



CITO Research
Tell Us a Question.

2010 年 7 月

マルチコアのジレンマ

ホワイト ペーパー

提供

PERVASIVE®

目次

はじめに.....	1
開発者とハードウェア:歴史的観点.....	2
マルチコアによる複雑なパフォーマンス インパクト	2
マルチコアによってパフォーマンスが向上する状況.....	4
マルチコアによってパフォーマンスが低下する状況.....	5
マルチコアの課題.....	7
マルチスレッド アプリケーションの実行速度がマルチコア上で遅くなる原因	8
マルチスレッド アプリケーションでよく見られる問題	9
OS の役割.....	11
開発者の選択肢.....	12
Pervasive PSQL™ v11 について.....	13
結論.....	14

はじめに

新しいシステムを購入すれば、パフォーマンスが向上します。ラップトップからサーバーへ、という発想も今や一般に広く通用しています。事実、パフォーマンスの悪化がハードウェアの購入を検討するきっかけとなることは多いでしょう。

しかし、マルチコア プロセッサについてはどうでしょう。ハイエンド サーバーでは長く使用されてきたこれらのプロセッサが、PC サーバーやデスクトップ システムにも搭載されるようになりました。ここで問題なのは、新しいハードウェアを購入さえすればパフォーマンスが確実に向上する、と中小企業の購入者が期待していることです。しかし、アプリケーションのパフォーマンスとマルチコア プロセッサの相互作用は非常に複雑で、アプリケーション スタックおよびオペレーティング システム スタックの多数のレイヤーに影響を与えます。業務アプリケーションによっては、マルチコア プロセッサでパフォーマンスが向上しないものもあります。それどころか、ある特定のアプリケーションではマルチコア プロセッサによって実行速度が遅くなってしまいます。

CITO Research はこの問題に気づき、これを解決するための方法を検討しました。このホワイト ペーパーでは、マルチコア プロセッサが、ある特定の業務アプリケーションのパフォーマンスにどのような悪影響を与えるのか、そして、この問題を軽減するために、その業務アプリケーションの開発者が得られる選択肢について説明します。

このホワイト ペーパーでは、以下のトピックについて取り上げます。

- マルチコア プロセッサとは
- マルチコア プロセッサによってアーキテクチャの変更が必要な理由
- アプリケーションによってはマルチコア プロセッサ上で実行速度が遅くなる原因
- マルチコア プロセッサ上で実行速度が遅くなるアプリケーションに共通する特徴
- マルチコア プロセッサ上で実行速度が速くなるアプリケーションに共通する特徴
- マルチスレッドおよび並列アプリケーションの定義
- マルチスレッド アプリケーションの実行速度がマルチコア プロセッサ上で頻繁に遅くなる原因
- マルチコア プロセッサが開発者に及ぼす影響
- マルチコア プロセッサ上でパフォーマンスを向上させるために開発者ができること

このホワイト ペーパーの著者について

CITO Research 社の CTO(最高技術責任者)兼編集者である Dan Woods(ダン・ウッズ)は、2009 年に Forbes.com のコラム記事の中で、マルチコア プロセッサのパフォーマンス インパクトおよび大量データの処理を加速させる方法について書き始めました。Pervasive はマルチコア プロセッサの潜在的なパフォーマンス インパクトについて認識を高めるため、当著者のスポンサーとなっています。Pervasive の解説については、このホワイト ペーパーの最後の節をご覧ください。

開発者とハードウェア：歴史的観点

初期のころ、プログラマーはアセンブリ コードで直接記述するハードウェア レベルの命令を使用してソフトウェアを作成していました。プログラマーは常にプロセッサのアーキテクチャを意識していました。オペレーティング システムと高級言語によって、基本的にはこの負担が軽減されました。開発者は Basic、COBOL、C、Pascal または Fortran などの言語でコードを記述し、そのコードをコンパイラでコンパイルして機械可読形式の命令に翻訳することができるようになったのです。

並列プログラミングには依然として専門的なスキルが必要でした。オペレーティング システムを作成したプログラマーがコードを並列化する必要がある一方、アプリケーション プログラマーはアプリケーションのほとんどの部分を並列化する必要がありませんでした。

現在まで話を進めましょう。より速いコンピューティングへの欲求はとどまるところを知りません。AMD やインテルなどのチップ開発者は、クロック速度の限界を押し上げてきました（速度をさらに加速させていけば、チップは過熱し焼けてしまうでしょう）。シングル チップ上に複数のコアを追加することが、演算処理能力を向上させる最も簡単な方法です。これはまさに、「クロック速度が速くなることでソフトウェアの実行速度が必ず速くなるのであれば、コアを増やすことで同じ効果が得られるのか？」という核心的な疑問につながります。

マルチコアによる複雑なパフォーマンス インパクト

ユーザーがマルチコア対応のノート PC で複数のプログラムを実行する場合、以前に比べプログラム間の切り替えが速く、より多くのプログラムを一度に実行できると感じられるかもしれません。しかし、個々のプログラム レベルで見ると、パフォーマンスが向上するかどうかはまったくわかりません。開発者は、マルチコア システム上で 1 つのアプリケーションのパフォーマンスを向上させるために、複数のコアを同時に利用できる並列化コードを記述する必要があります。

多くのアプリケーションがマルチスレッド化されているにもかかわらず、ほとんどの場合、スレッドの作業は完全に独立した領域に限定されています。たとえば、ユーザー インターフェイスは利便性を向上させるためにスレッド化されることが多いですが、アプリケーションのロジックは概して一連のシリアル、順次処理で構成されており、簡単には並列化させることができません。したがって、まず必要なのは、マルチコア システムの使用によるパフォーマンス向上のための並列処理に対応するなんらかの可能性が、アプリケーションの中に存在していなければならないということです。並列処理を考慮することなく設計および作成されたアプリケーションでこれを見いだすことは困難でしょう。

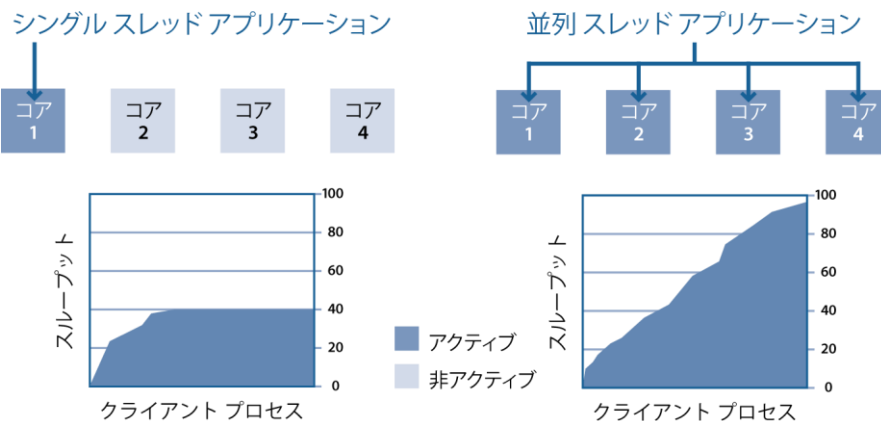


図 1: シングル スレッド アプリケーション vs 並列スレッド アプリケーション

もう 1 つの障害は必要とされるスキルです。マルチコア プロセッサに対応する効果的なプログラミングは簡単なことではなく、マルチスレッド プログラミングの経験を積んでいる開発者でさえ新たな挑戦に直面します。

アプリケーション開発者には 2 つの選択肢があります。

- 並列処理向けにコードを再設計する(再設計を目的に作られたツールキットや開発環境を使用)
- 基盤となるデータベースなど、この問題の解決に役立つアプリケーション スタックのコンポーネントをスワップアウトする

これらの選択肢は、実際には互いに矛盾するものではありません。最終的には、マルチコアの世界で最大限のパフォーマンスを得るため、開発者はマルチコア プロセッサを活用できるようアプリケーションを再設計しなければならないでしょう。基盤となるデータベースなど、アプリケーション スタックの要素をスワップアウトすることが可能であればパフォーマンスは向上するので、開発者はマルチコア ハードウェアを活用できるアプリケーションを再作成するための時間を稼ぐことができます。

マルチコアのジレンマを分析する上で課題となるのが、一般的なアドバイスを見つけるのが困難であるということです。特定のアプリケーション(またはその一部)をリファクタリングする必要性がどれほど急務であるかを判断するためには、アプリケーション、チップ アーキテクチャおよびオペレーティング システムのいくつかの側面を理解する必要があります。

CITO Research

Tell Us a Question.

- すべてのマルチコア プロセッサがまったく同じというわけではありません。チップの主要メーカーであるインテルや AMD は、マルチコア プロセッサの設計に異なるアプローチをとっています。ご自身が選択したプロセッサのタイプにより、マルチコア チップのアーキテクチャの影響を受ける可能性があります。アプリケーションの実行が、インテルで改善されることもあれば、AMD で改善されることもあります。
- 並行スレッドを処理する能力は、オペレーティング システムのリリースによって異なります(たとえば Windows Server 2008 タスク スケジューラーは、Windows Vista および Windows Server 2008 の両方で大幅な改善が見られます)。
- マルチコア プロセッサ上のアプリケーションの実行速度は、そのアプリケーションのニーズによって速くなることもあれば、遅くなることもあります。

このような問題の特徴を覚えておいてください。マルチコアのジレンマについてさらに詳しく説明していきます。

マルチコアによってパフォーマンスが向上する状況

ある状況においては、マルチコア処理によってパフォーマンスを劇的に向上させることができます。複数のアプリケーションを同時に実行することを考えてみましょう。デスクトップ システム上で、ビデオ ゲームのプレイと最新のビデオ プロジェクトのエンコードを同時に行うといった状況は、マルチコア プロセッサの影響が明確に現れる身近な事例です。

アプリケーションによってはマルチコア処理にうまく対応します。アプリケーションの 1 つの作業が自然に分割され、並列に実行し、操作の最後に集約することができれば、そのアプリケーションはマルチコア処理の恩恵を受けることができます(ただし、このように実行するようアプリケーションをリファクタリングする必要があり、場合によってはアプリケーションの動作を並列化するのに役立つツールキットを利用することになるでしょう)。そのようなアプリケーションの例として、ビデオおよび音声処理、科学および財務モデリング アプリケーション、CAD レンダリングのようなプロセッサ集中型のタスクが挙げられます。多くの高パフォーマンスおよび高スループット コンピューティング システムはこの概念を極限まで利用し、コンピューター アニメーションのレンダリングなどの問題を多数のチャンクに分割して、巨大なコンピューターファームに分散させます。

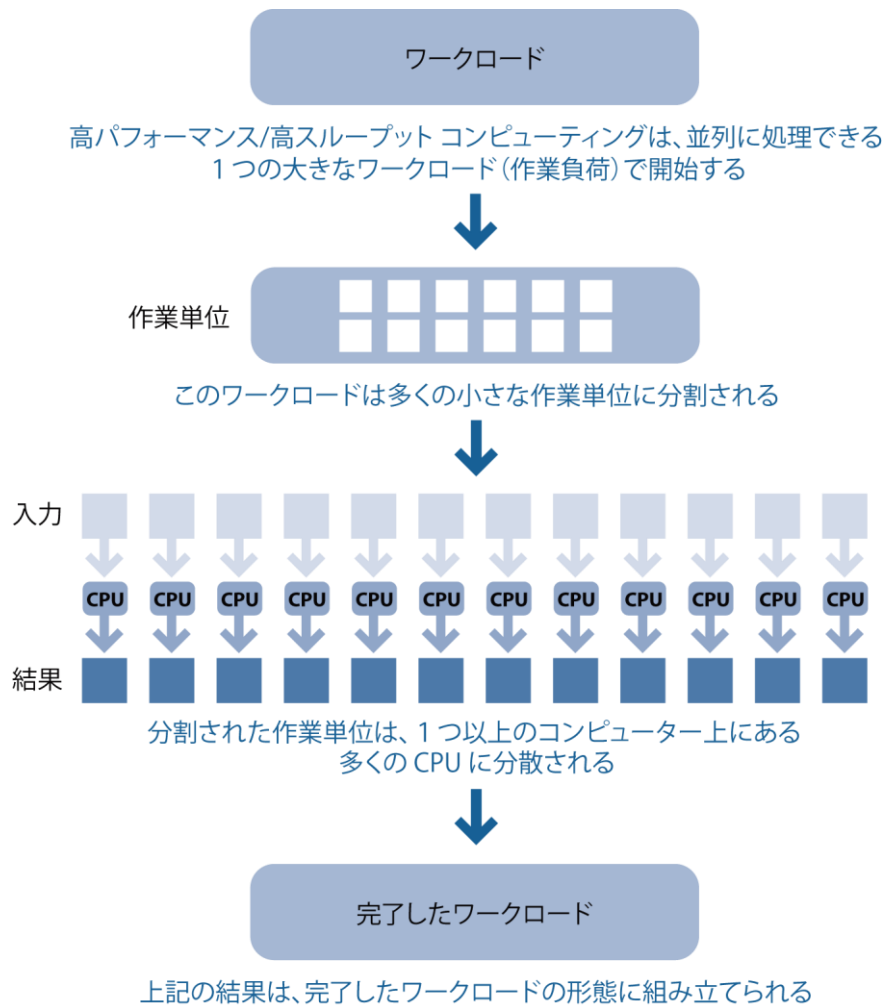


図 2: 高パフォーマンス/高スループット コンピューティングの基本構造

マルチコアによってパフォーマンスが低下する状況

ソフトウェア開発コミュニティでは、アプリケーションが次のような状況の場合にマルチコア プロセッサによってパフォーマンスが悪化する、ということがよく理解されていません。

- 一度に多くの情報をメモリに保持する必要があるため、操作を十分に分割することができない
- データベースへのアクセスなど、同時に多くのユーザーが同じことを行う

CITO Research

Tell Us a Question.

- データベースを使用しそのデータの状態に依存して結果を出す
- データを共有する

このような場合、マルチコア プロセッサ上での実行によってパフォーマンスが低下する可能性があります。

操作を分割することが難しい事例としては、まず動的な会計モデルが挙げられます。このモデルでは実行時にユーザーによってパラメーターが変更でき(そのモデルのビューをスライス & ダイスするためなど)、それと同時に入力もリアルタイム データ フィードから入ってきます(在庫または商品価格など)。

ネット ショップは(うまくいけば)多くの同時ユーザーを持ちます。ホリデー ショッピング シーズンには、特定のアイテムに対する需要が高くなります。在庫の有無の照会、商品アイテムをショッピング カートへ追加、および注文の確認といった操作によって特定のリソースに競合が生じ、パフォーマンスの問題や顧客の不満を引き起こします。そのようなアプリケーションの場合、マルチコア処理は問題があるかもしれません。

データベースには相互の依存関係が複雑な多くのスレッドと多くの同時処理があります。これは処理によってはほかの処理を待つ必要があるということで、それらの処理の並列化を困難にしています。データの状態を慎重に管理し、トランザクションの整合性を保証する必要があります。多くの場合、複雑なマルチユーザー アプリケーションにはデータベースが伴います。そのようなアプリケーションは、マルチコア プロセッサにおいてパフォーマンスの低下が起こりやすくなります。

アプリケーションの特徴

アプリケーションを変更しない場合の、マルチコアにおけるパフォーマンス傾向

同時に実行している複数のデスクトップ アプリケーション。	パフォーマンスは向上
複数の独立した処理ステップに分割することができるアプリケーション(ゲーム、シミュレーション、モデリング、レンダリング)。このような状況では、データは静的であり処理は完了するまで中断または更新することなく実行できます。	パフォーマンスは同じ、あるいは向上
データベースを使用するアプリケーション。これは、トランザクションの整合性を維持する必要があり、多くのユーザーが存在します。	パフォーマンスは低下
データを共有し多くのユーザーを持つアプリケーション。	パフォーマンスは低下

表 1: アプリケーションの特性とマルチコア プロセッサ上のパフォーマンス

CITO Research

Tell Us a Question.

一般的に、データを共有するマルチユーザー アプリケーションは業務アプリケーションなので、以下のような影響が予想されます。

- デスクトップ ユーザーは、マルチコア プロセッサによるプラス効果が得られる
- 小規模ビジネス ユーザーは、Microsoft Office などの製品アプリケーションの実行時に、マルチコア プロセッサによるプラス効果が得られる
- 小規模ビジネス ユーザーは、事業を運営するために使用するマルチクライアント アプリケーションのパフォーマンスが、マルチコア プロセッサ上で著しく低下することがわかる

中小規模の企業は次々に ISV に対して不満の声を上げ、問題解決に役立つ支援を要求しています。ISV における過去 2 年間のサポート電話について分析してみると、その内容に 1 つの傾向が見られます。マルチコア ハードウェアにアップグレードしたら、今まで問題なく実行していたアプリケーションのパフォーマンスが低下した、というものです。このように、顧客側でのハードウェア アップグレードが ISV 側でサポート問題となります。

マルチコアの課題

ここまで述べた全体像から、PC ハードウェアにおける大きな変化がソフトウェア産業にもたらす問題を明確に理解できるようになりました。マルチコア プロセッサによって、特定のアプリケーションの実行速度が速くならないどころか遅くなることもあるのです。開発手法、さらにデバッグやテストに関しても根本的に変更されています。この速度低下が起こる理由、そしてその対処策をさらに詳しく見てみましょう。

マルチコアも 1 つの対処策として考えられます。適切な調整を行うことで処理速度が上がるソフトウェアであれば、マルチコア プロセッサの利点を活用し、パフォーマンスを大幅に向上させることができます。場合によっては、データベースなどアプリケーション スタックの一部をスワップアウトすれば問題を解決できることもあります。これならアプリケーション開発者がすぐにアプリケーションの変更を行う必要はありません。開発者は、このアプローチをリスクが低い方法として考え、アプリケーションの作り直しという大きな課題に取り組むまでの時間を稼ぐことができます。図 3 では 3 つの選択肢を示します。

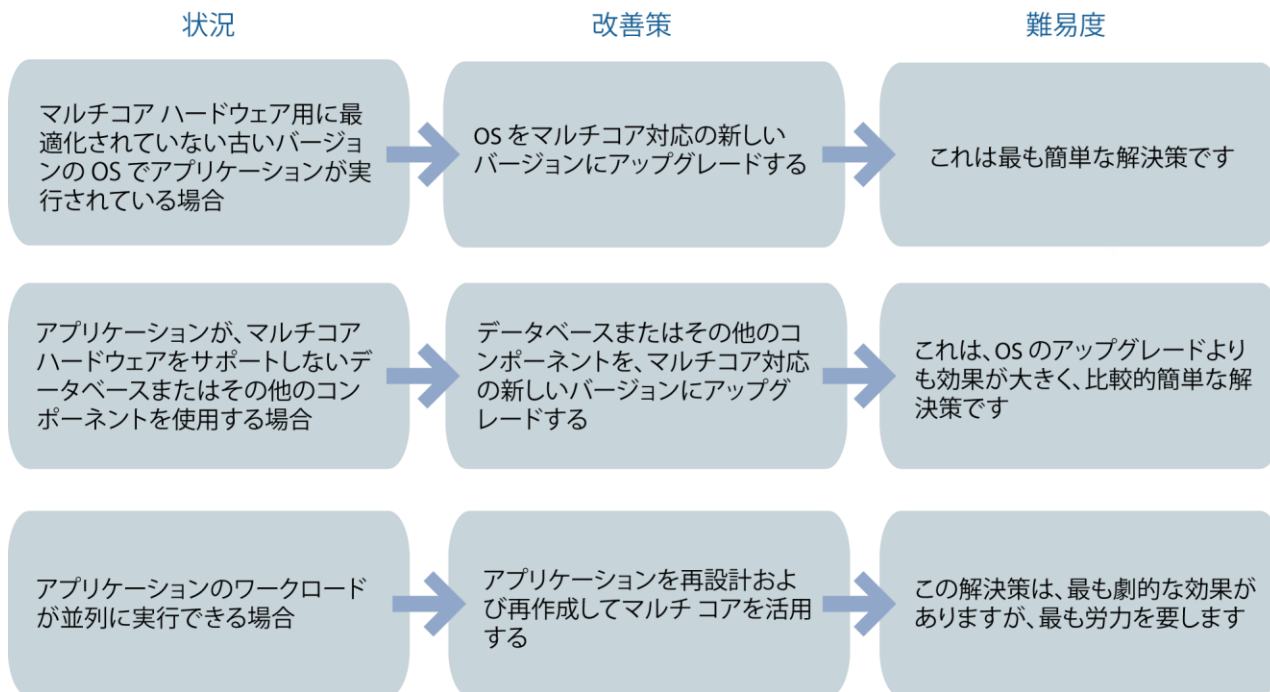


図 3: 並列処理向けにアプリケーションを最適化するための決定マトリックス

マルチスレッド アプリケーションの実行速度がマルチコア上で遅くなる原因

最初は、マルチスレッド アプリケーションの実行速度がマルチコア システム上で遅くなるということが信じられないかもしれませんが、この原因は複雑なので、詳しい説明が必要でしょう。

データを共有するマルチスレッド アプリケーションの場合、マルチコア マシン上ではスレッド間の同期によってシステムのリソースを大量に消費します。事実、データの共有は並列コンピューティングの悩みの種となっています。複数のスレッドが同じデータにアクセスする場合、それらのアクセスを同期させる必要があります。データへのアクセスを同期させる場合、そのコード部分は複数スレッドによる実行が行えないため、並行(同時)とはなりません。このコード部分は、通過するためにすべての人が並ぶ必要があるたった 1 つのドアとなります。

CITO Research

Tell Us a Question.

キャッシュではこの問題を改善できません。キャッシュは逆に事態を悪化させてしまいます。複数のコアまたはプロセッサに同じデータを指すキャッシュがある場合、1 つのコアがそのデータを変更すると、別のコアにキャッシュされたデータは無効になります。このため、キャッシュを同期させる必要があります。

そのような操作をすべて同期させるために蓄積されるオーバーヘッドは相当に大きく、結局のところ、そのような同期を必要としないシングルコア マシンよりもマルチコア マシンの方がパフォーマンスが低下してしまう可能性もあります。

可能であれば、各コアが別々のデータを処理するような設計にしてください。そうしないと、同期に伴うオーバーヘッドがあるため、そのオーバーヘッドによってパフォーマンスが大きく低下してしまうことがあります。

マルチスレッド アプリケーションでよく見られる問題

この移行でプログラマーが直面する困難はほかにもあります。マルチコア システム上の並列プログラミングには昔から以下のような問題が存在します。

- Thundering Herd(大群)
- 誤った共有
- メモリの競合

これらの問題を検証してみましょう。

Thundering Herd(大群)

Thundering Herd(大群)問題は、スケジューラーが 1 つのイベント(Web 接続など)を処理する可能性があるすべてのスレッドをスリープ解除し、その中から対象とするスレッドを特定し、それ以外のすべてのスレッドは実行対象外と判断されときにスリープ状態へ戻されるような場合に発生します。

消防署を例にして説明しましょう。マルチスレッド スケジューラーで、プロセッサが 1 つのタスクを終了し、作業を受け入れる準備ができたときに火災警報が鳴ります。待機しているすべてのスレッドは、仮眠室で就寝中の消防士に例えることができます。出動にあたり、全消防士を起床させておいて、支度させ消防車に乗り込む人員を特定するといった行為は非常に効率が悪いです。特に多くのスレッドがある場合は、次に実際に出動する人を特定する処理が行えないので、当然のことながらパフォーマンスが低下します。この結果は「スラッシング」と呼ばれることもあります。

マルチコアでは、スケジューラーがさらに多くのプロセッサに対応する必要があるので、マルチスレッド アプリケーションでよく見られるこの Thundering Herd 問題がマルチコア システムで一層深刻になります。

誤った共有

マルチスレッド プログラミングの利点の 1 つは、複数のタスクを同時に実行できることです。これが適切に動作するためには、タスクはメモリからデータを検索する際、キャッシュ ラインの優先順位を付ける必要があります。たとえデータが異なるものであっても、メモリ内のデータの位置は選択されるキャッシュ ラインに影響します。アプリケーションの観点では 2 つのスレッドが別々のデータを探しているとしても、それらのスレッドはメモリの同じキャッシュ ラインを目指して争い始め、時間がかかる競い合いの中、リソースがやり取りされます。マルチコア処理の利点を活用するには、このようなキャッシュの競合が起こらないようなアプリケーションを作成する必要があります。

メモリの競合

マルチコアを活用するよう作成されたプログラムは、アプリケーションがメモリから情報を取得する際の同時処理を認識しています。マルチコア向けに最適化されていないプログラム(率直に言えば、ほとんどのプログラム)は連続的に情報へアクセスするため、"メモリの競合" と呼ばれる状況に陥ります。この場合、プロセッサは繰り返しキャッシュをチェックしてデータが最新であることを確認します。このチェック手続きは、あるプロセッサのスレッドが、別のプロセッサの別スレッドによって作成された古いデータに対して実行されないようにすることを目的としています。この手続きによってエラーは回避されますが、正当性を保証するためにメモリ バスが新しいリクエストを中断するのでマシンの速度は大幅に低下します。

プロセッサを追加してもこの問題は解決しません。実際には、より多くのプロセッサがメモリ キャッシュに対して要求を出すので事態はもっと悪化するでしょう。互いに検証しなければならないキャッシュが多ければ多いほど、トラフィックが多くなります。キャッシュがなくても、複数のスレッドが中央メモリへアクセスしようとするので、互いに競合して処理が進まなくなってしまうでしょう。

マルチスレッド アプリケーションは、きめの粗いロックをきめの細かいロックに分割し、ロードを分散し、またメモリなどの共有リソースに対するスレッド間の競合を減らすようリファクタリングする必要があります。

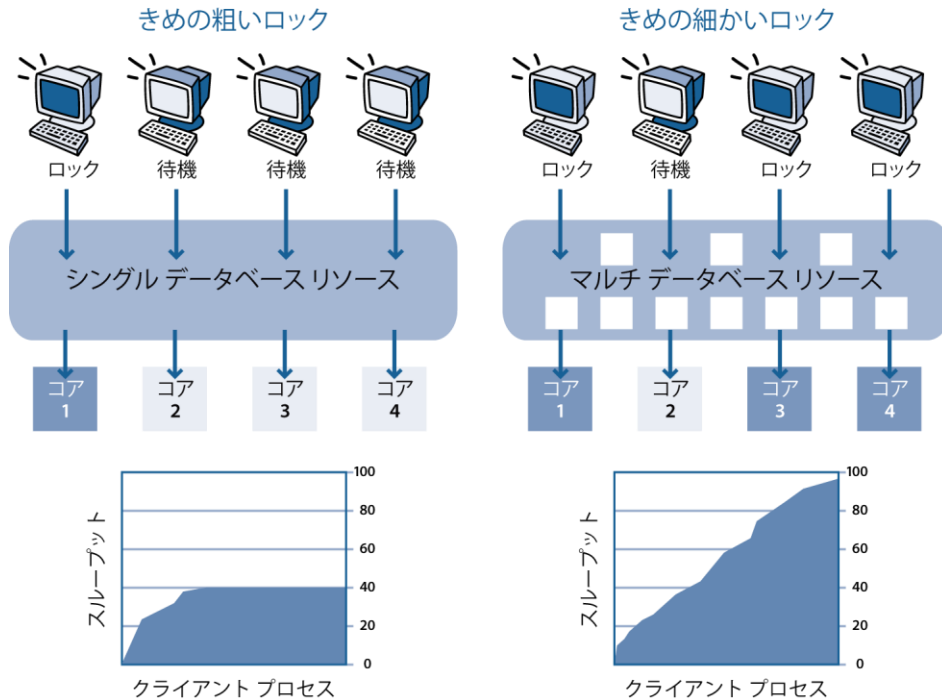


図 4:きめの粗いロックからきめの細かいロックへ移行した場合のスループットの向上

OS の役割

リソースの競合が発生した場合、オペレーティング システムがその競合の解決にあたります。CITO Research が開発者に話を聞いたところ、オペレーティング システムもマルチコア システム上でスレッドの競合を処理する速度が遅くなるように見えるそうです。つまり、マルチコア システム上のオペレーティング システムでは、競合ポイントの解決に時間がかかるということです。これは Windows Server 2008 や Windows Vista であっても同様です。それ以前の Windows オペレーティング システムでは、マルチコア システム上での速度がさらに低下します。

これはマルチコア オペレーティング システムの神話に一石を投じるものです。アプリケーションがオペレーティング システムに対し、単独ファイルで複数のタスクを実行する方法を今もなお要求している場合は、マルチコア用に最適化されたオペレーティング システムをインストールしても問題を解決することはできません。

オペレーティング システムを交通量の多い交差点で交通整理する警察官と考えてみましょう。各自動車(各スレッド)は一刻も早く進みたいでしょうが、警察官の指示を待っています。警察官が、次にどちらの車列を進行させるかについて独自に判断し、ある一定間隔で車列に対し通行指示を出すのではなく、停止線より後方に止まって待っている各ドライバーに対し進行する準備ができていかどうかを尋ねてから通行指示を出さなければならない、という状況を想像してください。

CITO Research

Tell Us a Question.

警察官がいなければ、もっとひどい交通渋滞が起きるでしょう。この交通渋滞は基本的に、マルチコア処理用の命令が組み込まれていないアプリケーションからの要求をオペレーティング システムが受けたときに起こるものです。当然のことながら、ユーザーはオペレーティング システム レベルの交通渋滞をアプリケーションのパフォーマンスの問題としてとらえます。

マルチコア用に最適化されたアプリケーションは、共有リソースの管理方法およびリソースへのアクセス優先度について、オペレーティング システムへ指示を与えることができます。これが多くの ISV が言うところのアプリケーション機能 "Lock-Free な実装" で、情報の要求はキャッシュ ラインを争ったり中央メモリにアクセスしたりしないよう整理されます。

開発者の選択肢

ここまで述べてきたすべての問題を踏まえ、ISV が行うことは何でしょうか。最終的には、マルチコア マシンにおいて最高レベルのパフォーマンスで実行されるようアプリケーションを再設計する必要があります。この再設計を支援するため、Microsoft の Parallel FX Library といったツールキットが登場しました。これには PLINQ (Visual Studio 2010 で導入)、インテル Parallel Studio および Open MPI が含まれています。これらのツールキットは開発者に対し、アプリケーションを再構築するための並列認識フレームワークを提供します。Google Go など新しい言語も開発されています。

問題は、この作業に時間がかかるという点です。場合によっては数年を要することもあります。ISV はそれぞれ、この課題を解決する方法を考え、お客様に悪影響を与えないようにする必要があります。

場合によっては、データベースなどアプリケーション スタックの一部をスワップアウトすることができます。これにより、マルチコアのジレンマを解決する簡単な方法が得られるので、将来的には(間違いなく普及するであろう)強力なマルチコア システムを活用するアプリケーションの作成に取り組んでください。Pervasive PSQL あるいは (ISAM のように) Pervasive PSQL へ簡単に移行できるタイプのデータ ストレージを使用している場合、マルチコア向けに最適化されたバージョンの Pervasive PSQL へ移行すれば、ほとんどの場合において再コンパイルやファイルの変換を行わなくてもアプリケーションのパフォーマンスを向上させることができます。

Pervasive PSQL™ v11 について

Pervasive はこのホワイトペーパーの著者である CITO Research 解説者のスポンサーとなっています。なぜなら、Pervasive はマルチコアによる問題がまだ業界で十分に理解されていないと感じ、また、特定クラスの ISV に対してはソリューションを持っているからです。

マルチコアのジレンマに対し、Pervasive はマルチコア システムをフル活用するため、そのデータベース システムである Pervasive PSQL v11 の最適化に重点的に取り組みました。

最新バージョンの Pervasive PSQL は、タスクの実行時にデータベースがマルチコアを取り入れることで、複数の類似動作を実行できる並列スレッドを提供するよう設計されています。この結果、マルチコア環境で、データベース エンジンのマルチユーザー パフォーマンスが向上します。コアを追加することにより、マルチコア動作が大幅に改善されるだけでなく、**データベースのパフォーマンスも向上します**。Pervasive における独自テストおよびお客様の環境では、Pervasive PSQL v10 と比較してパフォーマンスが概して 50% 上回り、アプリケーションによっては 400% 向上することもありました。

以下に、Pervasive PSQL v11 について、そしてそれがどのようにマルチコアのジレンマを解消するのか簡単に述べます。

- Pervasive PSQL v11 は、マルチコア プロセッサを活用し、ほとんどのマルチユーザー アプリケーションの実行速度を劇的に加速させる、真の並列実装です。
- Pervasive PSQL v11 は、トランザクショナル インターフェイスにおける低レベルの同期メカニズムについても機能強化しています。複数のユーザーが、キャッシュされた同じファイル ページを同時に読み込むことができ、またユーザーの操作は別々のコアで処理することができます。
- チェックポイントやログ管理などの非ユーザー操作には、さらに別のコアを使用することもできます。
- データベース エンジンにおける多くのボトルネックが除去された、あるいは減少しました。たとえば、複数のユーザーがそれぞれ別々のファイルにアクセスする場合、それらの操作を別々のコアで処理することができます。また、データベース エンジンは、以前に比べ少ないオーバーヘッドで多くのユーザー ロードを処理することもできるので、より安定したスループットが実現します。
- アプリケーションの再コンパイルは必要ありません。既存の Pervasive PSQL アプリケーションは変更することなく Pervasive PSQL v11 で実行します。

PSQL を導入すれば、速度低下の回避、あるいは十分なパフォーマンス向上の提供によって、すぐにマルチコア ハードウェアにおけるマルチユーザー パフォーマンスが向上します。これため、並列バージョンのアプリケーションが直ちに必要となることはありません。

Pervasive PSQL v11 を導入する理由はほかにも次のような事が挙げられます。

- 非常に速い埋め込みデータベースです。
- 優れた ISAM パフォーマンスとディスク IO 管理を提供します。

CITO Research

Tell Us a Question.

- パフォーマンスの向上を必要とする COBOL アプリケーション向けの非常に理想的なデータベースとなる可能性があります。
- 64 ビット SQL エンジンのほか 64 ビット ODBC および ADO.NET プロバイダーも含まれます。
- Pervasive PSQL または Btrieve に基づくすべてのアプリケーションに適用可能です。

Pervasive PSQL をデータベースとして使用するアプリケーションの開発者にとっては、PSQL v11 へアップグレードすれば、データベース レイヤーをスワップアウトするだけでマルチコアにおけるパフォーマンス問題を解決することができます。また、これは Btrieve ベースのアプリケーションのほか、COBOL で作成されたアプリケーション、ISAM データベース (PSQL は本来、非常に速い ISAM) に依存するアプリケーションにも適用できます。その他 PSQL がサポートするデータベース アクセス方法である ADO.NET、ODBC、JDBC、JCL、PDAC、OLE DB および ActiveX のいずれの方法を使用する ISV にとっても、アップグレードは素晴らしい選択です。

結論

コンピューティングの歴史では初めて、新しいハードウェアが事実上アプリケーションの速度を低下させています。速度低下が少ないアプリケーションもありますが、共有リソースを処理し、その状態が非常に重要となる (トランザクションなど) マルチユーザー アプリケーションは大いに速度が低下する可能性があります。

アプリケーションの並列化は、難解で時間もかかり、リスクが伴います。プログラミング、デバッグ、そしてテストのすべてが難しい作業です。ご自分の作業チームが、そのアップグレード サイクルの期限を必ず守っている場合でも、マルチコアの出現における多くの悩みを解決すればすべてが一変される可能性があります。単なるスレッド間の競合ではなく、マルチコア上のスレッド間の競合を管理する必要がある場合、各スレッドが独自のスレッド グループを持ち、問題が拡大します。マルチスレッド プログラムになんらかの弱点または効率性の問題があれば、それらの問題はマルチコア システムで拡大します。オペレーティング システムのリソース管理では不十分で、大多数のアプリケーションがマルチコアの効果を得ることができません。

しかし、すでに述べてきたように、データベースなどアプリケーション スタックの一部を変更することは可能です。これにより、マルチコアのジレンマを解決する簡単な方法が得られるので、将来的には強力なマルチコア システム (現在は事実上ハードウェアのアップグレード) を活用するアプリケーションの再設計に取り組んでください。Pervasive PSQL を使用している、あるいは PSQL に簡単に移行できるデータ ストアを使用しているのであれば、次期バージョン (v11) の PSQL へ移行することによって、アプリケーションのパフォーマンスを向上させ、マルチコアによる速度低下を回避することができます。