>441,613,248</td> <td>0</td> </tr> <tr> <td>alloc_test.cpp:109 ( 305 MB )</td> <td>6.3%</td> <td>439,213,176</td> <td>449,613,488</td> <td>0</td> </tr> <tr> <td>alloc_test!l_data_init.436.0.6 ( 576 B )</td> <td>5.2%</td> <td>742,422,272</td> <td>676,820,304</td> <td>0</td> </tr> <tr> <td>[vmlinux]</td> <td>4.6%</td> <td>173,605,208</td> <td>116,003,480</td> <td>0</td> </tr> <tr> <td>[Others]</td> <td>11.5%</td> <td>1,533,646,008</td> <td>1,674,450,232</td> <td>0</td> </tr> </tbody> </table>	Memory Object	Total Latency	Loads	Stores	LLC Miss Count	alloc_test.cpp:157 ( 30 MB )	65.6%	4,239,327,176	4,475,334,256	0	alloc_test.cpp:135 ( 305 MB )	6.8%	411,212,336	441,613,248	0	alloc_test.cpp:109 ( 305 MB )	6.3%	439,213,176	449,613,488	0	alloc_test!l_data_init.436.0.6 ( 576 B )	5.2%	742,422,272	676,820,304	0	[vmlinux]	4.6%	173,605,208	116,003,480	0	[Others]	11.5%	1,533,646,008	1,674,450,232	0
Memory Object	Total Latency	Loads	Stores	LLC Miss Count																																
alloc_test.cpp:157 ( 30 MB )	65.6%	4,239,327,176	4,475,334,256	0																																
alloc_test.cpp:135 ( 305 MB )	6.8%	411,212,336	441,613,248	0																																
alloc_test.cpp:109 ( 305 MB )	6.3%	439,213,176	449,613,488	0																																
alloc_test!l_data_init.436.0.6 ( 576 B )	5.2%	742,422,272	676,820,304	0																																
[vmlinux]	4.6%	173,605,208	116,003,480	0																																
[Others]	11.5%	1,533,646,008	1,674,450,232	0																																
<p><b>データ構造の最適化</b></p> <ul style="list-style-type: none"> <li>キャッシュミス (コード行だけでなく) データ構造に紐付け</li> </ul> <p><b>NUMA レイテンシーとスケラビリティの最適化</b></p> <ul style="list-style-type: none"> <li>共有とフォルス・シェアリングのチューニング</li> <li>ソケット間の帯域幅のチューニング</li> </ul> <p><b>Intel® Optane™ DC パーシステント・メモリー向けに設計および最適化</b></p>	<table border="1"> <thead> <tr> <th>Bandwidth Utilization Type / Function / Thread</th> <th>CPU Time</th> <th>L1 Bound</th> <th>DRAM Bound</th> </tr> </thead> <tbody> <tr> <td>▼ Low</td> <td>10.531s</td> <td>0.133</td> <td>3.3%</td> </tr> <tr> <td>▶ grid_intersect</td> <td>5.795s</td> <td>0.154</td> <td>3.3%</td> </tr> <tr> <td>▶ sphere_intersect</td> <td>3.282s</td> <td>0.102</td> <td>1.2%</td> </tr> <tr> <td>▶ shader</td> <td>0.135s</td> <td>0.109</td> <td>0.0%</td> </tr> <tr> <td>▶ tri_intersect</td> <td>0.059s</td> <td>0.301</td> <td>0.0%</td> </tr> <tr> <td>▶ func@0x1401513f0</td> <td>0.040s</td> <td>0.000</td> <td>58.0%</td> </tr> <tr> <td>▶ func@0x10009c00</td> <td>0.037s</td> <td>0.147</td> <td>0.0%</td> </tr> </tbody> </table>	Bandwidth Utilization Type / Function / Thread	CPU Time	L1 Bound	DRAM Bound	▼ Low	10.531s	0.133	3.3%	▶ grid_intersect	5.795s	0.154	3.3%	▶ sphere_intersect	3.282s	0.102	1.2%	▶ shader	0.135s	0.109	0.0%	▶ tri_intersect	0.059s	0.301	0.0%	▶ func@0x1401513f0	0.040s	0.000	58.0%	▶ func@0x10009c00	0.037s	0.147	0.0%			
Bandwidth Utilization Type / Function / Thread	CPU Time	L1 Bound	DRAM Bound																																	
▼ Low	10.531s	0.133	3.3%																																	
▶ grid_intersect	5.795s	0.154	3.3%																																	
▶ sphere_intersect	3.282s	0.102	1.2%																																	
▶ shader	0.135s	0.109	0.0%																																	
▶ tri_intersect	0.059s	0.301	0.0%																																	
▶ func@0x1401513f0	0.040s	0.000	58.0%																																	
▶ func@0x10009c00	0.037s	0.147	0.0%																																	

## 技術的な質問を直接問い合わせることができるプライオリティー・サポート

Intel® ソフトウェア開発製品の有償ライセンスには、購入日から 1 年間の Online Service Center でのプライオリティー・サポートが自動的に含まれます (Intel® ソフトウェア開発製品のプライオリティー・サポートは英語でのみ受け付けています)。サポートサービス期間はお得な価格で延長/更新できます。

- 製品の新しいアップデートおよび以前のバージョンへの**無料アクセス**。
- 機密の質問やコードサンプルを提出して**Intel のエンジニアに直接問い合わせ可能**。
- 製品ニーズと技術的な質問への**迅速な対応** (古いバージョンと新しいバージョン)。
- すべての Intel® ソフトウェア開発製品をカバーする**コミュニティ製品フォーラム**。
- 過去数十年のハイパフォーマンス・コード作成の経験を基に構築された**ドキュメント・ライブラリーへのアクセス**。

## 動作環境

プロセッサ	インテル® プロセッサと互換プロセッサ、インテル® Arria® 10
言語	C、C++、C#、Fortran、Java*、Python*、Go*、ASM、OpenCL* ほか
コンパイラ	Microsoft* コンパイラ、GCC、インテル® コンパイラ、OS 標準に準拠するその他のコンパイラで動作
開発環境	Microsoft* Visual Studio* 統合環境またはスタンドアロン
ホスト OS	Windows*、Linux*、macOS*
ターゲット OS	Linux*、Windows*、FreeBSD*、Android*、および一部の組み込み OS
スレッド解析	インテルによる OpenMP* 実装、インテル® TBB、ネイティブスレッド
MPI 並列処理	インテル® Trace Analyzer & Collector の MPI プロファイラと統合
GPU	最新のインテル® プロセッサ上で OpenCL* プログラムとメディア・アプリケーションをチューニング、OpenCL* カーネルの hotspot 解析

「インテル® VTune™ プロファイラの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。優れたスケーラビリティは、インテル® TBB と OpenMP\* 並列化手法の組み合わせを使用して得られたものです。以前のバージョンと比較してパフォーマンスを 8 コアで 8 倍以上、16 コアで約 11 倍向上できました。」

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



インテル® VTune™ プロファイラの製品情報 >

30 日間の無料評価版 >

性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサ用に最適化されていることがあります。SYSmark\* や MobileMark\* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行っただけです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。詳細については、<http://www.intel.com/performance> (英語) を参照してください。

インテル® コンパイラでは、インテル® マイクロプロセッサに限定されない最適化に関して、他社製マイクロプロセッサ用に同等の最適化を行えないことがあります。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® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサ依存の最適化は、インテル® マイクロプロセッサでの使用を前提としています。インテル® マイクロアーキテクチャに限定されない最適化のなかにも、インテル® マイクロプロセッサ用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。

インテル® ソフトウェア開発製品のパフォーマンスおよび最適化に関する詳細は、最適化に関する注意事項 (<https://software.intel.com/articles/optimization-notice#opt-jp>) を参照してください。

この文書および情報は、インテルのお客向けの参考情報として記載されているものであり、現状のまま提供され、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適合性、特定目的への適合性、知的財産権の非侵害性への保証を含みますが、これらに限定されるものではありません。本資料は、本資料に記述、表示、または記載された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。インテル製品は、医療、救命、延命措置、重要な制御または安全システム、核施設などの目的に使用することを前提としたものではありません。

Intel、インテル、Intel ロゴ、Arria、Intel Optane、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 またはその子会社の商標です。

\* 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。

インテル株式会社

〒100-0005 東京都千代田区丸の内3-1-1

<http://www.intel.co.jp/>

© 2019 Intel Corporation. 無断での引用、転載を禁じます。

JPN/1912/PDF/XL/CPDP/ND