x86 ハイブリッド・アーキテクチャー向けのソフトウェア最適化

ホワイトペーパー

2021年10月

リビジョン 1.0

注意事項:

この日本語マニュアルは、インテル コーポレーションのウェブサイトで公開されている『Optimizing Software for x86 Hybrid Architecture』の参考訳です。

ドキュメント番号: 348851-001JA

著作権と商標について: 本資料には、開発の設計段階にある製品についての情報が含まれています。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ないでください。

性能は、使用状況、構成、その他の要因によって異なります。詳細については、www.Intel.com/PerformanceIndex (英語) を参照してください。

性能の測定結果はシステム構成の日付時点のテストに基づいています。また、現在公開中のすべてのセキュリティー・アップデートが適用されているとは限りません。詳細については、公開されている構成情報を参照してください。絶対的なセキュリティーを提供できる製品またはコンポーネントはありません。

すべての製品計画とロードマップは、予告なく変更される場合があります。

開発コード名は、開発中で一般に公開されていない製品、テクノロジー、またはサービスを識別するためにインテルによって使用されます。それらは、"商用的な" 名称ではなく、商標としての機能を意図したものではありません。

実際の費用と結果は異なる場合があります。

インテル® テクノロジーの機能と利点はシステム構成によって異なり、対応するハードウェアやソフトウェア、またはサービスの有効化が必要となる場合があります。

本資料に記載されているインテル製品に関する侵害行為または法的調査に関連して、本資料を使用または使用を促すことはできません。本資料を使用することにより、お客様は、インテルに対し、本資料で開示された内容を含む特許クレームで、その後に作成したものについて、非独占的かつロイヤルティー無料の実施権を許諾することに同意することになります。

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。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。

インテルは、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、商品適格性、特定目的への適合性、および非侵害性の黙示の保証、ならびに履行の過程、取引の過程、または取引での使用から生じるあらゆる保証を含みますが、これらに限定されるわけではありません。

インテルは、本資料で参照しているサードパーティーのベンチマーク・データまたはウェブサイトについて管理や監査を行っていません。本資料で参照しているウェブサイトにアクセスし、本資料で参照しているデータが正確かどうかを確認してください。

目次

改訂履歴

リビジョン番号	説明	日付
1.0	ドキュメントの最初のリリース。	2021年10月

はじめに

この技術資料は、ハイブリッド・アーキテクチャーをサポートするインテル® Core™ プロセッサー向けにソフトウェアを最適化する情報を提供します。

この資料では、x86 ハイブリッド・アーキテクチャーの概要、Windows* でのハイブリッド・コアの利用法、ソフトウェア・アプリケーションおよびドライバーにおける最適なコアの利用方法について詳しく説明します。

また、x86 ハイブリッド・アーキテクチャーをサポートするインテル® Core™ プロセッサーで利用できるシステムのパフォーマンスと消費電力の目標を達成する Windows* 電力管理設定 (PPM 設定) についても説明します。

1. x86 ハイブリッド・アーキテクチャーの概要

1.1. パフォーマンス・ハイブリッド・アーキテクチャーをサポートする第 12 世代インテル® Core™ プロセッサー

パフォーマンス・ハイブリッド・アーキテクチャーをサポートする第 12 世代インテル® Core™ プロセッサーは、最大 8 つの Performance-core (P-core) と 8 つの Efficient-core (E-core) で構成されています。これらのプロセッサーには、IDI モジュールごとに 3MB の最終レベルキャッシュ (LLC) が含まれます。ここで、モジュールとは 1 つの P-core または 4 つの E-core を意味します。シンメトリックな ISA を備えており、さまざまな構成があります。

P-core はシングルスレッド、または制限付きスレッドのパフォーマンスをもたらし、E-core は改善されたスケーリングとマルチスレッド効率を提供します。これらのプロセッサー上の P-core では、インテル® ハイパースレッディング・テクノロジーを有効にすることもできます。OS がすべてのプロセッサーのスケジュールを決定すると、それらのコアを同時にアクティブにできます。

ハイブリッドを有効にする重要な OSV の要件は、パフォーマンス・ハイブリッド・アーキテクチャーにおける異なるコアタイプ間でのシンメトリックな ISA です。パフォーマンス・ハイブリッド・アーキテクチャーをサポートする第 12 世代インテル® Core™ プロセッサーでは、ISA は P-core と E-core 間の共通のベースラインに収束されます。

1.2. ハイブリッド・アーキテクチャーをサポートする第 11 世代インテル® Core™ プロセッサー

ハイブリッド・アーキテクチャーをサポートする第 11 世代インテル® Core™ プロセッサーは、共有 4MB LLC に CCF で接続された 1 つの P-core と 4 つの E-core クラスターで構成されています。E-core モジュールと P-core は、2 つの IDI チャンネルを介して CCF に接続され 4MB LLC を共有します。LLC はインクルーシブであり、すべてのコアは完全にコヒーレントです。

P-core は、インテル® ハイパースレッディング・テクノロジーが無効化された単一の物理コアです。OS は 5 つの物理コアを列挙して認識します。OS が 5 つのコアすべてのスケジュールを決定すると、それらのコアを同時にアクティブにできます。

ハイブリッドを有効にする重要な OSV の要件は、パフォーマンス・ハイブリッド・アーキテクチャーにおける異なるコアタイプ間でのシンメトリックな ISA です。

以下に、ハイブリッド・アーキテクチャーをサポートする第 11 世代インテル® Core™ プロセッサーの詳細を示します。

	詳細
プラットフォームの目的	• 新しいデュアル・スクリーン・デバイスのカテゴリーと体験。 • モバイル・フォームファクターとバッテリー残量。 • ワークロード: 軽負荷の生産性、ブラウズ、メディア再生、接続性、デュアルスクリーン。
ハイブリッドの構成と ISA	• 4 つの E-core + 1 つの P-core、4MB LLC、P-core の SMT は無効。 • 統合/共通 ISA に統合。 • ハイブリッド ISA (Perfmon、MCA) をデバッグ/パフォーマンス・チューニング・ツールに公開。
ハイブリッドのスケジューリング	• すべてのコア (Performance と Efficient) を OS に公開。 • P-core と E-core の同時実行とスケジューリング。 • 新しい Windows* ハイブリッド・スケジューリングにより IA ハイブリッドに対応 (HW ガイドによるスケジューリング)。

2. ハイブリッドのスケジューリング

2.1. ハードウェア・ガイドによるスケジューリング

ハードウェア・ガイドによるスケジューリングでは、ハードウェアは次の方法で OS に動的なフィードバック (ハードウェア・フィードバック・インターフェイスとも呼ばれます) を提供します。

- 電力/熱制限に基づく P-core および E-core の動的なパフォーマンスとエネルギー効率。
- 電力と熱が制限される場合のアイドリングのヒント。

ハイブリッド・アーキテクチャーをサポートするプロセッサーでは、すべてのコアが OS に公開されます。OS のスケジューラーは、ソフトウェア・スレッドをどのコアタイプにスケジュールするか決定する責任があります。

ハードウェアによるサポートは CPUIID 命令によって照会でき、MSR への書き込みによって OS が有効にします。

関連する CPUID:

- CPUID[6].EAX[19] – ハードウェアによるフィードバックのサポートを示します。
- CPUID[6].EDX[7:0] – サポートされる機能のビット。
- CPUID[6].EDX[11:8] – HGS テーブルのサイズ。
- CPUID[6].EDX[31:16] – HGS テーブル内の論理プロセッサーの行のインデックス。

行インデックス MSR の設定:

- IA32_HW_FEEDBACK_CONFIG MSR (17D1H)
  - ビット 0 – HGS を有効にします。

このテクノロジーの詳細については、www.intel.com/sdm (英語) にある『Intel® 64 and IA-32 Architectures Software Developer Manuals』を参照してください。

2.2. インテル® スレッド・ディレクター

インテル® スレッド・ディレクターを使用すると、ハードウェアは各種 IPC パフォーマンス特性に基づいて、次の形式でスレッドごとに OS へランタイム・フィードバック (拡張ハードウェア・フィードバック) を提供します。

- 電力/熱制限に基づく P-core および E-core の動的なパフォーマンスとエネルギー効率。
- 電力と熱が制限される場合のアイドリングのヒント。

インテル® スレッド・ディレクターは、第 12 世代インテル® Core™ プロセッサー・ファミリー以降のパフォーマンス・ハイブリッド・アーキテクチャーをサポートするインテル® Core™ プロセッサーで利用でき、OS がスレッドに適切なコアを選択できるようにします。

スレッド固有のハードウェアによるサポートは CPUIID 命令によって照会でき、MSR への書き込みによって OS が有効にします。

関連する CPUID:

- CPUID[6].EAX[23] – インテル® Enhanced Hardware Feedback のサポートを示します。
- CPUID[6].ECX[11:8] – インテル® スレッド・ディレクターのクラス数を示します。
- CPUID[0x20, ECX=0][EBX[0]] – HRESET 命令のサポートを示します。

MSR の設定:

- IA32_HW_FEEDBACK_CONFIG MSR (17D1H)
  - ビット 1 - インテル® スレッド・ディレクターのマルチクラスのサポートを有効にします。
- IA32_HW_FEEDBACK_THREAD_CONFIG MSR (17D4H)
  - ビット 1 - インテル® スレッド・ディレクターのマルチクラスのサポートを有効にします。

このテクノロジーの詳細については、www.intel.com/sdm (英語) にある『Intel® 64 and IA-32 Architectures Software Developer Manuals』を参照してください。

2.3. x86 ハイブリッド・アーキテクチャーに対応するプロセッサーで実現されるインテル® ハイパースレッディング・テクノロジーによるスケジューリング

E-core は、P-core でハイパースレッディングがビジーの論理コアよりも優れたパフォーマンスを発揮するように設計されています。

2.4. マルチ E-core モジュールによるスケジューリング

アイドル状態のモジュール内の E-core は、ビジー状態のモジュール内の E-core よりも優れたパフォーマンスを発揮します。

2.5. x86 ハイブリッド・アーキテクチャーにおけるバックグラウンド・スレッドのスケジューリング

ほとんどのシナリオでは、バックグラウンド・スレッドは、E-core のスケーラビリティーとマルチスレッド効率を活用できます。

3. Windows* の主要な電源とパフォーマンス機能

3.1. スレッド・スケジューリングの概要

x86 ハイブリッド・アーキテクチャーをサポートするプロセッサーは、ハードウェアと OS の共有メモリー領域に列挙されるパフォーマンスと効率に関する機能に基づいて分類されます。このメモリー領域は、OS によってハードウェアに設定および列挙され、このドキュメントではハードウェア・フィードバック・メモリー・テーブルと呼んでいます。メモリー領域はハードウェアによって更新され、更新されると OS に通知されます。

x86 ハイブリッド・アーキテクチャーをサポートするプロセッサーのスケジューリングは、QoS および優先順位主導であり、プリエンプティブな特性を持ちます。ほとんどのシナリオでは、スケジューリングの制約の中で QoS が最も高く、優先度の高いスレッドが最もパフォーマンスが高いコアで実行されます。

開発者は、電力とパフォーマンスを最適化するため、効率良い周波数とコアで実行するスレッドを選択できます。この動作は、QoS API を使用してスレッドの**電力スロットリング**を定義することで制御できます (6.4 節の API の詳細を参照)。

ハード・プロセッサーのアフィニティーは、OS の判断を混乱させる可能性があります。アフィニティーは、スレッドのパフォーマンスや効率を求めるために使用されることがありますが、パフォーマンスや効率は動的に達成されるものであり、コアタイプに基づくものではありません。そのため、ハード・アフィニティーでは望ましい結果が得られないことがあります。電力スロットリングを活用することで、パフォーマンスや効率が最適なコアにワークを誘導できます。どうしてもアフィニティーの制御が必要である場合、CPUSets API を使用します (詳細は 6.3 節を参照)。

パフォーマンス・ハイブリッド・アーキテクチャーをサポートする第 12 世代インテル® Core™ プロセッサーでは、ハードウェアが Windows* にスレッドクラス情報を提供することで、さらに高いパフォーマンスのスレッドをプロセッサーに配置できます。このヒントは、同一または下位の QoS/優先度を持つスレッドで使用されます。

5 節の特定のスケジューリングの例を参照してください。

3.2. Windows* コア・パーキング・エンジンの概要

一般に、Windows* のコア・パーキング・エンジンは、ワークロードに対してグローバルなスケーラビリティーを決定し、実行に最適な計算コアのセットを選択します。x86 ハイブリッド・アーキテクチャーをサポートするプロセッサーでは、さらに P-core と E-core の最適なセットを選択できます。

パフォーマンスと効率に関する情報は、ハードウェア・フィードバック・メモリー・テーブルから得られます。

Performance-core (P-core) では最もパフォーマンスが高いコアが最初にパーク解除され、Efficient-core (E-core) では最も効率が良いコアが最初にパーク解除されます。

x86 ハイブリッド・アーキテクチャーをサポートするプロセッサーには、2 つの高レベルパーキング構成があります。

- 1 つは、パフォーマンスのみでパーキングを決定する、標準パーキングまたは優先コアパーキング構成です。
- 2 つ目は、パフォーマンスと効率の両方に基づいてパーキングを決定する、ヘテロパーキング構成です。

これらの構成は、ヘテロポリシー PPM レジストリーおよび関連するパーキング PPM のしきい値を調整して制御できます。

5 節の特定の例と構成を参照してください。PPM に関する詳細は 7.2 節にあります。

3.3. パフォーマンス・ステートのコントロール・エンジン

ホモジニアス・システムと同様に、インテル® スピード・シフト・テクノロジー (HWP) は Windows* によって利用され、効率良いパフォーマンス・レベルで実行する必要があるスレッドのパフォーマンスを制限したり、システム全体のパフォーマンスとバッテリー残量の目標を達成するため、スライダーによってエネルギー・パフォーマンスの優先順位を調整したりします。

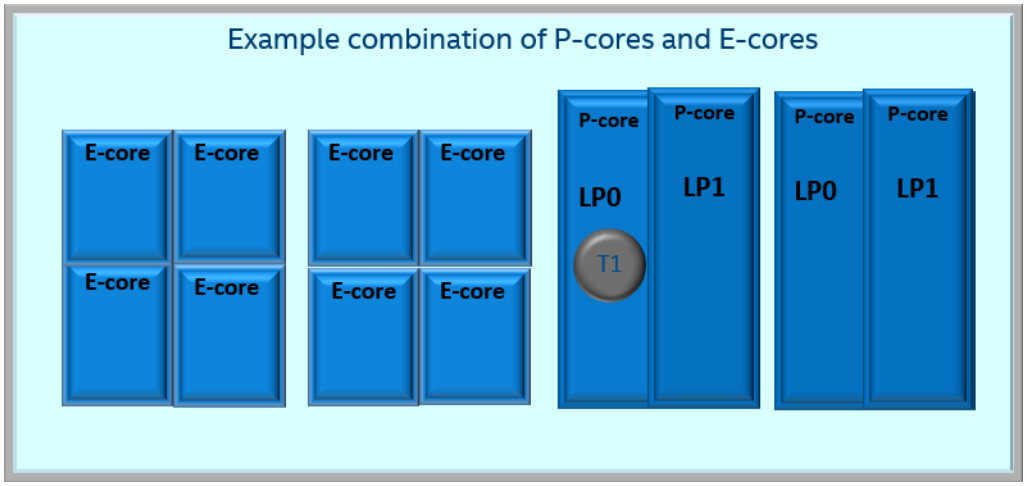
多様な PPM 設定 – *EnergyPerfPreference*、*MinimumPerformance*、*FrequencyCap*、*MaxPerformance* – は、異なるスレッドの QoS 要件を満たすため、異なる Windows* スライダーの位置、プロファイル、および Windows* によって動的に変更されます。

PPM の設定に関する詳細は 7.3 節にあります。

4. 第 12 世代インテル® Core™ プロセッサーにおける Windows* のスケジューリング/パーキングの例

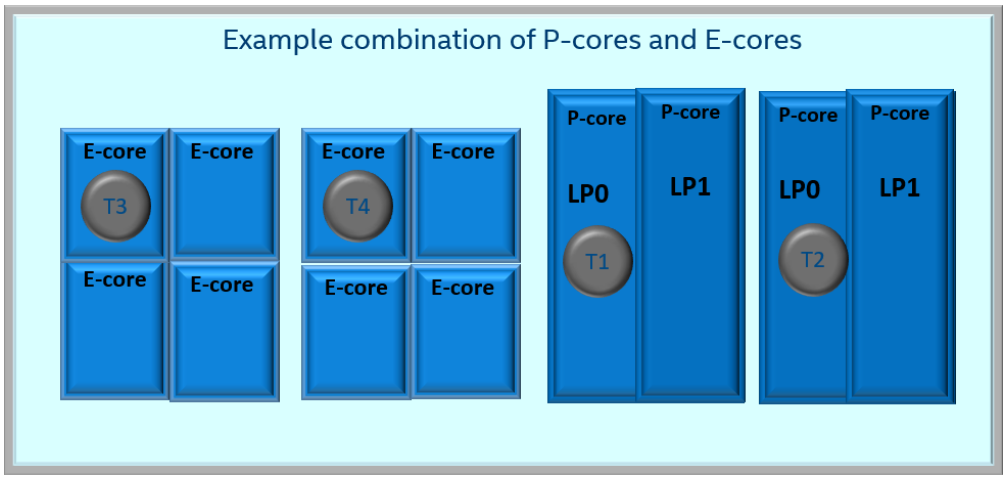
4.1. シングルスレッドのシナリオ

次の例では、Windows* がインテルのコアを活用してシングルスレッドのパフォーマンスを達成しています。この動作は、論理プロセッサー (LP) 0 が最高のパフォーマンスを備えている場合に動的に実現されます。



4.2. 制限付きスレッドシナリオの例 1

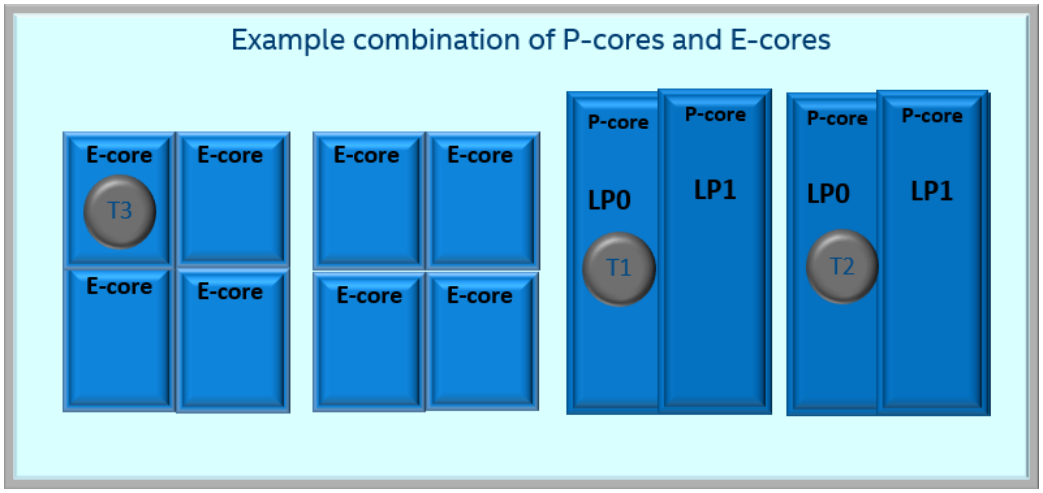
次の例は、制限付きソフトウェア・スレッドのシナリオにおけるスケジューリングの動作を示しています。この動作は、P-core が E-core よりも高いパフォーマンスを持つ場合、Windows* スケジューラー/パーキングエンジンによって動的に実現されます。E-core は、ビジーな P-core の SMT よりもパフォーマンスが高くなります。能力が動的に変化する場合、Windows* はそれを考慮して最適なスケジューリングを行います。



4.3. 制限付きスレッドシナリオの例 2

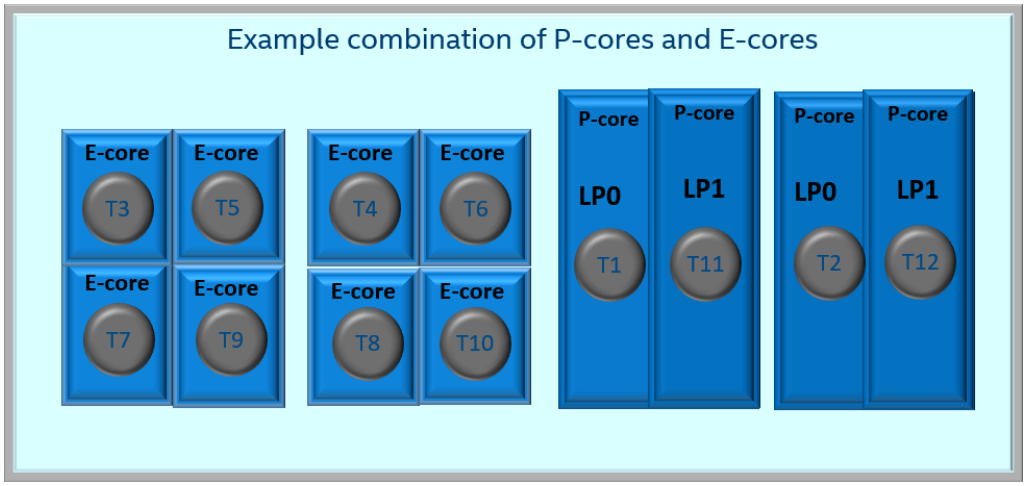
次の例では、T1 と T2 を P-core に、T3 を E-core に配置しています。この例では、T3 のパフォーマンスは T1 と T2 コアよりも低くなります。

能力が動的に変化する場合、Windows* はそれを考慮して最適なスケジューリングを行います。



4.4. マルチスレッドのシナリオ

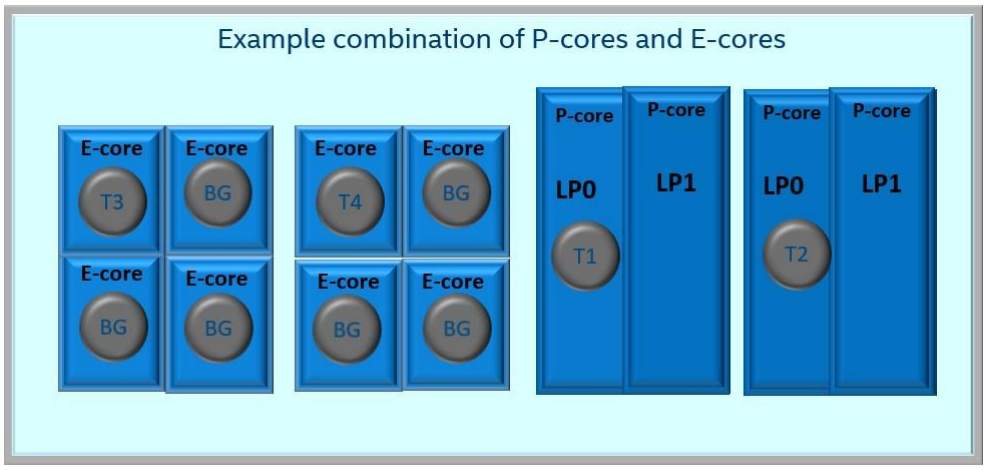
マルチスレッドのシナリオでは、Windows* によってすべてのコアが使用されます。



電力や熱の制約があるシステムでは、システムの最適なパフォーマンス/効率を得るため一部のコアが利用されない場合があります。この動作は、ハードウェアが Windows* にフィードバックを提供し、Windows* がフィードバックに基づいて動作することで動的に行われます。

4.5. バックグラウンド・スレッド

次の例では、Windows* がバックグラウンド・スレッドを E-core に動的に配置しています。ここでは、E-core が最も高いマルチスレッド効率を提供できます。



このオプションは、パフォーマンスを重視する構成では有効にならない場合があり、OEM に特化したチューニング向けに PPM パッケージで有効にできます。

この動作は、*CurrentScheme* に PPM が設定されている場合、デフォルトで有効になります。

a. *Background->SchedulingPolicy* は、3 もしくは 4 に設定されます。
b. *Background->SchedulingPolicy* が設定されていない場合、*default-> SchedulingPolicy* が 5 であるか確認します。

シナリオによっては、P-core がアイドル状態にある場合、P-core のバックグラウンド・アクティビティーを実行するほうが適切であることがあります。これは、Windows* コア・パーキング・エンジンが、どの P-core と E-core にスケジュールするか適切に判断することで実現されます。

4.6. マルチメディア・スレッド

バックグラウンド・スレッドと同様に、マルチメディア・スレッドも E-core に動的に配置されます。

このオプションは構成によっては無効化されている場合がありますが、OEM 固有のチューニング向けに PPM パッケージで有効にできます。

この動作は、*CurrentScheme* に以下の PPM 設定がある場合デフォルトで有効になります。

a. *MultiMediaQoS->SchedulingPolicy* は、3 もしくは 4 に設定されます。
b. *MultiMediaQoS->SchedulingPolicy* が設定されていない場合、*default-> SchedulingPolicy* が 5 であるか確認します。

シナリオによっては、フォアグラウンドを上回る QoS スレッドのコアが必要でない場合、コアでバックグラウンド・アクティビティーを実行するほうが効率良いことがあります。

4.7. エコ QoS スレッド

バックグラウンド・スレッドと同様に、エコスレッドも E-core に動的に配置されます。

このオプションは構成によっては無効化されている場合がありますが、OEM 固有のチューニング向けに PPM パッケージで有効にできます。

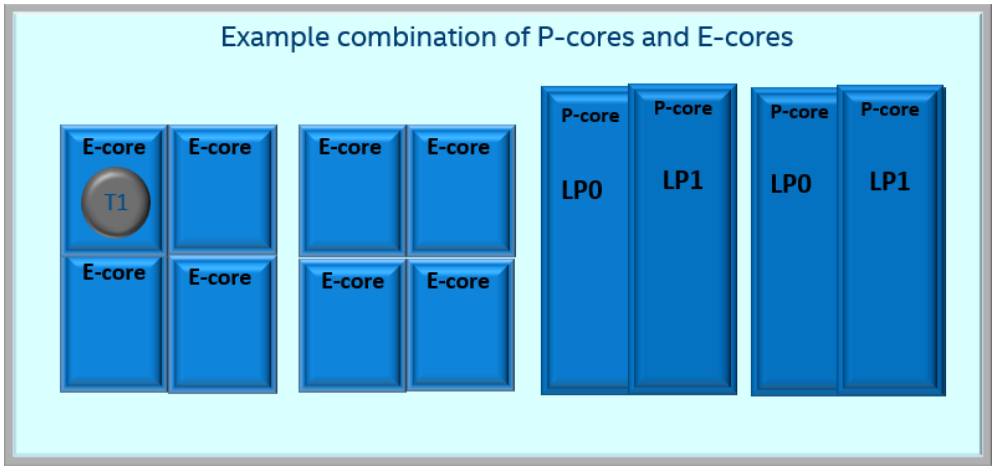
この動作は、以下のように PPM が設定されている場合デフォルトで有効になります。

a. *EcoQoS->SchedulingPolicy* は、3 もしくは 4 に設定されます。
b. *EcoQoS->SchedulingPolicy* が設定されていない場合、*default-> SchedulingPolicy* が 5 であるか確認します。

シナリオによっては、フォアグラウンドを上回る QoS スレッドのコアが必要でない場合、コアでバックグラウンド・アクティビティーを実行するほうが効率良いことがあります。

4.8. 利用率の低いスレッド

低消費電力のエンベロープ構成では、利用率が低いスレッドを E-core で実行するほうが望ましい場合があります。



一部の第 12 世代インテル® Core™ プロセッサーにおけるバッテリースライダーの設定は、Windows* がデフォルトでこの動作を有効にすることがあります。もしくは、OEM による PPM 設定 (*HeteroParking* ポリシー) を利用してこの動作を有効にできます。

注: この Windows* コアパーキング機能を有効にすると、利用率が低い重要またはクリティカルなスレッドにパフォーマンス・コストがかかる可能性があります。

4.9. Windows* のイベントトレース

4.9.1. インテル® スレッド・ディレクターはシステムで有効になっているか?

Windows* がインテル® スレッド・ディレクターを正常に有効化して活用しているか、Windows* イベントトレースから確認できます。これは、トレースログの “*microsoft-windows-kernel-processor-power*” イベントにある “AdvanceHgsEnabled” フィールドをチェックすることで行われます。

4.9.2. プロセス/スレッドの QoS を確認する方法

Windows* のイベントトレースの **CPU 利用率** -> **新しいスレッドの** *BAMQoSLevel* で、各プロセスのスレッドや QoS レベルを確認できます。Windows* QoS の詳細については 7.6 節をご覧ください。

5. 第 11 世代インテル® Core™ プロセッサーにおける Windows* のスケジューリング/パーキングの例

ハードウェアのガイド付きスケジューリングは、Windows* がさまざまな目標を達成するために使用されます。

1. 第 1 の目的は、ST 主体のワークロードのバースト・パフォーマンスを向上させることです。OS のスケジューラーはパフォーマンスが要求されるスレッドを識別し、E-core に比べて P-core のパフォーマンスが高い場合 (P-core は電力や熱の制約を受けない)、スレッドを P-core にスケジュールします。
2. 2 番目の目標は、すべてのシナリオにおけるワークロードとバックエンド・アクティビティーで E-core を利用することです。この場合、IA パフォーマンスの恩恵は少なく、エネルギー効率/バッテリー残量が主な目的です。これらのワークロードは、エネルギー効率が良い周波数で実行されることが期待されており、ほとんどの場合 E-core が P-core よりも効率的です。
3. 3 番目の目標は、GFX アプリケーションでエネルギー効率の良いパフォーマンスを実現することです。そのようなアプリケーションでは、GFX デバイスで十分な IA パフォーマンスを発揮しながら、GFX の電力バジェットを最大限に維持することが重要です。ほとんどの GFX アプリケーションでは、E-core がエネルギー効率良く IA パフォーマンスを提供しながら、GFX の電力バジェットを最大化します。これらのアプリケーションは E-core にスケジュールする必要があります。しかし、一部の GFX ワークロードは P-core で実行するほうが良いケースがあります。これは、ワークロードがより高い IA パフォーマンスを必要とする、あるいは E-core よりも P-core を効率良く実行する必要がある場合です。このようなケースでは、P-core にスケジュールすることで、アプリケーションのパフォーマンス要求を満たしつつ、GFX の電力バジェットを増やすことができます。

次の表は、さまざまなシナリオを想定したスケジューリングの例です。

シナリオ	スケジューラーの動作
ビデオ再生アプリケーションの起動 (プロファイルがサポートされます)	P-core はパークが解除され、重要なスレッド向けに常時すぐに起動できるようにしておきます。
ビデオ再生 (起動を除く) (セミアクティブ・シナリオ)	P-core は常にパークされます。ワークは、利用率が < 60% (パーク解除のしきい値*) の間 E-core にスケジュールされます。重要でないスレッドは P-core を使用しません。
ADK アプリケーションの起動 (プロファイルがサポートされます)	P-core はパークが解除され、重要なスレッド向けに常時すぐに起動できるようにしておきます。
CINEBENCH を最小化	最小化すると、CINEBENCH スレッドは E-core モジュールのみで実行されます。

シナリオ	スケジューラーの動作
wPrime 1 スレッド (フォアグラウンド/高い QoS)	シングルスレッドの wPrime は P-core で実行されます。
Geekbench マルチスレッド (MT) (フォアグラウンド/高い QoS MT)	5 コア (1 つの P-core + 4 つの E-core) と 4 コア (4 つの E-core) の比較により、MT パフォーマンスが ST よりも高くなりました。
デュアルスクリーン	高い QoS* とは異なる MediumQoS* をスケジュールする可能性があります。
ゲーム/グラフィックス	電力/熱の状況に応じて、ハードウェア・フィードバックにより P-core はパークされます。
スクリーンオフ	P-core は常にパークされます。E-core にのみスケジュールされます。

6. ソフトウェア・アプリケーション: x86 ハイブリッド・アーキテクチャーをサポートするインテル® Core™ プロセッサーに関する一般的な質疑と推奨事項

6.1. ソフトウェアから見える ISA の違いは?

いくつかの ISA の違いはソフトウェアから識別できます。以下に例を示します。

1. P-core と E-core の構造的な違い (TLB、キャッシュ、トポロジー)
2. パフォーマンス・モニタリング、インテル® プロセッサー・トレース、最終分岐レコード、MCA ISA など。
3. いくつかのモデル固有 MSR。

これらの違いがアプリケーション・ソフトウェアに影響しないと想定されます。

6.2. x86 ハイブリッド・アーキテクチャーをサポートするプロセッサーでアフィニティーを使用できるか?

一般に、ソフトウェアはプロセッサー・アフィニティーの設定を避けるべきです。プロセッサー・アフィニティーを設定すると、x86 ハイブリッド・アーキテクチャーをサポートするプロセッサーのパフォーマンス、または効率が損なわれる可能性があります。パフォーマンスと効率の決定は、ハードウェアから Windows* に列挙される現在のパフォーマンスと効率、および QoS、優先順位などさまざまな要因に基づいて動的に行われます。

アフィニティーは特定のシナリオで使用される場合があります。例えば、特定のプロセッサーでカウンターを読み取るような場合です。

アフィニティーを使用する必要がある場合は、*CPUSets* API – *SetProcessorDefaultCpuSets*、*SetThreadSelectedCpuSets* を呼び出し、OS の電源管理と互換性のある「ソフト」な方法でアプリケーションのアフィニティーを宣言します。

これは、アプリケーションの割り込みでも同様です。x86 ハイブリッド・アーキテクチャーをサポートするインテル® Core™ プロセッサー上の Windows* で P-core または E-core への割り込みを最適化するため、ドライバーは特定のプロセッサーに割り込みの親和性を持たせてはなりません。

6.3. Windows* のパワー・スロットリング API とは? アプリケーション開発者はハイブリッドのスケジューリングを制御できるか?

開発者は、Windows* API の *SetProcessInformation*、*SetThreadInformation* を使用して、P-core や E-core にワークを適切に誘導できます。

これらの API が *ProcessPowerThrottling* パラメーターとともに使用されていない場合、Windows* の自動メカニズムがデフォルトで起動されます。これにより、Windows* は E-core を活用できるスレッド (エコ QoS など) と P-core を活用できるスレッド (高い QoS など) を誤認識する可能性があります。開発者は、これらの API を使用することで、Windows* の分類を改善できます。これにより、効率/バッテリー残量が向上し、ファンノイズが軽減され、システムのパフォーマンスが向上する可能性があります。

SetProcessInformation 関数は次のように定義されています。

```
BOOL SetProcessInformation(
  HANDLE                    hProcess,
  PROCESS_INFORMATION_CLASS ProcessInformationClass,
  LPVOID                    ProcessInformation,
  DWORD                     ProcessInformationSize
);
```

次の例は、ProcessPowerThrottling を使用して SetProcessInformation を呼び出すことで、現在のプロセスでスロットリング・ポリシーを有効にする方法を示します。

```
PROCESS_POWER_THROTTLING_STATE PowerThrottling; RtlZeroMemory(&PowerThrottling,
sizeof(PowerThrottling)); PowerThrottling.Version =
PROCESS_POWER_THROTTLING_CURRENT_VERSION;

//
// 実行スピードのスロットリングをオンにします。ControlMask はメカニズムを選択し、
// StateMask はオン/オフするメカニズムを宣言します。
//
PowerThrottling.ControlMask = PROCESS_POWER_THROTTLING_EXECUTION_SPEED;
PowerThrottling.StateMask = PROCESS_POWER_THROTTLING_EXECUTION_SPEED;


SetProcessInformation(GetCurrentProcess(),
                      ProcessPowerThrottling,
                      &PowerThrottling,
                      sizeof(PowerThrottling));

//
// 実行スピードのスロットリングをオフにします。ControlMask はメカニズムを選択し、
// オフにするメカニズムを StateMask でゼロに設定します。
//
PowerThrottling.ControlMask = PROCESS_POWER_THROTTLING_EXECUTION_SPEED;
PowerThrottling.StateMask = 0;


SetProcessInformation(GetCurrentProcess(),
                      ProcessPowerThrottling,
                      &PowerThrottling,
                      sizeof(PowerThrottling));

//
// システムにすべての電力管理を任せます。メカニズムを制御しないため、
// ControlMask はゼロに設定されています。
```

```
//
PowerThrottling.ControlMask = 0;
PowerThrottling.StateMask = 0;

SetProcessInformation(GetCurrentProcess(),
                      ProcessPowerThrottling,
                      &PowerThrottling,
                      sizeof(PowerThrottling));
```

6.4. スレッド数

プロセッサー数に基づくスレッドの生成は、常に最適であるとは限りません。一部のコアは、ほかのコアよりもパフォーマンスが低いか、アイドル状態/パーク状態になっている可能性があります。特定のシナリオにおけるマルチスレッドのスケジューリングには潜在的な問題があり、ハイブリッドにのみ関連するものではありません。

異なるコア数を選択する際のトレードオフを理解することは、アプリケーションに理想的なスレッド数を特定するのに重要であると考えられます。

6.5. アクティブスピン

アクティブなスピンウェイトを行うスレッドは、重要なスレッドやクリティカルなスレッドのパフォーマンスに影響を及ぼす可能性があります。ソフトウェアはスピンウェイトの代わりに、UMWAIT、TPAUSE、または PAUSE など軽量のスピンを利用できます。

6.6. ドライバー開発者は最適な割り込み制御が可能か?

6.6.1. 割り込みポリシーは設定しない

ドライバーは、割り込みポリシーとして IrqPolicyMachineDefault を使用して、Windows* の割り込み処理を選択する必要があります。これにより、デフォルトで Windows* は割り込みを発生するコアに割り込みを誘導し、コア・パーキング・エンジンと連携できるようになります。同一コア上で割り込みを処理することで、アイドル状態のコアを起動するレイテンシーや、割り込みを処理するメタデータを別のコアに移行するレイテンシーを回避し、レイテンシーとパフォーマンスの向上につながります。さらに、アイドルコアへの割り込みを回避することで、バッテリー残量にも良い影響を与えます。

6.6.2. アーキテクチャーとの関連性

Windows* では、プロセッサーの関係を照会するため、アプリケーション・ソフトウェアは、ユーザーモード API の GetLogicalProcessorInformationEx() または、カーネルモード API の KeQueryLogicalProcessorRelationship() を使用できます。

これらの API は、ドライバーによってプロセッサーごとのキューが割り当てられている場合、ドライバーを利用して最適なキューを割り当てることができます。

7. ハイブリッドの主な PPM 設定と設定値

7.1. Windows* の電源管理設定

Windows* 10 October 2018 update (RS5) 以降には、これらの設定に関してハイブリッドを活用する追加の拡張機能があります。

a. パフォーマンス状態エンジンの設定。
b. コア・パーキング・エンジンの設定。
c. パフォーマンス固有の制御。

7.2. コア・パーキング・エンジンの設定

コア・パーキング・エンジンは、ワークロードに対してグローバルなスケーラビリティーを決定し、実行に最適な計算コアのセットを選択します。x86 ハイブリッド・アーキテクチャーをサポートするプロセッサーでは、さらに P-core と E-core の最適なセットも決定します。

7.2.1. ヘテロポリシー

`Common\Power\Policy\Settings\Processor\HeteroPolicy`

7.2.1.1 設定値: 0 (標準パーキングや優先コアパーキングなど)

この構成では、最適な計算コアのセットが、最もパフォーマンスが高いコアから順番にパークが解除されます。

パークを解除する最適なコア数の決定に関連する PPM 設定:

- *CPMinCores*: 常にパーク解除可能な論理プロセッサーの最小パーセンテージを、パフォーマンスが高いコアから順に指定します。論理プロセッサーとは、各 NUMA ノード内のシステムで有効化されるすべての論理プロセッサーを指します。
- *CPMaxCores*: 常にパーク解除可能な論理プロセッサーの最大パーセンテージを、パフォーマンスが高いコアから順に指定します。論理プロセッサーとは、各 NUMA ノード内のシステムで有効化されるすべての論理プロセッサーを指します。
- *CPIncreaseTime*: パークされた最もパフォーマンスが高い論理プロセッサーが、追加でパークされた状態からパークが解除されている状態に遷移するまでの最小時間を指定します。時間には、プロセッサーのパフォーマンス時間をチェックする間隔の数を単位として指定します。
- *CPDecreaseTime*: パークが解除されている最もパフォーマンスが低い論理プロセッサーが、追加でパークが解除されている状態から、パークされた状態に遷移するまでの最小時間を指定します。時間は、プロセッサーのパフォーマンス時間をチェックする間隔の数を単位として指定します。
- *CPConcurrency*: ノードの同時性を判断するしきい値を指定します。

- *CPDistribution*: ユーティリティーを分散する最もパフォーマンスが高い論理プロセッサー数を選択するため、同時実行分散で使用する利用率をパーセンテージで指定します。この数は、パークを解除するように選択された論理プロセッサー数より少ないことがありますが、上回ることはありません。
- *CPHeadroom*: パークが解除されている一連のプロセッサーの中で最も利用率の低いプロセッサーの利用率が高い場合、コア・パーキング・エンジンが追加で最もパフォーマンスが高いパーク済みの論理プロセッサーのパークを解除する要因となる利用率値を指定します。この設定により、同時実行性の増加を検知できます。
- *CpLatencyHintUnpark*: システムの低レイテンシー・ヒントが検出された場合に、パークが解除されたコアの最小数 (最もパフォーマンスが高いパークされたプロセッサーから開始) を指定します。

7.2.1.2 設定値: 4 (ヘテロパーキングなど)

この構成では、利用率に基づいて、最もパフォーマンスが高いコア、または最も効率の良いコアの組み合わせが最初にパーク解除されます。

低消費電力の SKU やバッテリー残量の向上など特定のシナリオでは、効率良い周波数で、効率良い能力のコアで、利用率の低いワークを実行するほうが優れていることがあります。このポリシーは、これらのシナリオで最適なパフォーマンス状態のエンジン設定と組み合わせて使用されます。

パークを解除する最適なコア数の決定に関連する PPM 設定:

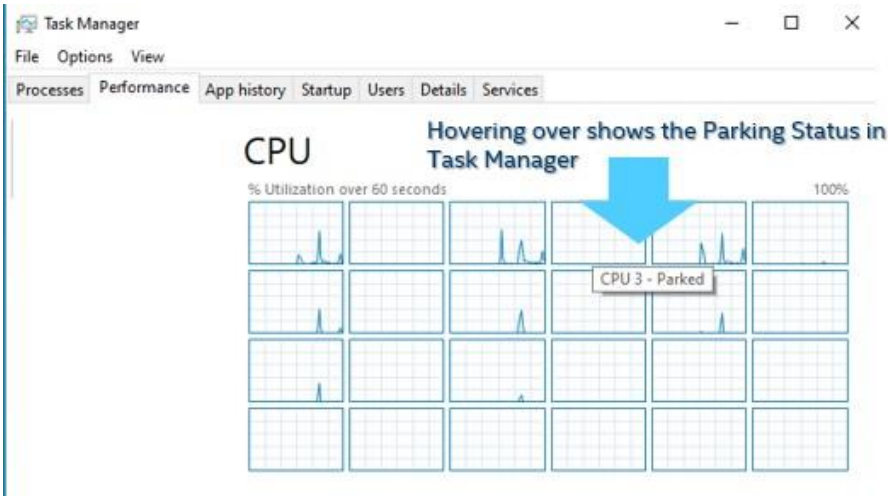
- *CPMinCores*: 任意の時点でパーク解除可能な論理プロセッサーの最小パーセンテージを指定します。論理プロセッサーとは、各 NUMA ノード内のシステムで有効化されるすべての論理プロセッサーを指します。
- *CPMaxCores*: 任意の時点でパーク解除可能な論理プロセッサーの最大パーセンテージを指定します。論理プロセッサーとは、各 NUMA ノード内のシステムで有効化されるすべての論理プロセッサーを指します。
- *CPIncreaseTime*: パーク状態の最も効率の良い論理プロセッサーが、追加でパーク状態からパーク解除された状態に遷移するまでの最小時間を指定します。時間には、プロセッサーのパフォーマンス時間をチェックする間隔の数を単位として指定します。
- *CPDecreaseTime*: パーク解除されている最も効率の低い論理プロセッサーが、追加でパーク解除されている状態からパーク状態に遷移するまでの最小時間を指定します。時間には、プロセッサーのパフォーマンス時間をチェックする間隔の数を単位として指定します。
- *CPConcurrency*: ノードの同時性を判断するしきい値を指定します。
- *CPDistribution*: ユーティリティーを分散する最もパフォーマンスが高い論理プロセッサー数を選択するため、同時実行分散で使用する利用率をパーセンテージで指定します。この数は、パークを解除するように選択された論理プロセッサー数より少ないことがありますが、上回ることはありません。

- *CPHeadroom*: パーク解除されている一連のプロセッサーの中で最も利用率の低いプロセッサーの利用率が高い場合、コア・パーキング・エンジンが追加で最もパフォーマンスが高いパーク済みの論理プロセッサーのパークを解除する要因となる利用率値を指定します。この設定により、同時実行性の増加を検知できます。
- *CpLatencyHintUnpark*: システムの低レイテンシー・ヒントが検出された場合に、パーク解除されたコアの最小数 (最もパフォーマンスが高いパークされたプロセッサーから開始) を指定します。
- *HeteroIncreaseThreshold*: N 番目にパフォーマンスが高いパークされたコアをパーク解除するために超えなければならないしきい値を指定します。コア・インデックスごとに個別の値になります。
- *HeteroDecreaseThreshold:* N 番目にパフォーマンスが低いパーク解除されたコアをパークするために超えなければならないしきい値を指定します。コア・インデックスごとに個別の値になります。
- *HeteroIncreaseTime:* パークされた Performance-core (最もパフォーマンスが高いパークされたプロセッサーから開始) が追加でパーク解除されるまでの最小時間を指定します。
- *HeteroDecreaseTime:* プロセッサー (パフォーマンスが最も低いパークが解除されているプロセッサーから開始) が、追加でパーク解除されている状態からパーク状態に遷移するまでの最小時間を指定します。
- *HeteroClass1InitialPerf*: パフォーマンスのためパーク解除されているプロセッサーの最初のパフォーマンス・パーセンテージを指定します。
- HeteroClass0FloorPerf: 少なくても 1 つのプロセッサーがパフォーマンス向けにパークが解除されている場合、E-core プロセッサーの下限パフォーマンスのパーセンテージを指定します。

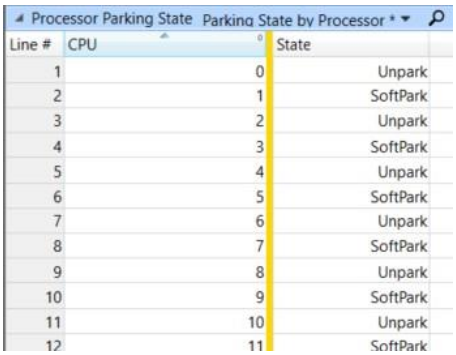
7.2.2. プロセッサーのパーキングステートを確認

プロセッサーのパーキングステートはさまざまなツールで確認できます。

7.2.1.3 ツールの例 #1: タスク・マネージャー



7.2.1.4 ツールの例 #2: Windows* パフォーマンス・アナライザーによる Windows* のトレースイベント



Processor Parking State　Parking State by Processor *

Line #	CPU	State
1	0	Unpark
2	1	SoftPark
3	2	Unpark
4	3	SoftPark
5	4	Unpark
6	5	SoftPark
7	6	Unpark
8	7	SoftPark
9	8	Unpark
10	9	SoftPark
11	10	Unpark
12	11	SoftPark

7.3. パフォーマンス・ステートの設定

HWP 対応システムでは、パフォーマンス・ステートはハードウェア制御のパフォーマンス・ステート (つまり HWP) を介して調整されます。HWP ハードウェア・エンジンに最適なパフォーマンス・ステートを決定するため各種設定があります。

7.3.1. PerfEnergyPreference

E-core の EPP（Energy Performance Preference）レジスターにプログラムする値を指定します。値はパーセンテージで指定し、0 - 255 のスケールで Windows* がレジスターに設定します。

HWP システムでは、単一論理プロセッサーの EPP レジスターは 774H[ビット 31:24] です。

7.3.2. PerfEnergyPreference1

P-core の EPP (Energy Performance Preference) レジスターにプログラムする値を指定します。値はパーセンテージで指定し、0 - 255 のスケールで Windows* がレジスターに設定します。

HWP システムでは、単一論理プロセッサーの EPP レジスターは 774H[ビット 31:24] です。

7.3.3. MinimumPerformance

E-core の最小パフォーマンス・レジスターにプログラムする値を指定します。値はパーセンテージで指定し、0 - 255 のスケールで Windows* がレジスターに設定します。

HWP システムでは、単一論理プロセッサーの EPP レジスターは 774H[ビット 7:0] です。

7.3.4. MinimumPerformance1

P-core の最小パフォーマンス・レジスターにプログラムする値を指定します。値はパーセンテージで指定し、0 - 255 のスケールで Windows* がレジスターに設定します。

HWP システムでは、単一論理プロセッサーの EPP レジスターは 774H[ビット 7:0] です。

7.3.5. MaximumPerformance

E-core の最大パフォーマンス・レジスターにプログラムする値を指定します。値はパーセンテージで指定し、0 - 255 のスケールで Windows* がレジスターに設定します。

HWP システムでは、単一論理プロセッサーの EPP レジスターは 774H[ビット 15:8] です。

7.3.6. MaximumPerformance1

P-core の最大パフォーマンス・レジスターにプログラムする値を指定します。値はパーセンテージで指定し、0 - 255 のスケールで Windows* がレジスターに設定します。

HWP システムでは、単一論理プロセッサーの EPP レジスターは 774H[ビット 15:8] です。

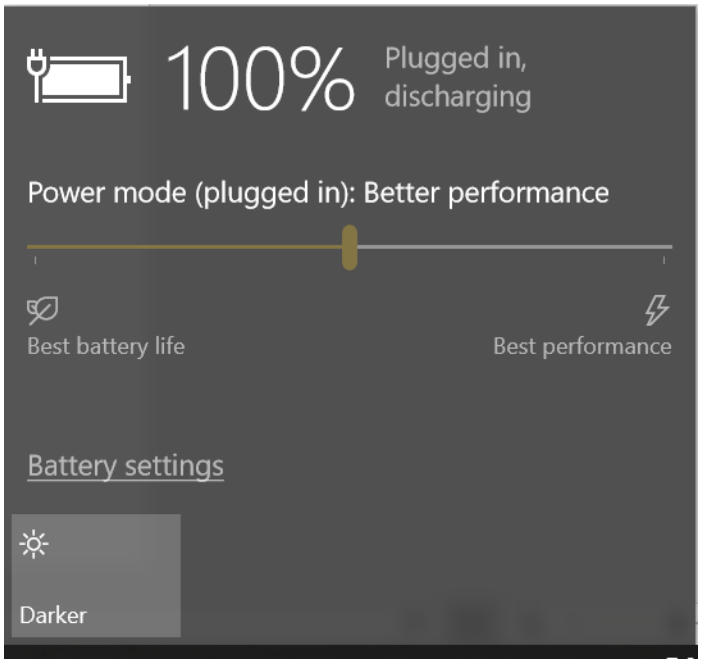
7.4. Windows* パフォーマンス電源スライダー

Windows* パフォーマンス電源スライダーは、利用者がシステムのパフォーマンスを迅速かつインテリジェントに変更して、バッテリー残量を引き延ばせるようにします。利用者が 4 つのスライダーモードを切り替えて、パフォーマンスとバッテリー駆動時間を調整すると、Windows* の電源設定が背後で動作します。

用意されているスライダーの位置は以下です。

a. 最大のバッテリー残量 (Best Battery)
b. 高パフォーマンス (High Performance)
c. 最も高いパフォーマンス (Best Performance)

次の画像は、利用者が利用できる Windows* パフォーマンス電源スライダーのスナップショットであり、それぞれの位置にスライダーを切り替えることができます。



次の場合、スライダーの位置によって PPM の設定が変更されます。

a. PerfEnergyPreference は、スライダーの位置ごとに異なり、バッテリー重視のスライダーの位置で高い値を示します。
b. QoS は、プランの設定が異なり、バッテリー駆動時間や電力重視の QoS/プランではスロットリング重視の設定 (PerfEnergyPreference を高くする) になります。

7.5. 主要な電源プラン

Windows* のパフォーマンス電力スライダーと同様に、Windows* の電源設定も電源プランごとに異なる設定が可能です。次のセクションでは、Windows* がサポートする各種プロファイルと推奨される設定を示します。

7.5.1. デフォルトのプロファイル

デフォルトのプロファイルは、ほとんどの時間がアクティブである設定セットです。これらの電源設定は、現在の電源方式の設定と同一です。

7.5.2. 低レイテンシー

低レイテンシーはブート時やアプリケーションの起動時に有効になるプロファイルです。

7.5.3. 低消費電力

低消費電力は、メディア再生シナリオのバッファリング・フェーズで有効になるシナリオです。

7.5.4. スクリーンオフ

スクリーンオフは、最近のスタンバイシステムで使用されるプロファイルです。このプロファイルは、システムが長期間スリープフェーズに入る場合に使用されます。システムの休止動作がすべて完了し、オーディオが再生されておらず、モバイル・ホットスポットが利用されない状態です。このプロファイルはシステムがスリープ状態から復帰すると解除されます。

7.6. QoS、HWP、およびハイブリッドのスケジューリング

7.6.1. デフォルト

この動作は、デフォルトおよび高い QoS (または重要なスレッド) に特化して最適化されています。また、ほかの QoS が定義されていない場合、ほかの QoS スレッドにも影響を与えます。

- 例: Default->EnergyPerfPreference は、スライダーの位置によって異なる値を保持でき、パフォーマンスとバッテリー残量の目標を達成するのに役立ちます。

7.6.2. オクルージョンされたウィンドウ

PPM の設定により、オクルードされたウィンドウ HWP の構成と、デフォルト/高い QoS スレッドとは異なるスケジューリング・ポリシーを適用できます。

7.6.3. バックグラウンド・スレッド

デフォルトでは次の PPM 設定により、デフォルトでバックグラウンド・スレッドに特化して動作が最適化されています。

a. Background->EnergyPerfPreference の値が Default->EnergyPerPreference の値と異なります。
b. Background->MaximumPerformance の値が default->MaximumPerformance の値と異なります。
c. Default->FrequencyCap はゼロ以外の値に設定されます。
d. Background->SchedulingPolicy の値は default->SchedulingPolicy の値と異なります。
e. Background ->SchedulingPolicy が設定されていない場合、default-> SchedulingPolicy は 5 に設定されます。

一部の構成 (最良のパフォーマンスのためのスライダーの位置) では、バックグラウンド・スレッドのスロットルを避けるため、バックグラウンドの設定をデフォルトと同じにすることが最適である場合があります。この動作の設定は次のようになります。

a. Background->EnergyPerfPreference = default->EnergyPerPreference。
b. Background->MaximumPerformance = default->MaximumPerformance。
c. Default->FrequencyCap は設定されていません。

d. Background->SchedulingPolicy = default->SchedulingPolicy。
e. Default-> SchedulingPolicy を 2 に設定。

7.6.4. マルチメディア・スレッド

デフォルトでは、次の PPM 設定により、マルチメディア・スレッドに特化して動作が最適化されています。

a. MMQoS->EnergyPerfPreference は default->EnergyPerPreference とは異なる値を持ちます。
b. MMQoS->EnergyPerfPreference は default->MaximumPerformance と比べて異なる値を持ちます。
c. MMQoS->SchedulingPolicy の値は default->SchedulingPolicy の値とは異なります。
d. MMQoS->SchedulingPolicy が設定されていない場合、Default->SchedulingPolicy は 5 に設定されます。

注: バックグラウンド・スレッドと同様に、一部の設定 (例えば、最良のパフォーマンス・スライダーの位置) では、マルチメディア・スレッドのスロットルを避けるため、マルチメディア設定をデフォルトと同じにすることが最適である場合があります。この動作では、設定は次のようになります (例; 最良のパフォーマンス・スライダーの位置)。

a. MMQoS ->EnergyPerfPreference = default->EnergyPerPreference。
b. MMQoS ->MaximumPerformance = default->MaximumPerformance。
c. MMQoS >SchedulingPolicy = Default->SchedulingPolicy。
d. Default-> SchedulingPolicy を 2 に設定。

7.6.5. エコスレッド

Windows* のパワースロットル API である、*SetProcessInformation*、*SetThreadInformation* を使用して、アプリケーション・スレッドやプロセスのパフォーマンス、もしくは効率の目標を Windows* が動的に検出することを拒否できます。開発者が効率的な動作を考慮してマークしたスレッドは、エコ QoS スレッドと呼ばれます。

PPM の観点からは、これらの設定によりエコ QoS スレッド専用に動作が最適化されます。

a. EcoQoS->EnergyPerfPreference は default->EnergyPerPreference と比べて高い値を持ちます。
b. EcoQoS->MaximumPerformance は default->MaximumPerformance と比べて低い値を持ちます。
c. EcoQoS->SchedulingPolicy の値は default->SchedulingPolicy と異なります。
d. EcoQoS->SchedulingPolicy が設定されていない場合、default-> SchedulingPolicy は 5 に設定されます。