THE PARALLEL UNIVERSE

Issue 8
2011 年 9 月

HPC 事例：

生物物理学者と数学者が支持する インテル® Parallel Advisor を使用した 並列化
Zakhar A. Matveev

インテル® Parallel Studio XE SP1
Michael D'Mello

インテル® スレッディング・ビルディング・ブロックのフローグラフ
Michael J. Voss, Ph.D.

目次

並列プログラミング：

新しい書籍を執筆することになったきっかけ

編集者からのメッセージ

James Reinders は、インテル コーポレーションのチーフ・エバンジェリストです。『Intel Threading Building Blocks: Outfitting C++ for Multicore Processor Parallelism』（日本語、中国語、韓国語の翻訳版があります）などの並列化に関する文献を発表しています。また、並列化に関する幅広いインタビューも受けています。

今回は、3 つの全く異なる記事を紹介しましょう。どの記事も役立つものばかりです。

インテル® Parallel Studio XE SP1 の記事では、機能の概要を説明します。まだこれらのツールを使用したことのない方には、ツールに興味を持つ良いきっかけとなるでしょう。ツールをすでに使用している方には、機能をさらに詳しく知りたくなるきっかけとなるかもしれません。

特に今回はお気に入りの記事を選択するのが難しいのですが、あえて 1 つ挙げるとすれば、ユーザー事例を紹介している「HPC 事例：生物物理学者と数学者が支持するインテル® Parallel Advisor を使用した並列化」でしょうか。私はユーザー事例を紹介している記事に弱いのです。この記事は、ユーザー事例であるというだけでなく、インテル® Parallel Advisor を使用したときに遭遇するさまざまな点をカバーしています。

私の知人は、私のことを「インテル® Parallel Advisor を信用しない人間」だと評しています。これは間違っていません。この数十年の間、私は並列化が簡単に行えるという多くのツールを目にしてきました。これらのツールは私を骨の髄まで懐疑論者にするものでした。ある意味、私は正しかったといえます。私は今でもすべてのコードを正確に理解して並列化するツールは存在しないと確信しています。現在でも、並列化を行うにはプログラマーの知識と経験が必要なのです。

ただし、私はインテル® Parallel Advisor を信用しないことはやめました。考えを変えたのは、インテル® Parallel Advisor が使えるツールだからです。

インテル® Parallel Advisor はこれまでの常識を覆すものです。SF 好きな人は、カークがどのように「コバヤシマル・シナリオ[1]」に勝利したのかといえばお分かりになるでしょう。インテルは、あらゆることができる魔法のツールを作成する代わりに、従来のツールではできなかった、可能性の解析を支援するツールを作成することにしました。ところが、このツールは、時間を費していた作業を短縮できる、まさに魔法のツールであることが分かったのです。

インテルは、あらゆることができる魔法のツールを作成する代わりに、従来のツールではできなかった、可能性の解析を支援するツールを作成することにしました。

私も時間を浪費していた一人です。私はコードの並列化についてアイデアを持っていましたが、それはあくまでもコードを記述してデバッグする方法であり、スケーリングに苦労する重要なポイントを見逃していたことに気付いたのです。もう少し早くこのことに気付いていれば良かったと思っています。「Good enough (これで十分)」なのか判断したり、時間を浪費することはもうなくなりました。インテル® Parallel Advisor は、長時間のプログラミングやデバッグを行うことなくこれらの情報を教えてくれるため、「Good enough」という判断がそもそも必要なくなったからです。HPC の記事では、あるユーザーと課題に取り組む事例を紹介しています。この記事をお読みになることで、同じような問題でお困りの方々が袋小路に入らないで済むことを願います。インテル® Parallel Advisor を使用して問題を詳しく調べることで、無駄な作業に時間を費やすことなく、アプリケーションをスケーリングする方法を見つけ出せるでしょう。幸いなことに、このツールがどの程度エキスパートや初心者の支援となるか、開発者チームと賭けをしていませんでした。もし賭けをしていたら、私の財布は空になっていたでしょう。インテル® Parallel Advisor が実際にユーザーの役に立っている様子を見るのはとても嬉しいことです。

「インテル® スレッディング・ビルディング・ブロックのフローグラフ」は、私のお気に入りの機能について述べています。私がインテル® スレッディング・ビルディング・ブロック (インテル® TBB) を気に入っていることは周知の事実ですが、この最新の機能を見れば、私がインテル® TBB とそれを取り巻くチームをなぜ気に入っているのか、お分かりいただけるでしょう。ユーザーはインテル® TBB にさまざまなインターフェイスを望んでいました。特に、ゲーム開発者は、タスクをスケジュールするイベントベースのインターフェイスを求めていました。別の開発者は、アプリケーションの多くの依存部分を調整するインターフェイスを切望していました。ほかの開発者からは特に要望はありませんでしたが、ある開発者が、複雑にもかかわらず、インテル® TBB のパイプライン機能で利用しようとしているインターフェイスは、サポートチームのメンバーの参考になるものでした。インテル® TBB の設計チームは、これらの 3 つのインターフェイスのニーズを満たすソリューションとして、汎用的な「グラフ」インターフェイスを提供することにしました。社内では、ベータテストの初期段階も含め、長い間この機能は tbb::graph と呼ばれていました。製品をリリースする前に、この名称は「フローグラフ」に変更されました。この名称は機能の特長をよく表しており、名称の変更について好意的なフィードバックが寄せられました。

依存性グラフで構成されているタスクのグループの概念は、多くのアプリケーションで非常に一般的なものです。私が最初にこの機能を使用したのはコンパイラーの設計でしたが、その後レーダーシステム設計のような信号処理アプリケーションでも何度も使用しました。インテル® TBB の新しいフローグラフは、このようなプログラムに最適なソリューションです。

ああ、すみません。新しい書籍の話をしていませんでしたね。現在、私は数人のエキスパートと並列プログラミングに関する書籍を執筆しています。ようやく完成の目処が見えてきたところです。原稿のかなりの部分を書き終え、レビュアーや出版社との間で最終的な詰めに入っています。この書籍は並列プログラミングの指南書となり得るでしょうか。そうなることを願っています。時間が明らかにしてくれるでしょう。レビュアーから受けた指摘と書籍の中身については、次回にお知らせする予定です。現時点では、並列化をどのように説明すれば理解してもらえるか、ユーザーとの共同作業で学んだことを書こうとしている、とだけお伝えしておきます。コミュニケーションが重要であり計算がすべてではないこと、一般的なパターンを知りさまざまな例を見ることで多くのことを知ることができること、などを学んだと思います。

インテルのツールはユーザーを支援します。この号が皆様のニーズに適したツールを確認する上で、お役に立つことを期待しています。

いつも多くのフィードバックをお寄せいただきありがとうございます。皆様からのフィードバックとご要望は将来の記事を選ぶ際に参考にさせていただきます。この号をお読みになった感想についても是非お聞かせください。

皆様からのご意見をお待ちしております。

James Reinders
2011 年 9 月

追記: この号の出版直前に、gcc でインテル® Cilk™ Plus がサポートされるというニュースが gcc コミュニティーから入りました。この導入作業を支援するため、インテルでは実装などの情報をコミュニティーに提供してきました。このニュースをお知らせできることをうれしく思います。もちろん、インテルのコンパイラーとツールはインテル® Cilk™ Plus を引き続きサポートします。gcc によるサポートは、インテル® Cilk™ Plus のサポートを向上させ、コーディングもしやすくなるでしょう。詳細はブログをお読みください。この件については、将来の号で詳しく紹介する予定です。幅広いサポートによりインテル® TBB が成功したように、インテル® Cilk™ Plus のサポートがこれからも増えることを期待しています。

[1] 劇場版スタートレック*で候補生が挑戦する、勝利なきシナリオ。

HPC 事例 : 生物物理学者と数学者が支持する インテル® Parallel Advisor を使用した並列化

ソフトウェア開発エンジニア
Zakhar A. Matveev

ロシアの研究者グループが増大する生物学実験のデータと複雑化するシミュレーション要件に対応するためにアプリケーションを並列化した事例を紹介します。

生物物理学の調査にハイパフォーマンス・コンピューティング (HPC) が用いられる機会はますます増加しています。コンピューター・モデリングを使用したシミュレーションは、生物学の「次の顕微鏡」となっています。生物物理学アプリケーションで取り扱う膨大な量の多種多様なデータを処理するには、ハイパフォーマンス・コンピューティングの技術が不可欠です。生物学的「単位」の複雑な相互作用により、モデルの複雑さが増しており、並列化を使用せずにこの分野の計算処理を効率良く行うことは実質的に不可能となっています。

G. Osipov 教授 (Ph.D.)、V. Petrov 氏 (Ph.D.)、M. Komarov 氏 (Ph.D.) を含む、ロシアの Nizhny Novgorod National Research University の研究者グループは、生物物理学の計算を多用するシミュレーション・アプリケーションの開発を何年も続けてきました。増大する生物学実験のデータと複雑化するシミュレーション要件に対応するため、研究者グループはこの分野で膨大な計算を行うという課題に直面しました。研究者グループは、アルゴリズムのパフォーマンスを向上させるため、インテル® Parallel Advisor 2011 (現在はインテル® Parallel Studio XE 2011 Windows* 版にバンドル) を使用してアプリケーションを並列化することを決めました。

以下の 3 つのシリアル C++ アプリケーションが並列化対象でした。

- CARDIAC: 3 次元 Cardiac 3D シミュレーション
- NEURAL: 脳の神経細胞集団の動的分析
- RATE: 仮想移動ロボットの制御を構成する速度現象論的モデル

この記事では、研究者グループがインテル® ソフトウェア・ツールを使用してどのようにコードを並列化したか、順を追って説明します。

3 ドメイン心臓シミュレーション・シリアル・アプリケーション

電気機械心臓シミュレーション・モデルは、医療データを読み取り、不整脈のメカニズムに関する仮説をテストするために幅広く使用されています。Nizhny Novgorod の研究者グループは、CARDIAC アプリケーションで通常の心筋細胞だけでなく、細胞外の空間と小さく受動的な心筋細胞 (線維芽細胞) を考慮に入れて新しく提唱された 3 ドメインモデルを使用しました。より複雑な実世界のモデリングを行うということは、より複雑な数の計算を行うということです。

CARDIAC の出力は、図 1 の CARDIAC ベンチマーク GUI で示されているように、3D ボリュームと期間 [0, T] における心筋細胞と線維芽細胞の電圧レベル V (x,y,z,t) の時空間分布です。

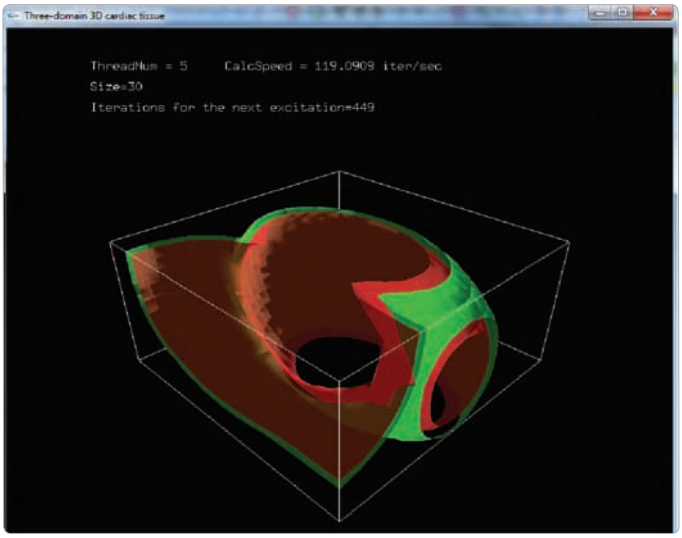


図 1: CARDIAC ベンチマーク・アプリケーション GUI。心室筋細胞の電圧レベルは赤で、線維芽細胞の電圧レベルは緑で示されています。


```
#1    for (t_i) // 時間 ([0, T]) 積分外部ループ
#2        for (P_i = {x,y,z}) // 空間（3D ボリューム）反復内部ループ
                //P_i の合計細胞外電流を計算

          //ポアソンソルバー:
          while (err > err0)
#3            for (P_i = {x,y,z}) // 空間反復内部ループ
                    //P_i の合計細胞外電流を計算

#4        for (P_i = {x,y,z}) // 空間反復内部ループ
                //P_i のイオン電流を計算

#5        for (P_i = {x,y,z}) // 空間反復内部ループ
                //P_i の筋細胞電圧を計算

#6        for (P_i = {x,y,z}) // 空間反復内部ループ
                //P_i の線維芽細胞電圧を計算
```

図 2: CARDIAC アプリケーションのプログラム構造

アルゴリズムには以下の 4 つのステップが含まれます。

1. 合計細胞外電流を調べます (5 地点の中央近似を使用した離散ラプラス作用素の数値計算が必要)。
2. 反復法を使用してポアソン方程式を解きます。
3. 心筋細胞に Luo-Rudy モデル、線維芽細胞に最近提唱されている Sachse モデルを使用して細胞のイオン電流を計算します。数値計算として見た場合、1 つの心筋細胞の動きを記述した 15 の非線形常微分方程式を解くことを意味します。
4. 偏微分方程式を解いて線維芽細胞と心筋細胞の電圧レベルを調べます。

問題の時空間の性質により、図 2 で示されているように、シリアルプログラム構造ではこの処理は入れ子のループになります。

次のセクションでは、並列に計算を実行するため、このプログラムをどのように変更したかを説明します。

CARDIAC 並列化のケーススタディー

概念的には、インテル® Parallel Advisor は、アプリケーションを並列化するための、相互に関連した解析ツールと証明された手法のコンビネーションであるといえます。(手法は研究者グループにとって馴染み深いものになる傾向があります。) 手法は、図 3 のワークフローにあるように、インテル® Parallel Advisor によって明示的に示されます。

研究者グループはワークフローに従ってステップ 1 で調査ツールを使用し、アプリケーションを実行してプロファイリングしました (図 4 を参照)。予想通り、調査ツールのレポートでは、外部時間積分ループが合計実行時間のほぼ 100% を占めていました。しかし、反復間に強い依存性があるため、この種のループを並列実行することは実質的に不可能です (i+1 の反復が反復 i の関数になっています)。このため、空間上の反復を含む内部ループに注目しました。

調査レポートのツリーで、図 4 で強調されているノード、myocyte 積分ループ #5 が内部ループで最良の候補のように見えました。研究者グループは、ほかのインテル® Parallel Advisor ツールで将来解析するために、インテル® Parallel Advisor のマクロ (注釈) を挿入して対応するコード領域 (サイト) をマークしました。

(注: インテル® Parallel Advisor の注釈では、「サイト」は並列化するコード領域、「タスク」はほかのタスク・インスタンスと並列に実行されるプログラム範囲です。たとえば、典型的なループの並列化では、タスク・インスタンスと対応する単一ループ反復はタスク、全体的なループはサイトであると見なされます。)

では、合計時間の 96% を占める 1 つのループを並列化すれば十分なのでしょうか。アムダールの法則によれば、シリアル実行領域が非常に小さな場合でも全体のパフォーマンスに大きく影響します。このため、研究者グループはさらにいくつかの領域を並列化しようと決めました。インテル® Parallel Advisor の注釈を使用すると、実際に並列フレームワーク・コードを実装することなく、単純かつ安全な方法で異なる並列化のアプローチを試せるので、この作業は簡単でした。ほかの並列処理の「hotspot」を特定するため、ソース調査機能を使用してループ時間を計測しました (図 5)。結果を見れば、2 つ目と 3 つ目の候補 (図 2 のポアソンソルバーのループ #3 と <時間ステップ * 電圧> 乗算ループ #6) は明白です。ポアソンソルバーの内部にも、別の小さな空間反復ループがありました。研

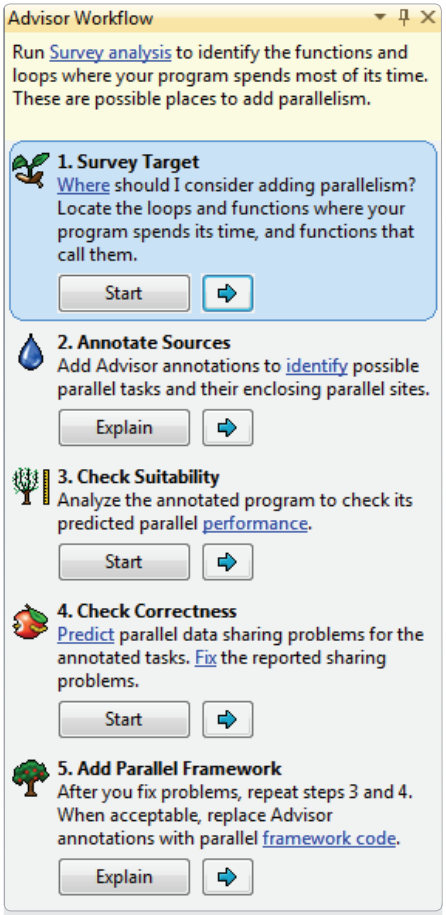


図 3: インテル® Parallel Advisor のワークフロー GUI は並列化の手順を分かりやすく示します。

Where should I add parallelism?

Summary | Survey Report | Suitability Report | Correctness Report

Function Call Sites and Loops	Total Time %	Total Time	Self Time	Source Location
Total	100.0%	276,3826s	0s	
RtlInitializeExceptionChain	100.0%	276,3826s	0s	
RtlInitializeExceptionChain	100.0%	276,3826s	0s	
BaseThreadInitThunk	100.0%	276,3826s	0s	
_tmainCRTStartup	100.0%	276,3826s	0s	crtexe.c:555
main	100.0%	276,3826s	0s	main.cpp:9
SolveEquations [loop]	100.0%	276,2665s	0s	Lattice.cpp:57
SolveEquations [loop]	98.8%	273,0998s	0s	Lattice.cpp:107
SolveEquations	98.8%	273,0998s	0,4477s	Lattice.cpp:110
OdeSolve_myocyte [loop]	96.4%	266,3364s	0s	Lattice.cpp:151
OdeSolve_myocyte	96.4%	266,3364s	123,2317s	Lattice.cpp:162
exp	44.7%	123,6009s	123,6009s	exp_wmt.c:267
log	2.9%	7,9007s	7,9007s	log_wmt.c:355

図 4: CARDIAC アプリケーションに対するインテル® Parallel Advisor の調査レポート

Where should I add parallelism? (Source)

Summary | Survey Report | Suitability Report | Correctness Report | Survey Source

File: Lattice.cpp:116 SolveEquations

Line	Source	Total Time	%	Loop Time	%
106	ANNOTATE_SITE_BEGIN(MySite1);				
107	for (i=0; i<N*N*H; i++)			273,100s	
108	{				
109	ANNOTATE_TASK_BEGIN(Task1);	0,197s			
110	OdeSolve_myocyte(Vm[i],mG[i],hG[i],jG[i],dG[i],fG[i],XG[i],Cai[i]);	267,225s			
111	OdeSolve_fib(i,Vf,FB);	5,338s			
112	ANNOTATE_TASK_END(Task1);	0,339s			
113	}				
114	ANNOTATE_SITE_END(MySite1);				
115	ANNOTATE_SITE_BEGIN(MySite2);				
116	for (i=0; i<N*N*H; i++) //this commented section integrates the cells' individuals dynamics over			0,401s	
117	{				
118	ANNOTATE_TASK_BEGIN(Task2);//(first "for" cycle). Then the coupling between the cells is added based c	0,070s			
119	Vm[i]+=dt*Im_e[i]; //currents (the second "for" cycle).	0,150s			
120	Vf[i]+=dt*If_e[i]; //The same is done by the parallel_for:Psolve	0,120s			
121	ANNOTATE_TASK_END(Task2);	0,060s			
122	}				
123	ANNOTATE_SITE_END(MySite2);				

図 5: インテル® Parallel Advisor のソース調査表示とループ時間機能

究者グループは、このループをメインのソルバーループとマージして、ループ本体の粒度を小さくすることにしました。

その並列化手法はどの程度スケーラブルなのでしょうか。その手法は時間と労力をかける価値があるものでしょうか。インテル® Parallel Advisor の適応性ツールの目的は、注釈でマークしたコードのおおよその並列パフォーマンスを予測することで、これらの問いに対する迅速な答えを提供することです。次のステップで、研究者グループは適応性ツールを実行しました。予測では、最も計算負荷の高い myocyte ループはかなりスピードアップし（32 コアで 22.79 倍のスピードアップ。図 6 を参照）、ほかのループはあまりスピードアップしない（任意のターゲット・プラットフォームでほぼ 2 倍）ことが示されました。

インテル® Parallel Advisor の適応性ツールは、ターゲットの並列フレームワーク・コードを実装するときに「タスクのチャンク化」を行うことを強く推奨しました。幸いなことに、研究者グループは、タスクのチャンク化をサポートしているインテル® スレッディング・ビルディング・ブロック（インテル® TBB）を使用することを計画していました。

プログラムはデバッグ構成を使用してリビルドされていたため、インテル® Parallel Advisor のエラー検証ツールで確認して、可能性のある並列データ共有問題を予測することができました。驚いたことに、解析の結果、ポアソンソルバー内部のマージしたループでデータ競合とメモリー再利用問題が検出され、読み取り/書き込みのコミュニケーション問題が発生しているコード領域が指摘されました（図 7）。この問題は、マージしたループを分割する（つまり、2 つの相互に関係するループを並列に実行しないようにする）ことで簡単に解決できました。こ

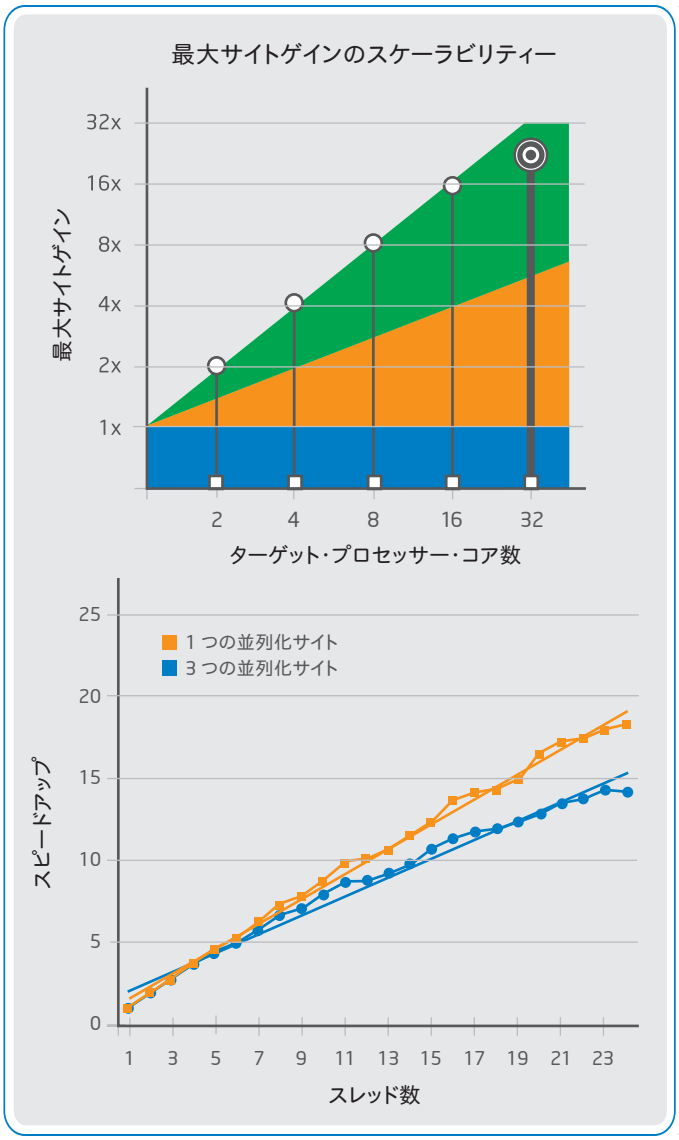


図 6: CARDIAC アプリケーション（インテル® TBB で並列化）に対するインテル® Parallel Advisor のスケーラビリティー予測と実験データ

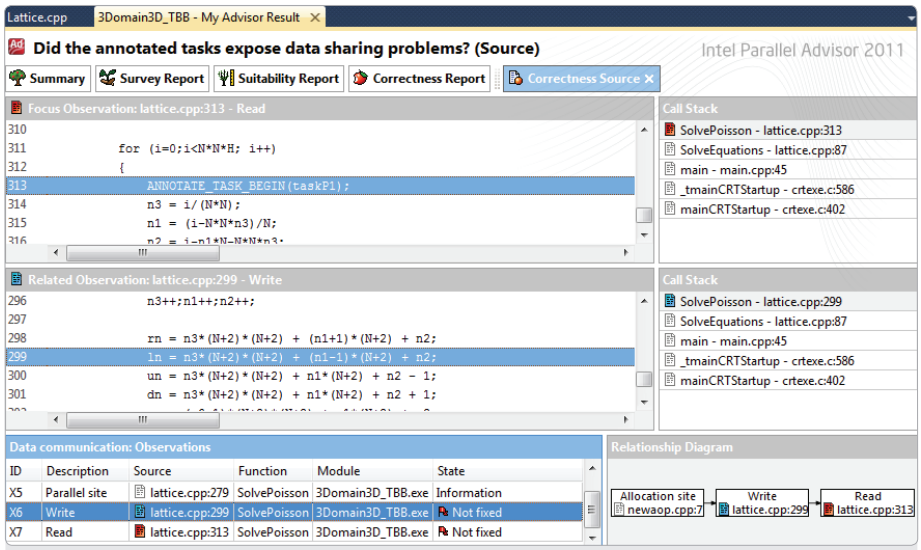
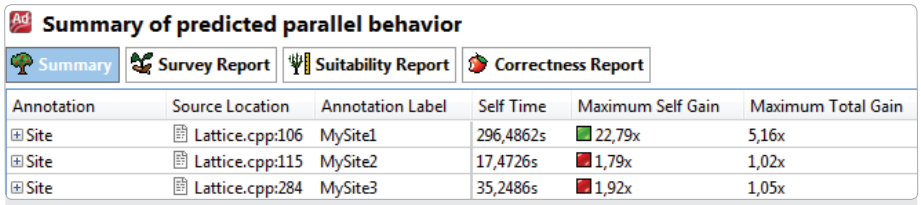


図 7: CARDIAC アプリケーションに対するインテル® Parallel Advisor のエラー検証レポート



Annotation	Source Location	Annotation Label	Self Time	Maximum Self Gain	Maximum Total Gain
Site	Lattice.cpp:106	MySite1	296,4862s	22,79x	5,16x
Site	Lattice.cpp:115	MySite2	17,4726s	1,79x	1,02x
Site	Lattice.cpp:284	MySite3	35,2486s	1,92x	1,05x

図 8: インテル® Parallel Advisor の注釈サマリーレポート

```
#1    for (ti) // 時間積分外部ループ (tî[0, T]))

#2        for (ni) // 頂点数 N のニューロン・ネットワークを反復

#3            for (ni) // 頂点数 N のニューロン・ネットワークを反復
                //ほかのニューロンとの相互作用を計算

              //求められたニューロンの特性を計算
```

図 9: NEURAL アプリケーションのプログラム構造

```
#1    for (ti) // 時間反復外部ループ

                // ロボットからセンサーデータを取得 (x, y, angle, など)

#2    for (ni)  // 頂点数 N のニューロン・ネットワークを計算
                // ニューロンの状態を計算

#3    for (ni)  // 頂点数 N のニューロン・ネットワークを反復
                // ニューロンの相互作用を計算

#4    for (ni)  // 頂点数 N のニューロン・ネットワークを反復
                // 変数を更新

          // データをロボットモーターに送信 (x, y, angle, など)
```

図 10: RATE アプリケーションのプログラム構造

の修正は適応性パフォーマンス予測にはあまり影響がなく、エラー検証ツールを再度実行したところ、問題は何も検出されませんでした。

最後のステップは、インテル® Parallel Advisor の注釈をインテル® TBB の並列フレームワーク・コードに置換して、並列アプリケーションのパフォーマンスを測定することでした。アドバイザー・サマリー GUI 機能を使用すると、前のステップで実行された内容を確認して、可能な並列化アプローチを比較し、並列フレームワーク命令への変換が実際に必要なソースコード領域を素早く特定できます (図 8)。

コードを並列化した後、インテル® Xeon® プロセッサー 7000 系を搭載した 4-way サーバー (24 コア) 上でこの並列アプリケーションのパフォーマンスを測定しました。実験データから、適応性予測は適切であることが確認できました。例えば、16 コアで 11 倍のスピードアップが示されました (図 6)。また、いくつかの並列化されたループを含むモデルがより有望であることも分かりました。アムダールの法則に従って、多くのコアを使用して最も多用するループにかかる時間を短縮できれば、短いコード領域の並列化でもパフォーマンスが大幅に向上します。

ニューラル・ネットワークのシリアル・アプリケーション

NEURAL アルゴリズムは適応システム領域に属しており、生物学、人工知能、人工ニューラル・ネットワークのさまざまなアプリケーションを利用します。Nizhny Novgorod の研究者グループは、モデルに決定論的カオスを導入することで生物学と人工システム間の既存のギャップを少なくしようと試みました。

神経細胞集団のモデリングは、頂点数 (ニューロン) が N の複雑な「全方向」ネットワーク・グラフ解析です。グラフの頂点はそれぞれ、非線形微分方程式 (Hodgkin-Huxley モデル) で記述されます。ニューロン間の相互作用も非線形微分方程式で記述されます。図 9 は、アルゴリズムを単純化して表現したものです。

速度現象論的モデルのシリアル・アプリケーション

別の広範囲なクラスのニューラルモデルは、学習と知覚目的に一般的に使用される、より単純な速度現象論的モデルの形式になっています。このモデリング手法は、リアルタイム仮想ロボット制御のプロジェクトで使用されました。

アルゴリズムは、図 10 に示すような基本的な形式になります。

NEURAL と RATE 並列化のケーススタディー

3 つのアプリケーションすべてのループ構造が同様になるため、CARDIAC で使用された並列化の手法は有望なアプローチといえます。NEURAL の場合、図 9 で示されているように、ネットワークの頂点で 1 つの内部ループを反復する、単純なソリューションになります（調査解析によれば合計シリアル・アプリケーション時間の 98.9% を占めています）。このため、研究者グループは注釈を付けて、ループ #2 と最内ループ #3 のいずれかを選択することにしました。最初にループ #3 を試しました。注釈を F_v 関数の内部に追加した後、クロスニューロンの相互作用を計算するため、適応性解析を実行しました。ところが、スケーラビリティーの予測は最大で 1.01 倍と非常に低いものでした。

次に、別のペアの注釈マクロを追加してループ #2 をテストしました。この試みは成功しました。ただし、適応性解析の結果は、4 コアと 8 コアの間でスケーラビリティーが極大になるという予測を示しました（図 11）。CARDIAC のスケーラビリティーを見た研究者グループにとって、このデータは意外なものでした。インテル® Parallel Advisor の予測が間違っているのではないかと思ったほどです。ところが、図 11 を見ると分かるように、実際のデータ（インテル® TBB で並列化したアプリケーションで計測）でも 6 コアでスケーリングが極大になり、ツールの予測と同じ結果が得られました。

CARDIAC と NEURAL でスケーリングが異なる理由は何でしょうか。適応性レポートの統計でサイトの「Average Instance Time（平均インスタンス時間）」の値を比較すると、1 つの答えが見つかります（図 12）。CARDIAC では、計算負荷の高い時間ステップ（「ヘビーな」サイト）の並列化に取り組んだため、並列実行の開始と終了処理を行う必要はほとんどありませんでした。これに反して、NEURAL ではスレッドプールが各時間ステップで作業をほとんど行わず、時間のかかるループで実行されるため、スレッド化のオーバーヘッドが大きくなります。最初にループ #3 で並列化を試したときにスケーラビリティーの予測が非常に低かったのはこのためです。ループ #2 を並列化しても、時間反復の粒度は優れたスケーラビリティーを提供できるほど大きくありませんでした。しかし、この NEURAL の例で学んだことが、次のモデルの並列化で役に立ちました。

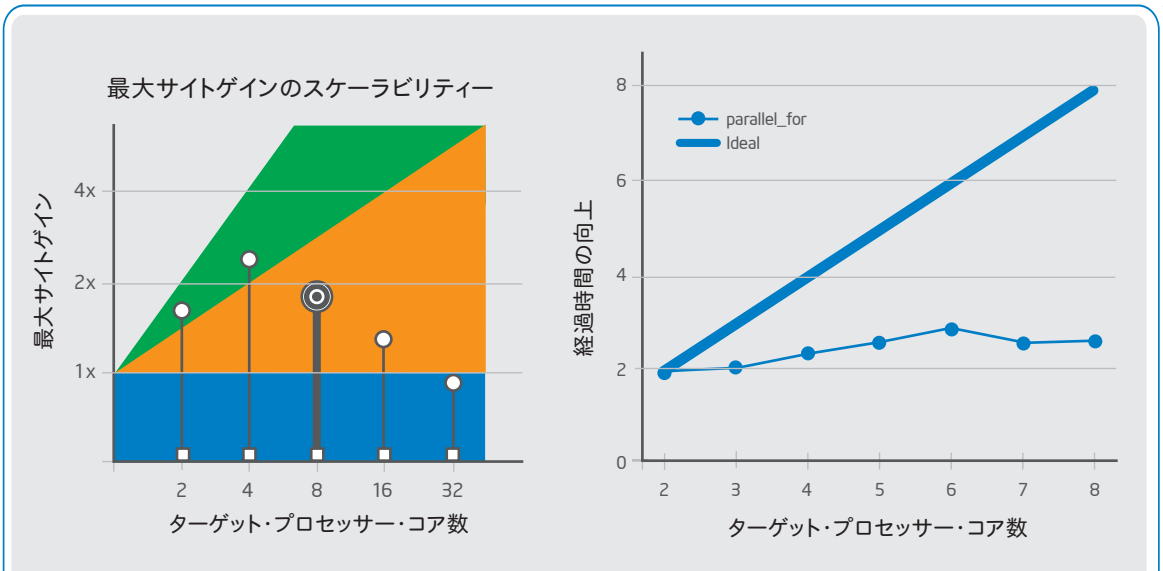


図 11: NEURAL アプリケーションに対するインテル® Parallel Advisor のスケーラビリティー予測と実験データ（実際の並列化にはインテル® TBB を使用）

アムダールの法則によれば、シリアル実行領域が非常に小さな場合でも全体のパフォーマンスに大きく影響します。

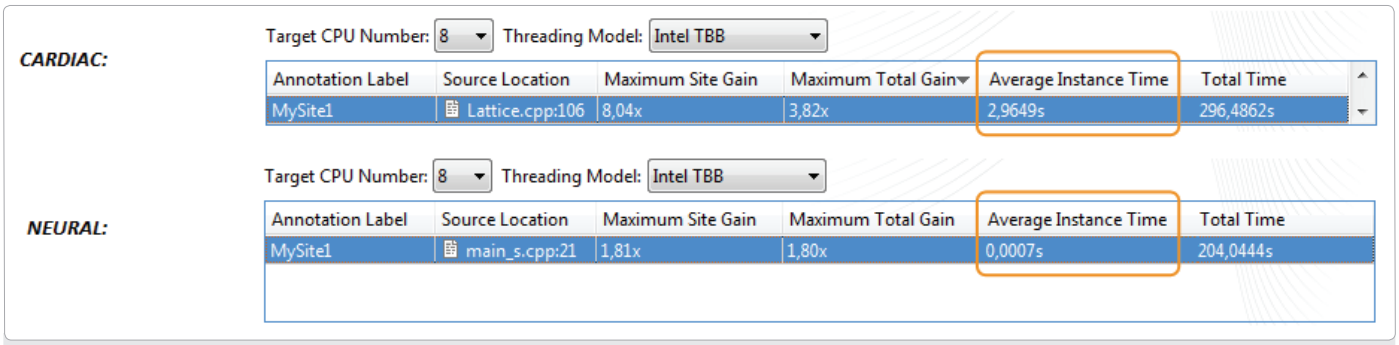


図 12: CARDIAC および NEURAL アプリケーションに対するインテル® Parallel Advisor の適応性サイト測定結果

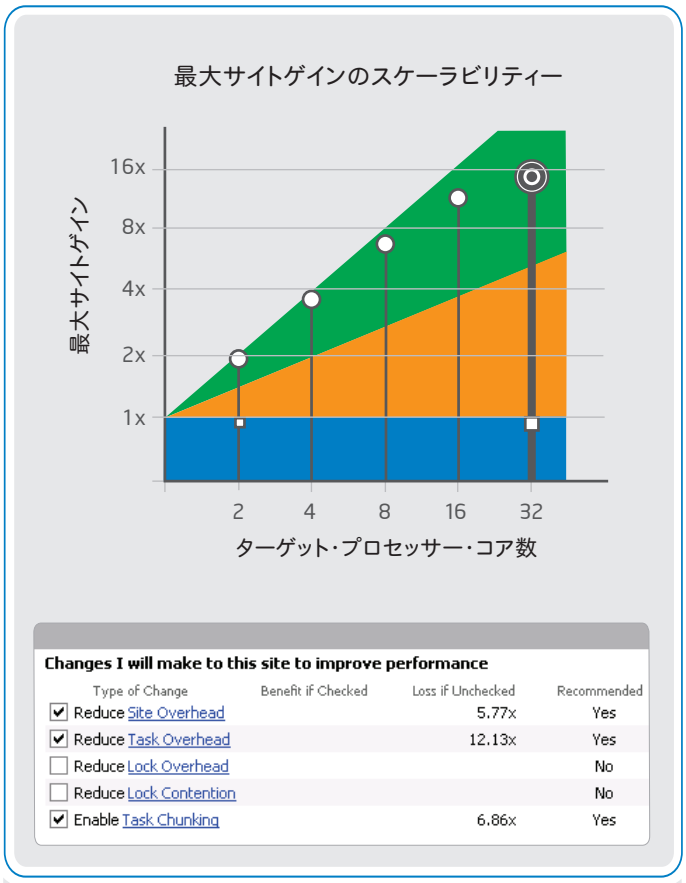


図 13: RATE モデルの適応性予測と推奨事項

アムダールの法則に従って、多くのコアを使用して最も多用するループにかかる時間を短縮できれば、短いコード領域の並列化でもパフォーマンスが大幅に向上します。

次の研究は RATE モデルで行われました。調査とソース調査のレポートでは、最初のループが時間の約半分を占めていましたが、3 つの内部ループすべてが有望であることが分かりました。レポートに基づいて、3 つの並列サイトに注釈を追加し、適応性解析を実行しました。スケーラビリティーの予測は、NEURAL より少し高いものでした。最大で 5-6 倍の予測はゼロよりはましですが、より優れたアプローチを探すことに決めました。研究者グループは、これまでのインテル® Parallel Advisor の予測から、シリアルコードの構成を少し変更することにしました。

まず思い浮かんだのは、前の研究で示されたオーバーヘッドを少なくするため、3 つのループを 1 つにして内部ループの作業量（つまり、サイトの「平均インスタンス時間」）を増やす方法でした。最小限のリファクタリングを行った後、アプリケーションは 1 つの大きな並列サイトになるように再構成されました。適応性解析を再実行したところ、スケーラビリティーの予測は 32 コアで 14 倍になりました（図 13）。

学んだこと

3 つの数学的モデルが実際には異なるにもかかわらず、3 つのアプリケーションすべての並列化手法が同様に見えることは容易に理解できます。これらの手法はすべて、並列化の直接の対象ではない外部の時間反復ループと、並列化の対象である複数の内部ループで構成されています。研究者グループは、並列化に適しているのは内部ループであり、その作業量はできるだけ大きくすべきであることを理解しました。

これは、ソリューションが反復時間積分（つまり、ソリューション精製）方式に基づく、さまざまな微分方程式ソルバーの並列化パターンとなるものです。以下の理由から、この並列化パターンは、Nizhny Novgorod の研究者グループの間でたちまち知られることとなりました。

- インテル® Parallel Advisor は、大幅なアプリケーションの修正を行うことなく、（注釈を使用して）さまざまな並列化アプローチを素早くモデル化できること。
- インテル® Parallel Advisor は、ステップごとに意思決定を簡単に行えるように推奨するアプローチと測定結果を示し、必要な並列化の段階で効率的な手法が提示されること。
- インテル® Parallel Advisor の予測が良くない場合でも、適応性解析とエラー検証ツールの結果を参考にして、研究者グループがオリジナルのコードを再構成するという正しい方向に進むことができたこと。
- 同じグループの研究者がいくつかの並列化実験に同時に参加して、これらのさまざまな並列化に取り組むときに得た経験を共有し、再活用し、まとめる支援をしたこと。

G. Osipov 教授の並列化研究により、異なる 3 つの HPC アプリケーションを並列化するときにインテル® Parallel Advisor の機能と手法が役立つことが証明されました。インテル® Parallel Advisor のツールは、並列に実行することでパフォーマンスが向上する領域の検出、データ共有問題の特定と修正、並列プログラム構造と潜在的な利点のモデル化を支援します。インテル® Parallel Advisor の予測精度と効率は実際の実験データによって検証されました。

Nizhny Novgorod では、現在もさまざまなモデリング・アルゴリズムを並列化する研究が行われています。研究者グループがアプリケーションのパフォーマンスを解析して向上させるにあたって、インテル® Parallel Studio の強力な機能が役立っています。また、インテル® Parallel Advisor の改良すべき点も報告されています。

BLOG highlights

インテル® Fortran Studio XE 2011 の紹介

開発者向け製品部門 **STEVE LIONEL**

インテル® Parallel Studio XE が最初にリリースされた 2010 年 11 月のことは今でも鮮明に記憶しています。このハイパフォーマンス・コンピューティング開発ツールのスイートには、新しいバージョンのインテル® C++ / Fortran コンパイラー製品（現在の名前は Composer XE）と 2 つの新しい解析ツール、インテル® VTune ™ Amplifier XE およびインテル® Inspector XE が含まれています。この 2 つの解析ツールは、2009 年に C/C++ の Windows* 版で提供されたインテル® Parallel Amplifier とインテル® Parallel Inspector を大幅に更新したものです。新しい XE ツールは、Fortran をサポートするだけでなく、初めて Linux* 用にも提供されました。

Fortran プログラマーの多くはコンパイラーの新しい機能をとても気に入っていたようですが、不満の声を聞くこともありました。ご存知のように、C++ コンパイラーと解析ツールを含むインテル® C++ Studio XE は以前からありましたが、Fortran プログラマー向けの対応する製品は存在していませんでした。解析ツールが必要な Fortran ユーザーは、これらのツールを別々に購入するか、C++ コンパイラーを含むインテル® Parallel Studio XE 製品を購入する必要があったのです。「Fortran Studio XE はないんでしょうか？」もちろん、あります。ピッチフォーク分岐や算術 IF を利用可能なインテル® Fortran Studio XE 2011 Linux* 版と Windows* 版がついにリリースされたのです。

この記事の続きはこちら（英語）

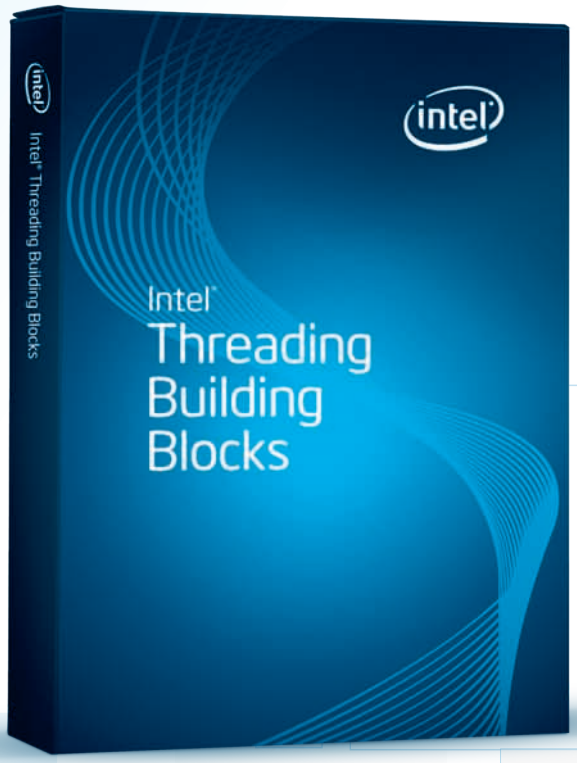
Go-Parallel.com（英語）でご覧になれます。

Go Parallel では、この他にも次のような関連トピックのブログをご覧いただけます :Translating Multicore Power into Application Performance（マルチコアのパワーでアプリケーション・パフォーマンスを引き出そう）。

インテル®
スレッディング・ビルディング・ブロック
FLOW GRAPH

ソフトウェア・アーキテクト
Michael J. Voss, Ph.D.

ユーザーのフィードバックに基づいてインテル® スレッディング・ビルディング・ブロックのフローグラフ機能が改良され、プログラマーは静的 / 動的依存性グラフ、反応 / イベントベースの各グラフを表現できるようになりました。



シニア・ソフトウェア・
エンジニア
MICHAEL J. VOSS

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) 4.0 は、フローグラフ機能をフルサポートしています。フローグラフは、静的/動的依存性グラフ、およびデータメッセージに応答してデータメッセージを渡す反応グラフをサポートしています。インテル® TBB 3.0 Update 5 で、Community Preview 機能[1] が導入され、数ヶ月におよぶユーザーからのフィードバックに基づいてフローグラフのインターフェイスが改良されました。

実際に、メディア、ゲーム、金融サービス、技術計算など、さまざまな分野の開発チームが、インテル® TBB 3.0 の Community Preview 機能としてフローグラフを評価してきました。フローグラフが利用可能になる前は、いくつかのイベントベースの反応プログラムをインテル® TBB を使用して実装することは非現実的でした。そのため、ユーザーは、低水準のインテル® TBB のタスク処理インターフェイスを直接使用して複雑なコードを記述するか、インテル® TBB のパイプラインを使用して過度な並列化を行っていました。フローグラフは、ほかのインテル® TBB ベースのソリューションのパフォーマンスを損なうことなく、多くのアプリケーションにより適した機能を提供します。

フローグラフ・インターフェイスの概要

インテル® TBB のフローグラフは、graph オブジェクト、ノード、エッジの 3 つの主要なコンポーネントで構成されます。graph オブジェクトは、グラフのコンテキストのタスクを実行し、グラフが完成するのを待つ方法を提供します。ノードは、メッセージを生成、変更、バッファーします。エッジは、メッセージを受け取るノードとメッセージを送信するノードを接続して、グラフの線をつなぎます。図 1 で示されているように、さまざまな種類のノードが用意されています。ユーザーコードを実行する関数ノード、バッファリング・ノード、メッセージの結合/分割ノード、その他のノードがあります。

依存性グラフの例

図 2 は、continue_node オブジェクトのセットを使用してウェーブフロント計算を実装するアプローチを示しています。この例で、各ノードは、計算を開始する前に、上のノードの計算と左のノードの計算が完了するのを待つ必要があります。continue_node は、前のノードから continue_msg を受け取ると計算を開始します。

図 3 で、このアプローチは、各計算が BxB ブロックの行列の value を更新する、ブロック・ウェーブフロント計算を実装するために使用されています。関数 run_graph の for ループは、continue_node オブジェクトのセットを作成します。continue_node コンストラクターは、graph オブジェクト g のリファレンスと、そのブロックの update_block を呼び出す関数オブジェクト(この場合はラムダ式)を渡されます。

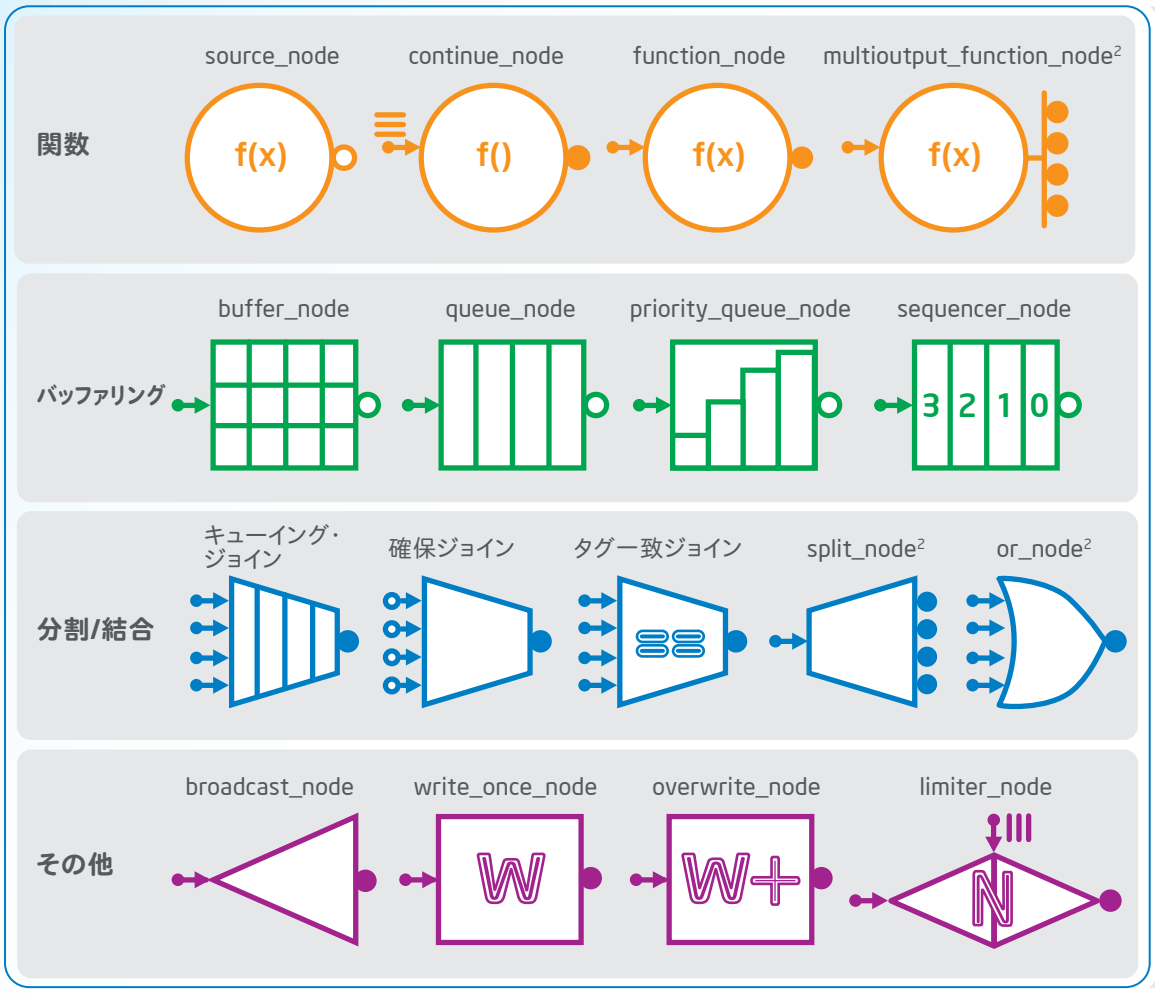


図 1: インテル® スレッディング・ビルディング・ブロックのフローグラフでサポートしているノードの種類

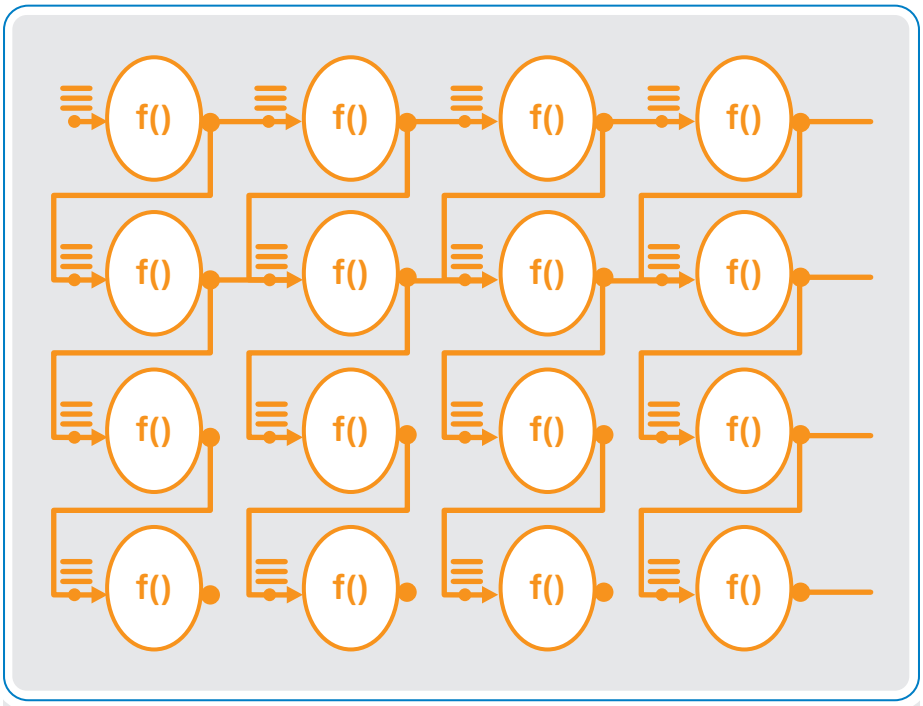


図 2: インテル® TBB のフローグラフを使用したウェーブフロント計算の表現

```
// M と N は行列の行と列の数
// MB と NB は行と列のブロックの数
// B はブロックサイズ(BxB の平方数)

using namespace tbb;
using namespace tbb::flow;

double value[M][N];

graph g;
continue_node<continue_msg> *node[MB][NB];

double run_graph( ) {
  value[M-1][N-1] = 0;
    for( int i=MB; --i>=0; ) {
     for( int j=NB; --j>=0; ) {
      node[i][j] =
       new continue_node<continue_msg>( g,
         [=]( const continue_msg& ) { update_block( i, j ); } );
       if ( i + 1 < MB ) make_edge( *node[i][j], *node[i+1][j] );
       if ( j + 1 < NB ) make_edge( *node[i][j], *node[i][j+1] );
    }
  }
  node[0][0]->try_put(continue_msg());
  g.wait_for_all();
  for( int i=0; i<MB; ++i )
     for( int j=0; j<NB; ++j )
          delete node[i][j];
  return value[M-1][N-1];
 }
```

図 3: ブロック形式のウェーブフロント計算の実装

フローグラフが設定されると、ウェーブフロント計算を開始するため、左上隅のノード node[0][0]に continue_msg が入れられます。g.wait_for_all()は、ウェーブフロント計算全体が完了するまで待機します。

この例の説明とソースコードは、http://software.intel.com/en-us/blogs/tag/flow_graph(英語)でご覧になれます。

メッセージグラフの例

図 4 は、インテル® スレッディング・ビルディング・ブロックのフローグラフを使用した単純な機能検出アプリケーションを示しています。多くのイメージがグラフに入力され、2 つの代替機能検出アルゴリズムがそれぞれ適用されます。いずれかのアルゴリズムが機能を検出すると、後の検査のためにイメージが格納されます。

図の source_node オブジェクト src は、確保ジョインノード resource_join にイメージを提供します。resource_join の 2 つ目の入力は、イメージバッファーのキュー buffers に接続されています。source_node は、現在の出力が消費された後に新しい項目を生成します。確保ジョインノードは、各入力ポートの入力が確保できるまで、入力項目を消費しません。そのため、このグラフのフロントはメモリー使用を制御するように構築されています。イメージバッファーがペアの buffers で利用可能であれば、新しいイメージが src によって生成されます。

入力イメージがバッファーとペアになると、イメージを前処理する function_node、 preprocess_function に送られ、関連するバッファーに結果が配置されます。preprocess_function は、複数のイメージを同時に処理できるように、並列に作成されます。例えば、機能検出アプリケーションでは、この前処理にイメージをぼかすアルゴリズムが含まれます。

preprocess_function の出力は、イメージの機能を検出する 2 つの代替アルゴリズムが実装された detect_A と detect_B のエッジで接続されます。これらのノードは、各ノードで複数のイメージを処理できるように、並列に作成されます。これらの検出ノードの出力は、タグ一致ジョインノード、detection_join に送られます。タグ一致ジョインノードは、一致するタグに基づいて項目をペアにします。この例では、処理したイメージに基づいて、detect_A と detect_B の出力をペアにします。複数のイメージがグラフで同時に使用されるかもしれないため、適切な結果と一致するように、ここでタグ一致ジョインを使用することが重要です。

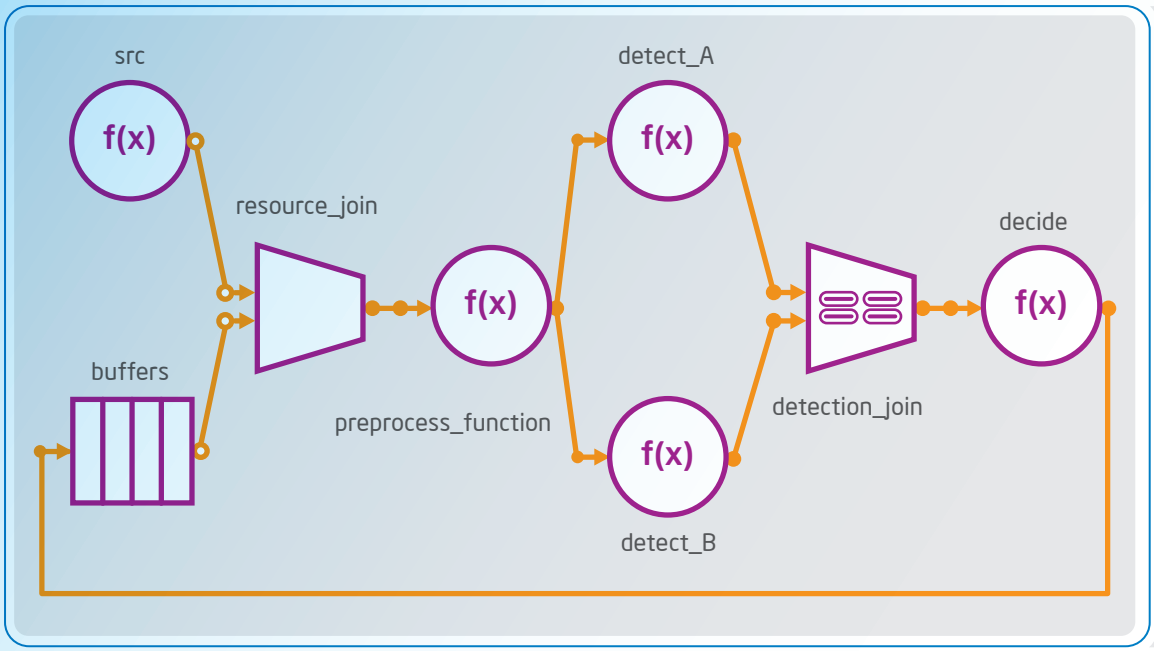


図 4: インテル® TBB のフローグラフを使用した機能検出の例

	タスクグラフ	パイプライン / parallel_pipeline	フローグラフ
表現力	閉路のない依存性グラフを表現可能	線形パイプラインを表現可能	閉路のない依存性グラフ、閉路のないメッセージグラフ、環状メッセージグラフを表現可能
使いやすさ	低レベルの粒子登録コードと明示的なタスクのスポーンが必要	簡潔でタイプセーフなインターフェイス	parallel_pipeline より冗長だが、明示的な粒子登録やタスクスポーンの必要がない
永続性	グラフは破壊的に実行され再実行できない	複数回実行可能 (パイプラインにのみ適用)	複数回実行可能
パフォーマンス [3]	タスク上で直接構築され破壊的に実行されるため、オーバーヘッドが非常に低い	キャッシュの局所性を最適化するため、後入れ先出し (LIFO) タスク・スケジューリングを使用。オーバーヘッドはフローグラフと同等。	グラフでメッセージのフローを保つため、 先入れ先出し (FIFO) タスク ・ スケジューリングを使用。オーバーヘッドはパイプラインおよび parallel_pipeline と同等。

表 1: タスクグラフ、parallel_pipeline、フローグラフの比較

最後に、結果のペアが function_node オブジェクト decide に到達します。機能がイメージに存在するかどうか確認するため、各アルゴリズムからの結果を検査します。イメージが存在する場合、後の検査のためにイメージを保存します。decide が完了すると、別の入力イメージとペアにできるように、バッファーを buffers に戻します。

この例の説明とソースコードは、http://software.intel.com/en-us/blogs/tag/flow_graph（英語）でご覧になれます。

フローグラフ、パイプライン、タスクの閉路のないグラフの選択

フローグラフはインテル® スレッディング・ビルディング・ブロック 4.0 の重要な機能ですが、フローグラフに適した一部のアプリケーションは、既存のタスクの閉路のないグラフおよび汎用 parallel_pipeline アルゴリズムを使用して実装することもできます。**表 1** は、これらの異なる機能の比較です。最適なモデルを選択できるように、機能の特長についても示しています。

まとめ

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）4.0 は、フローグラフ機能をフルサポートしています。フローグラフは、静的/動的依存性グラフ、計算間でメッセージを渡す反応グラフまたはイベントベースのグラフに使用できます。インテル® TBB 4.0 ライブラリーは、www.threadingbuildingblocks.org からダウンロードできます。

1. Community Preview 機能として、「フローグラフ」という名称は一旦「グラフ」に変更されました。しかし、この機能がアプリケーションの制御フローを表現することを強調するため、再度フローグラフという名称を使用することになりました。より汎用的な名称である「グラフ」は、データ構造中心のアプローチや従来のグラフベースのアルゴリズムのコレクションのように間違った意味で解釈されることがあったためです。
2. multioutput_function_node、split_node、or_node は、インテル® TBB 4.0 の Community Preview 機能です。
3. インテル® ソフトウェア製品のパフォーマンスおよび最適化に関する注意事項については、http://software.intel.com/en-us/articles/optimization-notice#opt-jp を参照してください。

BLOG highlights

tbb::graph の内部の理解：Push と Pull のバランス

シニア・ソフトウェア・エンジニア
MICHAEL J. VOSS, PH.D

この記事では、インテル® スレッディング・ビルディング・ブロックの Community Preview 機能で使用されているハイブリッド push-pull プロトコルについて説明します。

tbb::graph で使用されているハイブリッド push-pull プロトコルは、コミュニケーションにバイアスをかけることで、ポーリングを防ぎ、不要な再試行を減らします。tbb::graph を使用するためにこのプロトコルの詳細を理解する必要はありませんが、そのパフォーマンスをより簡単に理解できます。

グラフのノードはユーザーが明示的に破棄するまで存在しますが、一部のアクターシステムとは異なり、スレッドは tbb::graph の各ノードに割り当てられません。タスクは実行ノードの本体でオンデマンドに作成され、グラフにアクティビティーがある場合、ノード間のメッセージを渡します。そのため、tbb::graph ノードはメッセージの到着を待つためにループでスピンしません。代わりに、メッセージが到着すると、受信ノードの本体を入力メッセージに適用するタスクが作成されます。

この記事の続きはこちら（英語）

Go-Parallel.com（英語）でご覧になれます。

Go Parallel では、この他にも次のような関連トピックのブログをご覧いただけます :Translating Multicore Power into Application Performance（マルチコアのパワーでアプリケーション・パフォーマンスを引き出そう）。

インテル® Parallel Studio XE SP1

インテル® Parallel Studio XE は、業界最高レベルのインテルの C/C++ コンパイラーと Fortran コンパイラー、ハイパフォーマンスな並列ライブラリー、エラーチェック、コードの安定性、パフォーマンス・プロファイリングなどのテクノロジーを組合わせたツール・スイートです。SP1 リリースでは、マルチコアからメニーコア・ハードウェア・プラットフォームへの移行を簡略化する機能が追加されました。

Michael D'Mello

非常に優秀なソフトウェア開発者達からも、パフォーマンスや正当性についてはよく質問があります。「現在または次世代のハードウェアではソフトウェアをより速く実行できるようになりますか?」「何がコードのパフォーマンスを制限しているのでしょうか?」「エラーやセキュリティーの脆弱性にこのソフトウェアはどの程度対応できそうですか?」

何年もの間、多くの開発者がインテルのさまざまなソフトウェア・ツールを使用してこれらの疑問に取り組んできました。2010 年に、開発者コミュニティーの関心は、新世代のソフトウェア・ツール、インテル® Parallel Studio XE に移りました。この洗練されたツール群は、ユーザー体験を向上させながら広範囲の機能を提供することを目標に設計されており、パフォーマンスのボトルネック、メモリーエラー、スレッド化エラー、セキュリティー問題を可能な限り容易に特定、分類、修正します。将来のマルチコアおよびメニーコア・ハードウェアに対応できるように、このツール群には、業界最高レベルの C++ コンパイラー と Fortran コンパイラー および並列プログラミング・モデルのセットも含まれています。

インテル® Parallel Studio XE のコンポーネント

インテル® Parallel Studio XE を成功に導いた根本的なテクノロジーと、このツールキットの SP1 リリースでスイートに追加される機能を検証しておくことは重要です。

インテル® Parallel Studio XE には以下の機能が含まれています。

- インテル® C++ コンパイラー XE およびインテル® Fortran コンパイラー XE[1]
  - インテル® マス・カーネル・ライブラリー、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® スレッディング・ビルディング・ブロック
- インテル® VTune™ Amplifier XE パフォーマンス・プロファイラー
- インテル® Inspector XE の動的メモリー / スレッドチェッカー
- インテル® スタティック・セキュリティー解析のスタティック・エラー / セキュリティー・チェッカー

優れたコンパイラーが必要であることは明らかです。インテル® コンパイラーは、最高レベルのシングルコアのパフォーマンスとマルチコアのスケーラビリティーを提供します。コンパイラーは常に、最新のハードウェア・プラットフォームの機能に合わせて更新されます。最近の例としては、インテル® アドバンスト・ベクトル・エクステンション (インテル® AVX) のサポートがあります。このベクトル・レジスター・テクノロジーは、インテル® マイクロアーキテクチャー、Sandy Bridge (開発コード名) 以降の最新のプロセッサーで利用できます。このテクノロジーは、既存のストリーミング SIMD 拡張命令 (SSE) フォーマットと比較してパフォーマンスが 2 倍になります。SP1 リリースでは、AVX と Sandy Bridge プラットフォームのサポートをチューンアップしました。また、インテル® Parallel Studio XE の以前のリリース同様、Windows*、Linux*、Mac OS* X オペレーティング・システムで同じツールのセットが提供されます。

ガイド付き自動並列化

最新のマルチコアおよびメニーコア・ハードウェアの広範囲なサポートのほかに、ユーザーが見落としがちなインテル® コンパイラーの要素として、従来の -O1、-O2 スイッチよりも優れた最適化モードがあります。この最適化テクノロジーの 1 つであるガイド付き自動並列化 (GAP) は、ソースコードを変更するためにコンパイラーによるガイダンスを提供するワークフロー指向のアプローチで、コードをベクトル化、並列化、データ変換してコンパイルすることによりパフォーマンスを向上させます。ソースコード変更のアドバイスとコンパイラー宣言子 (例えば、プラグマ) の追加に加えて、GAP はコンパイラー・オプションに関するアドバイスも提供します。GAP は柔軟性があり、インテル® コンパイラーで提供されるほかの 2 つの最適化モードである、プロシージャー間の最適化 (IPO)[2] およびプロファイルに基づく最適化 (PGO)[3] と組合わせて使用できます。

マルチコア対応ライブラリー

リストの次にあるのは、インテル® マス・カーネル・ライブラリー (インテル® MKL) とインテル® 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) ライブラリーです。これらのマルチコア対応ライブラリーは、コードの並列化とパフォーマンス向上における最も容易かつ直接的なメカニズムを提供します。科学/工学アプリケーションで多用されるインテル® MKL は、エネルギー、ヘルスケア、金融解析、ハイパフォーマンス・コンピューティング (HPC) 分野において重要なライブラリーです。インテル® IPP ライブラリーは、マルチメディア、データ処理、通信分野のソフトウェアの最適化において同様の役割を果たします。これらのライブラリーは、最高のシングルコアおよびマルチコア・パフォーマンスを達成するため、ベクトル化とスレッド化をそれぞれ最大限に利用しようとします。SP1 リリースで、これらのライブラリーは、AVX と Sandy Bridge マイクロアーキテクチャー固有の最適化機能をシームレスにサポートするように拡張されました。

ガイド付き自動並列化は、ソースコードを変更するためにコンパイラーによるガイダンスを提供するワークフロー指向のアプローチで、コードをベクトル化、並列化、データ変換してコンパイルすることによりパフォーマンスを向上させます。

C/C++ 最適化コンパイラーには、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）とインテル® Cilk™ Plus が含まれます。インテル® TBB は、C++ 言語でタスクベースの並列化をサポートする機能を提供して、コンパイラーがベクトル化を行うようにします。インテル® Cilk™ Plus は、タスク、ベクトル、データ並列のための機能を提供します。どちらの機能も、同じプラットフォームに異なる種類のコア（つまり、ヘテロジニアス・コア）を含むハードウェアの進歩に関連しています。Fortran 開発者のために、インテル® Parallel Studio XE は Co-Array Fortran をサポートしています。SP1 リリースでは、Fortran 2008 標準規格もサポートしました。

インテル® TBB の C++ テンプレート・ライブラリーは、2006 年の登場以来、多くの C++ 開発者に幅広く採用されてきました。このタスクベースの並列ライブラリーは、スレッドプールとタスク・スケジューラーを内部的に管理します。タスク・スケジューラーは、ユーザーが作成したタスクをライブラリーが管理するスレッドのプールにマップします。スケジューラーは、タスクを特定のスレッドに関連付けます。この結果、いくつかの重要な最適化機能がライブラリーの機能として直接実装されます。例えば、同様のタスクはしばしば同様のデータを処理および消費します。これらのタスクを特定のスレッドに関連付けることで、ライブラリーは、ある程度、これらのタスクで要求されるデータがキャッシュに利用可能なまま（つまり、「ホット」な状態で）残されることを保証できます。SP1 リリースでは、インテル® TBB ライブラリーにフローグラフ機能が追加されました（Michael Voss の記事を参照）。開発者は、求められる機能を表すグラフに注目することで、並列化を導入できるようになります。この高レベルの観点により、ライブラリーに含まれるパフォーマンスを向上する機能を使用して、実装時間の大幅な短縮を期待できます。

パフォーマンス・プロファイラー

インテル® VTune™ Amplifier XE は、現在業界で利用可能な最も完全なプロファイラーの 1 つです。このツールの機能は、タイムベース・プロファイルとイベントベース・プロファイルの 2 つに分かれています。タイムベースの機能は、コールスタックを含むコードの従来のプロファイリングと、マルチスレッド・コード向けの「コンカレンシー」解析および「ロックと待機」解析をカバーしています。タイムベースのプロファイルで必要なすべての情報を取得するため、ツールは、ユーザーがハードウェア・カウンターを使用して、コードの実行とともに生成されるさまざまなプロセッサー・イベント（例えば、リタイアした命令、キャッシュミス、TLB ミスなど）を追跡できるようにします。この機能はイベントベース・サンプリング（EBS）と呼ばれ、インテル® プロセッサーで直接サポートされているため、オーバーヘッドが非常に低くて済みます。ユーザーは、ソフトウェアの指定した領域がハードウェアでどのように実行されているか、高度で詳細な情報を得られます。この情報は、プログラムがどのように実行されているかを理解するだけでなく、プログラムがどの程度適切に記述されているかを理解するために重要です。

また、インテル® VTune™ Amplifier XE の優れた機能の 1 つである「フレーム解析」は、コードのプロファイルのタイムライン・ビューをマーキングするために使用されます。基本的に、タイムライン・ビューは、スレッドの時間的な変遷とスレッド間の対話を示したものです。スレッドは、オペレーティング・システムのオブジェクト（ミューテックス、ロックなど）を通じて対話を行い、対話はスレッド間で描かれた線によってタイムラインで示されます。タイムライン・ビューはロードバランスを理解するための基礎となるものであり、前述の「コンカレンシー」解析および「ロックと待機」解析の不可欠な部分です。ただし、連続で動作するコードで、スケジュールの部分のみに関心があるユーザーは、フレーム解析を利用してタイムラインをマークすることで関心領域を特定できます。コンピューター・ゲームや金融トレードエンジンは連続で動作するコードの良い例です。この他にも、さまざまなアプリケーション分野の多くのコードが連続で動作するコードに分類されます。

完全なコードを目指す

コードは完全ではないため、統合ツールセットは、エラーチェックやエラー検出をサポートする必要があります。インテル® Inspector XE には、メモリーチェックとスレッドチェックの機能が用意されており、メモリーリーク、競合状態、デッドロックなどのエラーを追跡します。インテル® Inspector XE を使用して、インテル® C++ コンパイラーのソースレベルのエラーチェックの結果を視覚化することもできます。この機能はインテル® スタティック・セキュリティー解析と呼ばれ、ソースコードのエラーとセキュリティーの脆弱性を検証します。スイートのこれらのコンポーネントにより、コードの安定性を強化して、コードの全体的な品質を向上させる強力なメカニズムが提供されます。

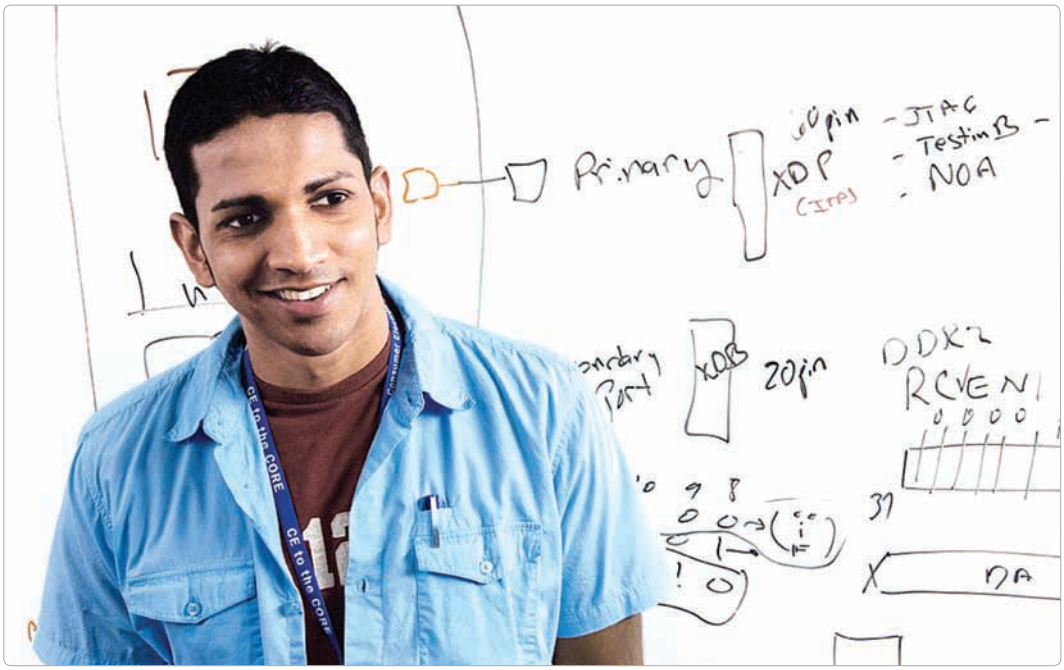
タイムライン・ビューはロードバランスを理解するための基礎となるものであり、「コンカレンシー」解析および「ロックと待機」解析の不可欠な部分です。

まとめ

インテル® Parallel Studio XE は、利用可能な最新のマルチコアおよびメニーコア・プラットフォームでより優れたコードを記述できるように、ソフトウェア開発者を支援するツール・スイートです。コードの品質、安定性、セキュリティー、パフォーマンス、スケーラビリティーの疑問に直に取り組みます。各コンポーネントの優れたデザインと応答性により、ポジティブで非常に有益なエンドユーザー体験が得られます。SP1 リリースでは、開発者が各自のソフトウェア・ソリューションおよび企業の価値を高めることができる、非常に便利なメカニズムが追加されています。

1. ほかのバリエーションも提供しています。例えば、リストの 3 つ目（インテル® スレッディング・ビルディング・ブロック）までのコンポーネントとインテル® スタティック・セキュリティー解析機能を含むセットが、インテル® Composer XE です。インテル® C++ Composer XE にも、同じセットが含まれています。ただし、インテル® Composer XE に含まれている Fortran コンポーネントは含まれていません。インテル® C++ Composer XE にリストの 4 つ目（インテル® VTune™ Amplifier XE）と 5 つ目（インテル® Inspector XE）を含むセットが、インテル® C++ Studio XE です。Fortran ユーザー向けのセット、インテル® Fortran Studio XE もあります。
2. 「プロシージャー間の最適化」とは、ほかの最適化テクニックで一般的な単一の関数やコードブロックの解析ではなく、プログラム全体の解析に基づくコンパイラーによる最適化テクニックです。
3. 「プロファイルに基づく最適化」は、より最適化された実行ファイルを生成するためのコンパイラー・テクノロジーです。代表的なデータセットまたはワークロードを使用してオリジナルの実行ファイルを実行した結果に基づいて、最適化されたバージョンを作成する情報を設定します。この情報に基づいて、コンパイラーは、オリジナルの実行ファイルよりも高速に実行される最適化された実行ファイルの生成を試みます。

このソフトウェアの評価版は、http://intel.com/software/products（英語）からダウンロードできます。



最適化に関する注意事項

インテル® コンパイラーは、互換マイクロプロセッサー向けには、インテル製マイクロプロセッサー向けと同等レベルの最適化が行われない可能性があります。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® SSE2) 、インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3) 、ストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。インテルでは、インテル製ではないマイクロプロセッサーに対して、最適化の提供、機能、効果を保証していません。本製品のマイクロプロセッサー固有の最適化は、インテル製マイクロプロセッサーでの使用を目的としています。インテル® マイクロアーキテクチャーに非固有の特定の最適化は、インテル製マイクロプロセッサー向けに予約されています。この注意事項で対象としている特定の命令セットに関する詳細は、該当製品のユーザーズガイドまたはリファレンス・ガイドを参照してください。

改訂 #20110804