Parallel Universe ～並列処理の世界～へ足を踏み入れる準備はできていますか：マルチコア向けの**アプリケーションの最適化**

「4 頭の馬に荷車を引かせることはできるが、1024 羽のにわとりに荷車を引かせる方法はわからない。」
—Enrico Clementi

Levent Akyil
Microsoft* Visual Studio* C/C++ を使用してWindows* アプリケーションを作成する開発者向けに設計された新しいインテル® Parallel Studio で実現可能な並列化手法について検証します。

はじめに

マルチコア・プロセッサーにより、消費者と開発者の両方にとって新しい時代が到来しました。消費者にさまざまな可能性が提供される一方で、開発者は、新しいマルチコア・プロセッサーの優れた処理能力を効率的に使用する製品を開発する必要性に迫られています。インテルは、開発に費やした莫大な投資を無駄にすることなく、急速に普及が進むマルチコア・システムを生産的に活用するツールをソフトウェア・コミュニティーに提供することを使命としています。この取り組みの一環として、インテルは Microsoft Visual Studio C/C++ を使用する開発者向けにインテル® Parallel Studio を発表し、幅広い分野のソフトウェア開発者に Parallel Universe への扉を開きました。インテル® Parallel Studio は、競争の激しいソフトウェア市場で高い地位が得られるアプリケーションの開発を支援するだけでなく、マルチコア向けの並列化を実現し、メニーコア・プロセッサー向けに優れたスケーリングを提供します。

図 1: 並列ソフトウェア開発の 4 つのステップ

ここでは、インテル® Parallel Studio とソフトウェア開発の異なる側面に対応した主要なツール (図 1) について紹介します。インテル® Parallel Studio は、インテル® Parallel Advisor (設計)、インテル® Parallel Composer (コーディング/デバッグ)、インテル® Parallel Inspector (検証)、およびインテル® Parallel Amplifier (チューニング) で構成されています。

インテル® Parallel Composer では、コンパイラーとスレッド化ライブラリーによって、ソフトウェア開発における並列化の導入を迅速に

行うことができます。広範囲の並列プログラミング・モデルがサポートされているため、開発アプリケーションに最も適したコーディング方法に合致するモデルを見つけることができます。インテル® Parallel Inspector は、事前に「バグを発見」します。並列化プログラミング・モデルを問わず、アプリケーションの信頼性を向上する柔軟なツールです。従来のデバッガーとは異なり、このインテル® Parallel Inspector は、マルチスレッド化された C/C++ Windows アプリケーションで発見が困難なスレッド化エラーを検出し、データ競合やデッドロックのような不具合の元となる原因を分析します。インテル® Parallel Amplifier は、スケーリングを制限してしまう予期しない直列化の発見に役立ち、マルチコア・プロセッサー向けに最適なパフォーマンスを発揮する並列アプリケーションのためのきめ細かな調整を行います。

インテル® Parallel Composer

インテル® Parallel Composer を使用することで、開発者は並列処理を簡単に表現し、マルチコア・アーキテクチャーを活用することができます。インテル® Parallel Composer は、並列処理の迅速な導入を支援する並列プログラミング拡張を提供します。Microsoft Visual Studio 環境に統合され、OpenMP* 3.0、ラムダ関数、自動ベクトル化、自動並列化、スレッド化ライブラリーのサポートなど、アプリケーション・レベルで並列化を行うための機能が追加されます。受賞歴もあるインテル® スレッディング・ビルディング・ブロック (インテル® TBB) も、インテル® Parallel Composer の重要なコンポーネントの 1 つです。C/C++ において、使いやすく、ハイパフォーマンスで移植性の高い並列プログラミングを行う方法を提供します。

例

```
void sum (int length, int *a, int *b, int *c )
{
 int i;
 for (i=0; i<length; i++)
                c[i] = a[i] + b[i];
}

//並列関数を使用した場合
sum(1000, a, b, c);
```

```
// シリアル関数を使用した場合
__taskcomplete
{
 __task sum(500, a, b, c);
 __task sum(500, a+500, b+500, c+500);
}
```

インテル® Parallel Composer により提供される並列化の主な拡張機能は次のとおりです。

■ **ベクトル化サポート:** インテル® コンパイラーのコンポーネントである自動ベクトライザーは、MMX® 命令、インテル® ストリーミング SIMD 拡張命令 (インテル® SSE、SSE2、SSE3 および SSE4 ベクトル化コンパイラー命令およびメディア・アクセラレーター命令) とストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) を自動的に使用します。

■ **OpenMP 3.0 サポート:** インテル® コンパイラーは、開発者がソースプログラムで指定した OpenMP 宣言子に従ってコード変換を実行し、マルチスレッド・コードを生成します。インテル® コンパイラーは、現在の業界標準である OpenMP 宣言子のすべてに対応し、OpenMP 宣言子の注釈のある並列実行プログラムをコンパイルします。また、OpenMP バージョン 3.0 仕様に、ランタイム・ライブラリー・ルーチンおよび環境変数を含むインテル独自の拡張機能を提供します。/Qopenmp スイッチを使用すると、コンパイラーは OpenMP 宣言子に基づいてマルチスレッド・コードを生成します。生成されるコードは、単一プロセッサー・システムおよびマルチプロセッサー・システムの両方で並列実行が可能です。

■ **自動並列化機能:** インテル® コンパイラーの自動並列化機能は、入力プログラムの直列部分を同等のマルチスレッド・コードに自動的に変換します。自動並列化機能は、ワークシェアリング候補のループを特定し、正しい並列実行を確認するためにデータフロー解析を行います。また、OpenMP 宣言子のプログラミングに必要な場合には、スレッド化コード生成のデータをパーティショニングします。/Qparallel を使用すると、コンパイラーはアプリケーションの自動並列化を試みます。

■ **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):** 受賞歴もあるランタイムベースのプログラミング・モデルです。マルチコア・プロセッサーの潜在的なパフォーマンスを活用できるように支援する、テンプレート・ベースのランタイム・ライブラリーで構成されています。開発者は並行収集と並列アルゴリズムを活用して、スケーラブルなアプリケーションを作成することができます。

■ **単純並列関数:** 4 つのキーワード (__taskcomplete、__task、__par、および __critical) を文のプリフィックスとして使用して、簡単に並列プログラムを作成することができます。キーワードは、OpenMP ランタイムサポートを使用して実装されます。プログラムの並列化を細かく制御する場合は、OpenMP 機能を直接使用してください。この機能を使用する場合は、/Qopenmp コンパイラー・スイッチを指定して並列実行機能を有効にします。コンパイラー・ドライバーは、自動的に OpenMP ランタイム・サポート・ライブラリーにリンクします。実際の並列処理は、ランタイムシステムによって管理されます。

■ **C++ ラムダ関数サポート:** C++ ラムダ式は、関数オブジェクトを定義する基本式です。このような式は、関数オブジェクトが想定される場所ではどこでも使用できます (例: Standard Template Library (STL) アルゴリズムの引数)。インテル® C++ コンパイラーは、ISO C++ ドキュメント (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/) (英語) で定められているラムダ式を実装しています。/Qstd=c++0x スイッチを使用すると、ラムダサポートが有効になります。

■ **インテル® インテグレーテッド・パフォーマンス・プリミティブ (インテル® IPP):** インテル® Parallel Composer に含まれており、マルチメディア、データ処理、通信アプリケーション向けに高度に最適化された、ソフトウェア関数の広範囲なマルチコア対応ライブラリーです。インテル® コンパイラーおよびパフォーマンス最適化ツールとともに使用できる、最適化されたソフトウェア・ビルディング・ブロックを提供します。インテル® IPP には、信号処理、オーディオ・コーディング、音声認識、音声コーディング、画像処理、ビデオ・コーディング、小行列の演算、3D データ処理などの分野でアプリケーションを作成するための基本的な低レベルの関数が用意されています。

■ **valarray:** ハイパフォーマンス・コンピューティング向けの配列演算を含む配列の C++ STL (標準テンプレート・ライブラリー) コンテナークラスです。これらの演算は、ベクトル化などのハードウェア機能を活用します。インテル® コンパイラーは valarray を組み込み型として認識し、インテル® IPP ライブラリーの呼び出しに置換します。

例

```
// 整数型の valarray を作成する
valarray_t::value_type ibuf[10] = {0,1,2,3,4,5,6,7,8,9};
valarray_t vi(ibuf, 10);

// ブール型の valarray のマスクを作成する
maskarray_t::value_type mbuf[10] = {1,0,1,1,1,0,0,1,1,0};
maskarray_t mask(mbuf,10);

// マスクされた配列の値を倍にする
vi[mask] += static_cast<valarray_t> (vi[mask]);
```

■ **インテル® Parallel Debugger Extension:** Microsoft Visual Studio に統合される、インテル® C++ コンパイラーの並列コード開発機能向けのアドオンデバッガーです。Microsoft Visual Studio のデバッグ機能を置換するのではなく、既存の機能を拡張します。

- **異なるスレッドからの同一データ要素へのアクセスを検出するスレッドデータ共有解析**
- **再入可能な関数呼び出しでプログラムの実行を停止するスマート・ブレークポイント機能**
- **OpenMP の並列ループで動的な追加作業スレッドの作成を有効/無効にするシリアル実行モード**
- **高度な OpenMP プログラム状態解析用の OpenMP ランタイム情報ビュー**
- **SIMD (Single Instruction Multiple Data) 命令セットを使用して並列データをデバッグするように、形式オプションと編集オプションが拡張されたストリーミング SIMD 拡張命令 (SSE) レジスタービュー**

このように、インテル® Parallel Debugger Extension は、スレッドデータ共有問題を特定するのに便利です。インテル® Parallel Debugger Extension は、ソース・インストルメンテーションを使用して、データ共有問題を特定します。この機能を有効にするには、[構成プロパティ] > [C/C++] > [General (全般)] > [Debug (デバッグ)] > [Enable Parallel Debug Checks (並列デバッグ検証を有効にする)] を選択して、/debug:parallel を設定します。図 2 は、インテル® Parallel Debugger Extension が、同じデータにアクセスする 2 つのスレッドを検出して、アプリケーションの実行を停止したことを示します。

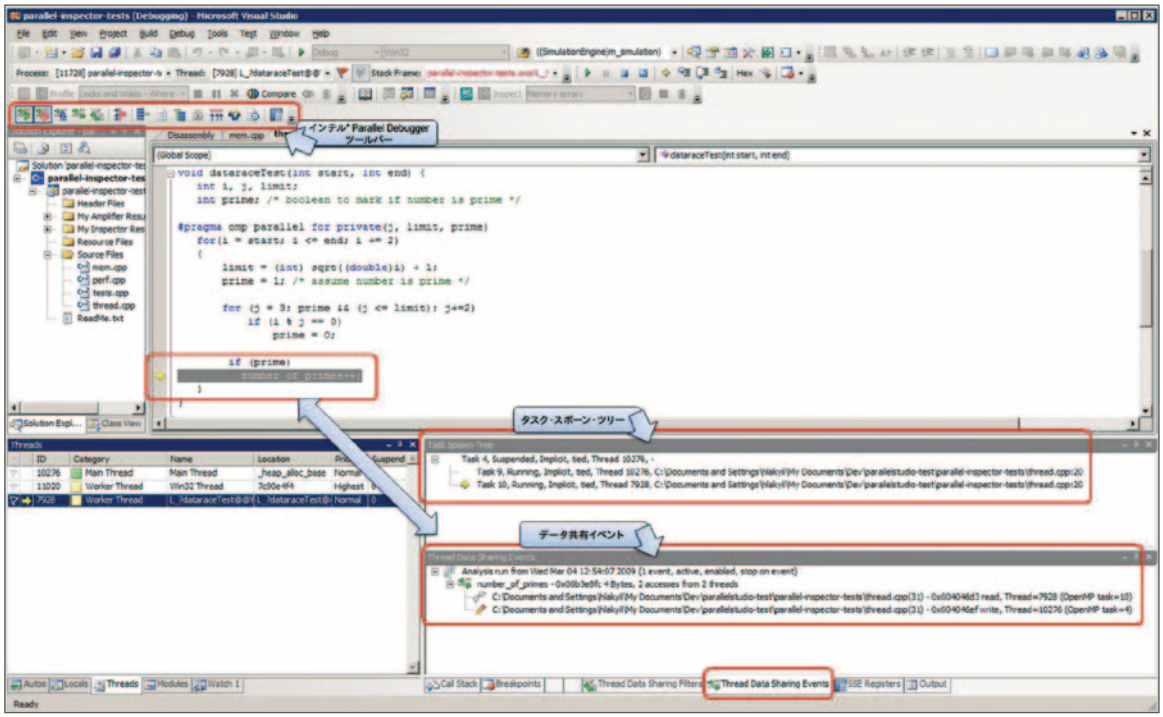

図 2: インテル® Parallel Debugger Extension はデータ共有問題を検出すると実行を中断することが可能

インテル® PARALLEL INSPECTOR

「この問題は、発生するまでその形跡すらありませんでした。非常に奥深くに埋もれていて、数百万行ものコードとデータを何週間もかかって詳細に調べ上げる必要がありました。」

—Ralph DiNicola
2003 年北米北東部大停電のアメリカ-カナダ合同調査委員会スポークスマン

マルチスレッド・アプリケーションでエラーの原因を特定することは困難です。インテル® Parallel Studio ツールの 1 つであるインテル® Parallel Inspector は、マルチスレッド・アプリケーションにおけるスレッド化エラーとメモリーエラーの元となる原因を分析し、事前にバグを発見します。

インテル® Parallel Inspector は、C および C++ アプリケーション開発者向けに、次の機能を提供しています。

- リーク、バッファー・オーバーラン・エラー、ポインターの問題など、メモリーとリソースに関するさまざまな問題を特定
- スレッド関連のデッドロック、データ競合、およびその他の同期化問題を予測し特定
- 並列アプリケーションの潜在的なセキュリティー問題の特定
- サイズ、頻度、種類順でエラーをすばやくソートし、重大な問題を認識して優先順位を設定

インテル® Parallel Inspector (図 3) は、Pin と呼ばれるバイナリー・インストルメンテーション・テクノロジーを使用して、メモリーエラーとスレッド化エラーを検証します。Pin は、インテルより提供されている動的なインストルメンテーション・システムです (http://www.pintool.org)。実行中の実行ファイルの任意の場所に C/C++ コードを挿入して、プログラムの動作を監視できます。

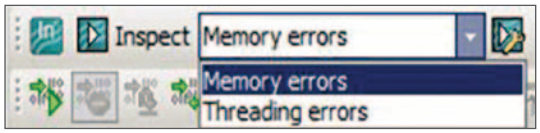

図 3: インテル® Parallel Inspector のツールバー

インテル® Parallel Inspector のメモリー分析レベル

インテル® Parallel Inspector は、Pin の設定を変更して、図 4 に示すように、設定とオーバーヘッドが異なる 4 つのレベルで分析を行うことができます。最初の 3 つの分析レベルはヒープ上で発生するメモリーエラーを分析し、4 つめのレベルはスタック上で発生するメモリーエラーも分析します。インテル® Parallel Inspector では、これらの分析レベルのサポートに、リーク検出テクノロジー (レベル 1) とメモリー・チェック・テクノロジー (レベル 2 ～ 4)を使用しています。また、これらのテクノロジーでは、さまざまな方法で Pin を使用しています。

レベル 1 レベル 1 分析は、アプリケーションのメモリーリークを見つけるのに役立ちます。メモリーリークは、メモリーブロックが割り当てられ、解放されない場合に発生します。

レベル 2 レベル 2 分析は、初期化されていないメモリーへのアクセス、不正な割り当て解除、割り当てと解除の不一致など、アプリケーションの不正なメモリーアクセスを検出します。不正なメモリーアクセスは、読み取り操作や書き込み操作が、論理的または物理的に不正なメモリーを参照した場合に発生します。このレベルでは、不正な部分メモリーアクセスも認識されます。不正な部分アクセスは、読み取り操作が、論理的に不正な部分を含むメモリーブロック (2 バイト以上) を参照した場合に発生します。

レベル 3 レベル 3 分析は、コールスタックの深さが 1 から 12 に変更され、ぶら下がりポインターの拡張チェックが有効であることを除き、レベル 2 に似ています。ぶら下がりポインターとは、すでに存在しないデータにアクセスしたり、そのようなデータを指定しているポインターです。インテル® Parallel Inspector は、割り当て解除を遅延して、メモリーが再割り当てされないように (別の割り当て要求によって返されないように) します。そのため、割り当て解除以降の参照は、ぶら下がりポインターによる不正な参照であることがわかります。この手法には追加のメモリーが必要で、遅延された割り当て解除リストに使用されるメモリーは制限されているため、インテル® Parallel Inspector は遅延した参照の割り当てをいずれ解除する必要があります。

レベル 4 レベル 4 分析は、すべてのメモリーエラーを見つけようとします。コールスタックの深さは 32 に変更され、ぶら下がりポインターの拡張チェック、システム・ライブラリーの分析、およびスタック上で発生するメモリーエラーの分析も行います。スタック分析は、このレベルでのみ有効になります。

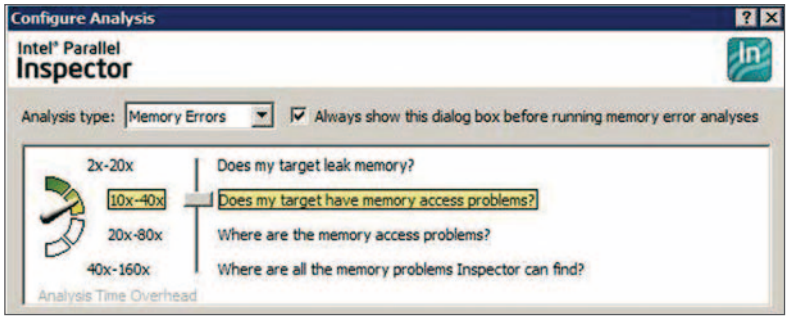

図 4: メモリーエラーの分析レベル

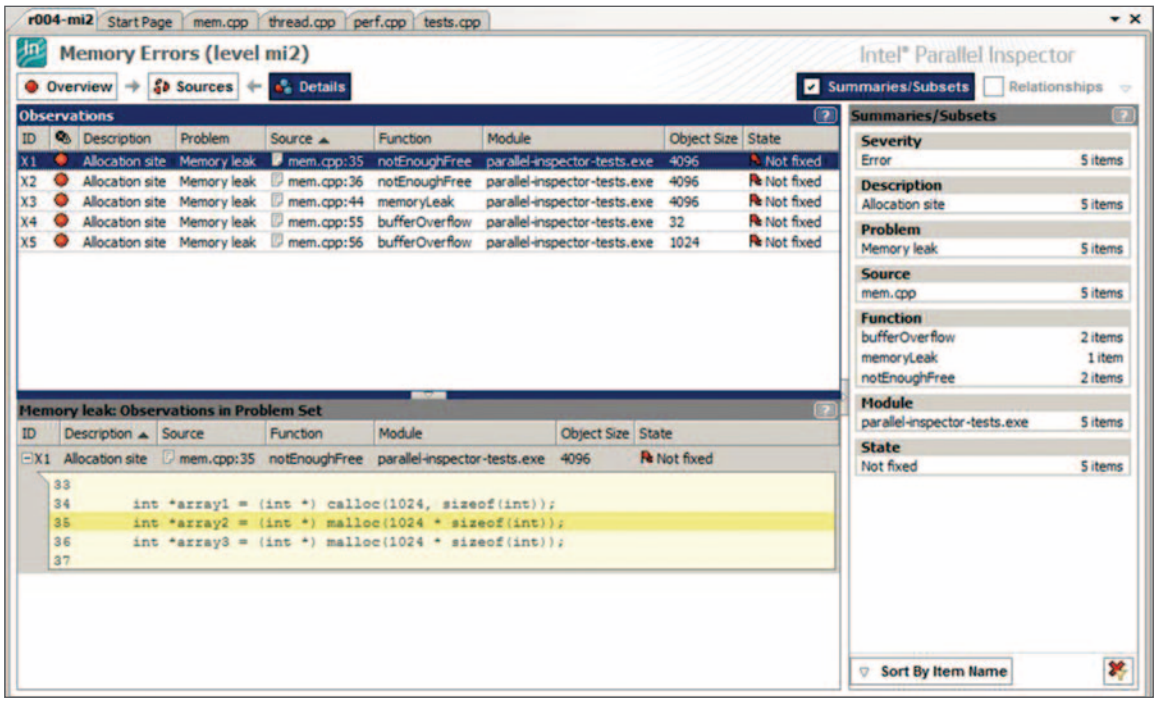

図 5: 検出されたメモリーエラーを示すインテル® Parallel Inspector の分析結果

図 5 に示すように、重大度、説明、原因、関数名、モジュールで結果をフィルターできます。

インテル® Parallel Inspector のスレッド化エラー分析レベル

インテル® Parallel Inspector には、スレッド化エラーの分析レベルも 4 つあります (図 6)。インテル® Parallel Inspector で認識される主なスレッド化エラーには、データ競合、デッドロック、ロック階層違反、および潜在的なプライバシーの侵害があります。

レベル 1 レベル 1 分析は、アプリケーションのデッドロックを見つけるのに役立ちます。デッドロックは、2 つ以上のスレッドが、他のスレッドがリソース (MUTEX、クリティカル・セクション、スレッドハンドルなど) を解放するのを待機しているのに、どのスレッドもリソースを解放しない場合に発生します。この場合、どのスレッドも先に進むことができません。コールスタックの深さは 1 に設定されます。

レベル 2 レベル 2 分析は、アプリケーションのデータ競合とデッドロックを検出します。データ競合は、最も一般的なスレッド化エラーの 1 つで、適切な同期処理が行われずに複数のスレッドが同じメモリー位置にアクセスした場合に発生します。このレベルでも、コールスタックの深さは 1 に設定されます。このレベルのバイト・レベルの粒度は 4 です。

レベル 3 レベル 3 分析は、レベル 2 のデータ競合とデッドロックを検出に加えて、それらの発生場所も検出します。より詳細な分析を行うために、コールスタックの深さは 12 に設定されます。このレベルのバイト・レベルの粒度は 1 です。

レベル 4 レベル 4 分析は、すべてのスレッド化エラーを見つけようとします。コールスタックの深さは 32 に変更され、スタック上で発生するエラーの分析も行います。スタック分析は、このレベルでのみ有効になります。このレベルのバイト・レベルの粒度は 1 です。

データ競合はさまざまな場合に発生します。インテル® Parallel Inspector は、write-write (書き込み-書き込み)、read-write (読み取り-書き込み)、および write-read (書き込み-読み取り) データ競合を検出します。

- write-write データ競合は、2 つ以上のスレッドが同じメモリー位置に書き込みを行う場合に発生します。
- read-write データ競合は、1 つのスレッドがメモリー位置から読み取りを行う際に、別のスレッドが同時に同じメモリー位置に書き込みを行う場合に発生します。
- write-read データ競合は、1 つのスレッドがメモリー位置に書き込みを行う際に、別のスレッドが同時に同じメモリー位置から読み取りを行う場合に発生します。

すべての場合において、実行順序が共有データに影響を与えます。

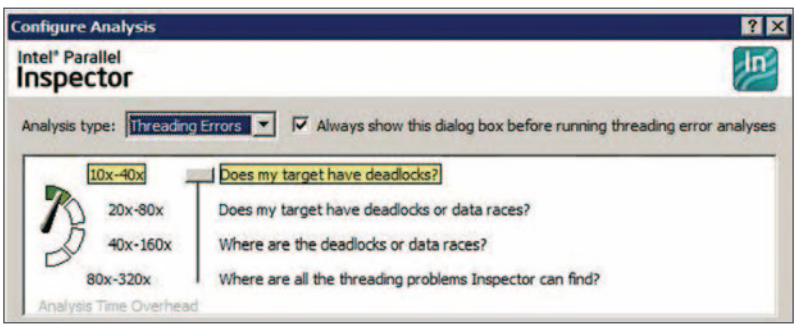

図 6: スレッド化エラーの分析レベル

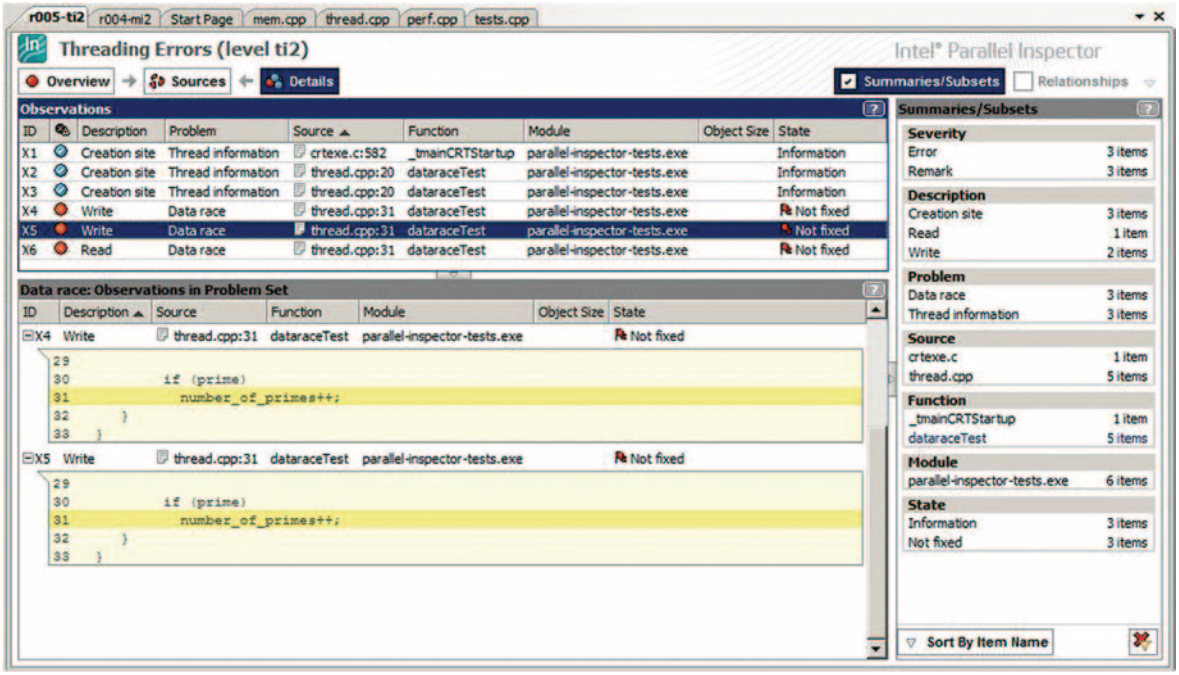

図 7: 検出されたデータ競合問題を示すインテル® Parallel Inspector の分析結果

インテル® PARALLEL AMPLIFIER

「プログラマーは、プログラムの重要ではない部分の速度について膨大な時間を無駄にしています。約 97 %の場合において、小さな効率は無視すべきです。中途半端な最適化は諸悪の根源と言えます。」

—*Donald Knuth*（出典：C. A. R. Hoare）

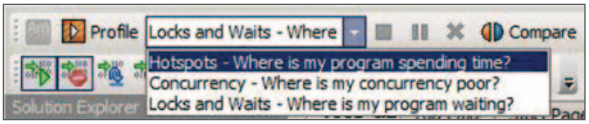

図 8: インテル® Parallel Amplifier のツールバー

マルチスレッド・アプリケーションは、並列化による複雑性のため、固有の問題セットを持つ傾向にあります。シリアルコードをスレッドセーフなコードに変換することは簡単ではありません。通常は、開発期間に影響を与え、既存のシリアル・アプリケーションよりも複雑になります。マルチスレッド・アプリケーション特有のパフォーマンス問題は、次のように要約できます。

- 複雑性（データ再構成、同期化の使用）
- パフォーマンス（最適化とチューニングの必要性）
- 同期オーバーヘッド

Knuth 氏のアドバイスに従って、インテル® Parallel Amplifier（図 8）は、開発者が投資収益率（ROI）の最も高い最適化を実現できるように、コードのボトルネックを特定するのを支援します。アプリケーションのパフォーマンス問題を特定し、適切に排除することは、効率的な最適化を行う上で重要です。

マウスをシングルクリックするだけで、インテル® Parallel Amplifier は、hotspot 分析、並列性分析、およびロックと待機の分析という 3 種類の強力なパフォーマンス解析を行うことができます。各分析レベルについて説明する前に、インテル® Parallel Amplifier で使用する指標について理解しておいたほうが良いでしょう。

経過時間　アプリケーションの実行時間です。一定の作業の実行において、アプリケーションの経過時間を減らすことは、重要な目的の 1 つです。アプリケーションの経過時間は、サマリービュー（図 11）に表示されます。

CPU 時間　プロセッサーでスレッドが実行に消費した時間です。複数のスレッドの場合は、すべてのスレッドの CPU 時間が合計されます。合計 CPU 時間とは、分析中に実行されたすべてのスレッドの CPU 時間の合計です。

待機時間　指定されたスレッドが、同期待機や I/O 待機などのイベントが発生するまで待機に要した時間です。

hotspot 分析

オーバーヘッドの低い統計サンプリング (スタック・サンプリングとも呼ばれる) アルゴリズムを使用することで、hotspot 分析は、開発者がアプリケーション・フローを理解し、実行に時間のかかるコードセクション (hotspot) を特定できるようにします。hotspot 分析中、インテル® Parallel Amplifier は、OS タイマーを使用して一定の間隔でサンプリングを行い、アプリケーションをプロファイルします。すべてのアクティブな命令アドレスとその呼び出しシーケンスのサンプルが収集されます。そして、格納されたサンプリング結果の命令ポインター (IP) と関連する呼び出しシーケンスを分析し、表示します。統計的に収集された IP サンプルと呼び出しシーケンスを使用して、インテル® Parallel Amplifier はコールツリーを生成し、表示します。

並列性分析

並列性分析は、アプリケーションがシステムで利用可能なプロセッサーをどれだけ活用しているかを測定します。図 10 に示すように、プロセッサーが有効に利用されていない hotspot 関数を特定するのに役立ちます。並列性分析中、インテル® Parallel Amplifier は、アクティブなスレッド数を収集して、提供します。アクティブなスレッドとは、実行中のスレッド、または定義されている待機 API やブロッキング API により待機中ではないキューイングされているスレッドです。実行中のスレッド数は、アプリケーションの並列レベルに対応します。並列レベルとプロセッサー数を比較することで、インテル® Parallel Amplifier は、アプリケーションによるシステムで利用可能なプロセッサーの利用状況を分類します。

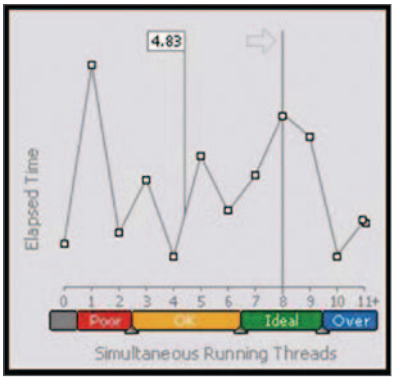

図 11: インテル® Parallel Amplifier の並列分析結果 - サマリービュー

[Concurrency (並列性)] ウィンドウと [Locks and Waits (ロックと待機)] ウィンドウの時間は、次の利用状況を示します (図 11)。

Idle: プログラム中のすべてのスレッドが待機中です。実行中のスレッドはありません。[Summary (サマリー)] タブのグラフには、アイドリング中であることを示す 1 つのノードだけが表示されます。

Poor: 十分に利用されていません。デフォルトでは、スレッド数が並列化ターゲットの 50% 以下の場合です。

OK: 許容可能範囲で利用されています。デフォルトでは、スレッド数が並列化ターゲットの 51 ～ 85% の場合です。

Ideal: 理想的に有効利用されています。デフォルトでは、スレッド数が並列化ターゲットの 86 ～ 115% の場合です。

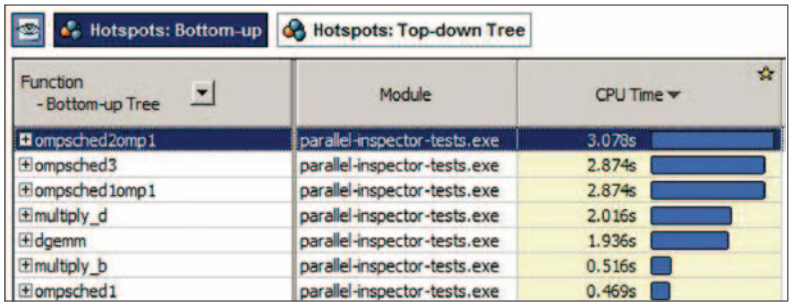

図 9: インテル® Parallel Amplifier の hotspot 分析結果

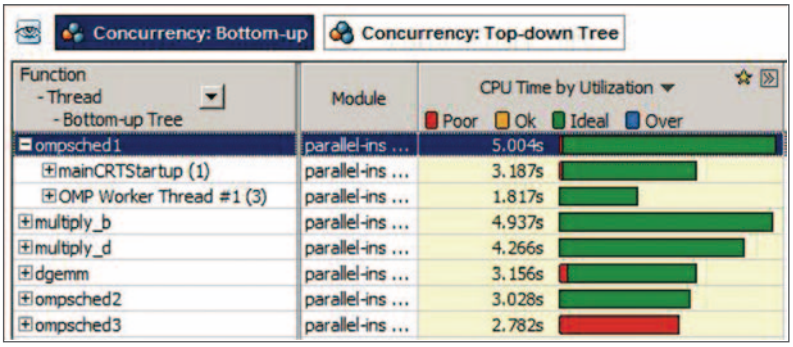

図 10: インテル® Parallel Amplifier の並列性分析結果 (粒度が Function-Thread の場合)

ロックと待機の分析

並列性分析は、アプリケーションの並列化されていない場所や利用可能なプロセッサーを十分に活用していない場所を特定し、ロックと待機の分析は、プロセッサーが有効に利用されていない原因を特定するのに役立ちます。プロセッサーが有効に利用されていない最も一般的な原因は、スレッドが同期オブジェクト (ロック) を長時間待機しているためです。多くの場合、この間に有効な作業が行われないため、パフォーマンスが低下し、プロセッサーが有効に使用されません。

ロックと待機の分析中、開発者は、各同期オブジェクトの影響を予測することができます。分析結果は、各同期オブジェクトまたはスリープやブロッキング I/O などのブロッキング API における、アプリケーションの待機時間を理解するのに役立ちます。

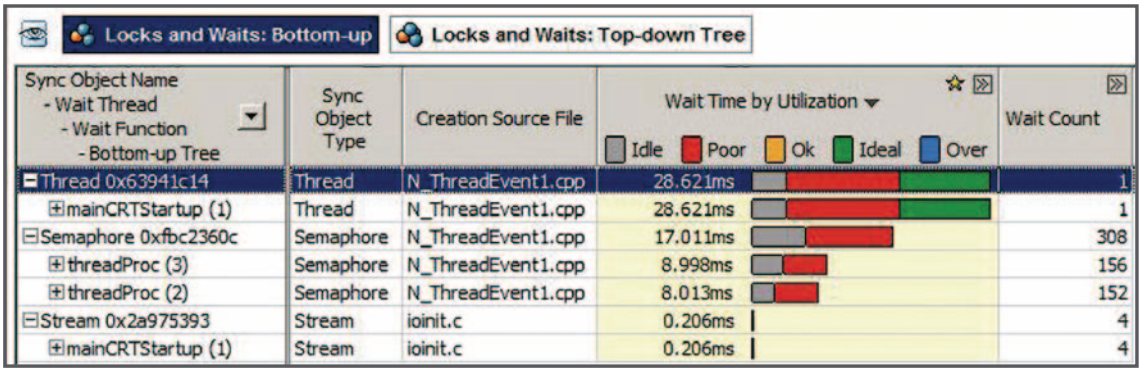

図 12: ロックと待機の分析結果

分析結果を基に、同期オブジェクトを mutex (相互排他オブジェクト)、セマフォー、クリティカル・セクションおよび fork-join 処理にすることができます。ほとんどの場合、待機時間が最も長く、並列レベルの高い同期オブジェクトがアプリケーションのボトルネックになっています。

インテル® Parallel Inspector とインテル® Parallel Amplifier は、両方ともソース・コード・レベルまでドリルダウンすることができます。これは非常に重要なことです。例えば、図 13 の行をダブルクリックして、ソースコードにドリルダウンし、問題を引き起こしている同期オブジェクトを確認できます。

まとめ

並列プログラミングは新しいものではありません。長年の間、ハイパフォーマンス・コンピューティング (HPC) の分野でよく研究され、使用されてきました。今日、マルチコア・プロセッサーの普及に伴い、並列プログラミングは主流となりつつあります。今こそ、まさにインテル® Parallel Studio の出番と言えるでしょう。インテル® Parallel Studioはソフトウェア開発者が並列化プログラミングへ挑戦する機会をもたらすツールであり、並列化というパラダイムシフトをサポートすることにより、ソフトウェア開発者が並列処理の世界に足を踏み入れる際の障壁を劇的に減らします。PARALLEL UNIVERSE へようこそ。

著者紹介 : **Levent Akyil** は、Intel Corporation のソフトウェア開発製品部パフォーマンス、分析、スレッド化研究室のソフトウェア・エンジニアです。

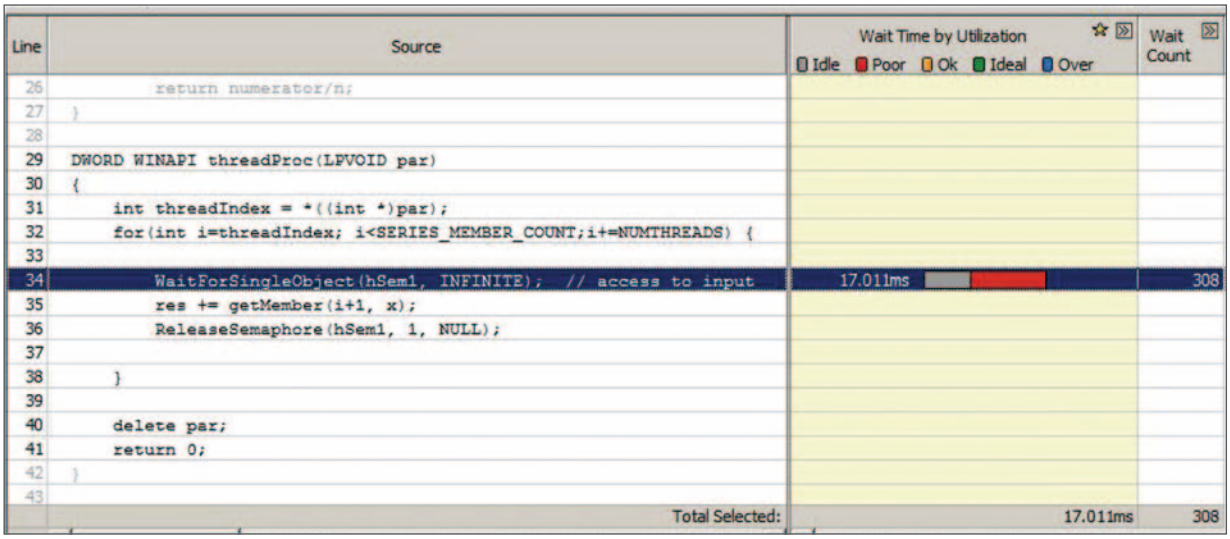

図 13: ロックと待機の分析のソース・コード・ビュー