# Python* と C++ 分析によるシミュレーションと後方伝播の高速化

## 言語間の障壁を超えるグラフ・コンパイラー

**Dmitri Goloubentsev MatLogica 最高技術責任者**

AADC（Algorithmic Adjoint Differentiation Compiler、自動随伴微分コンパイラー）は、計算最適化の最前線に立っています。元々は、計算集約型タスクを効率良く処理できることから C++ 向けに設計されましたが、現在では当初の目的を超えて、ハイパフォーマンス・コンピューティングを必要とするシミュレーションや推論タスクなど、幅広いアプリケーションを高速化しています。演算子のオーバーロードを使用して有向非巡回グラフ（DAG）を効果的に抽出する AADC は、計算グラフが膨大な数のノードで構成される環境で優れた性能を発揮します。高密度テンソル演算を処理する一般的なマシンラーニング（ML）やディープ・ニューラル・ネットワーク・フレームワークとは異なり、AADC は主にスカラー演算で構成される非常に大規模な計算グラフを迅速にコンパイルすることに特化しており、複雑で大量の計算を迅速に処理する必要があるドメインに不可欠なツールです。

特に Python* では、主に ML およびディープラーニング（DL）アプリケーションのニーズによって計算ツールが形成されてきました。これらのアプリケーションは、テンソル演算の処理に優れたフレームワークを中心としています。代表的な例として、ニューラル・ネットワーク計算を処理する TensorFlow* や PyTorch* などがあります。ただし、主要なドメイン以外に適用する場合、これらのフレームワークには固有の制限があります。その 1 つは、メモリー内に完全な DAG を格納する必要があることです。

多くの計算タスク、特に金融におけるシミュレーションや随伴微分法を含むタスクでは、大きなグラフを素早くコンパイルする能力がパフォーマンスに直接影響します。これらのコンテキストでは、ジャストインタイム（JIT）コンパイルが重要になりますが、JIT の利点は、コンパイルが高速でなければ十分に発揮されません。コンパイルが遅いと、JIT によってもたらされるはずのランタイム・パフォーマンスの向上が打ち消される可能性があります。

## AADC アーキテクチャー

AADC は、ストリーミング・グラフ・コンパイラー・フレームワークを実装することで、高性能な計算タスクの厳しいニーズを満たすように設計されています。この革新的なアーキテクチャーは、ユーザープログラムの実行時に x64 インテル® AVX2/ インテル® AVX-512 マシンコード命令を動的に生成することで機能します。このメカニズムは、ベクトル化されたカーネルをコンパイルして、入力から出力までユーザー関数を複製するだけでなく、随伴または後方伝播も管理します。これにより、AADC は小さなメモリー・フットプリントを維持しながら、高いパフォーマンスを実現できます。**図 1** はこれを図解したものです。

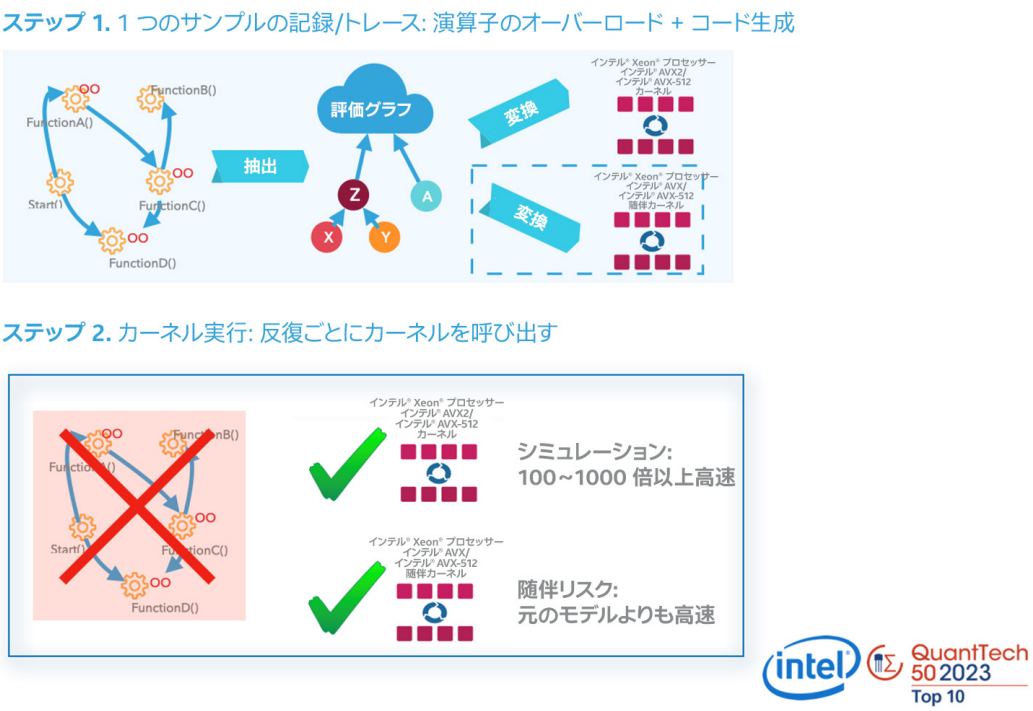


**図 1. AADC**

AADC 設計の中核は、基本操作を直接トレースし、プログラムの実行時にマシンレベルの命令に再コンパイルする機能です。このプロセスにより、高レベルのプログラミング言語の非効率性が排除されます。操作を x64 マシンコードに直接変換することで、AADC はマルチスレッド、インテル® AVX2、インテル® AVX-512 などの最新のプロセッサー機能を活用して、計算タスクを最大限の効率で実行できます。

AADC の重要なアーキテクチャー上の利点は、高度なコード折り畳みおよび圧縮技術です。従来の自動微分ツールは、大量の中間データを格納する必要があるため、メモリー使用量が大きくなりがちです。これに対し、AADC は高度なアルゴリズムを使用して、計算グラフと中間結果の格納に必要なメモリーを最小限に抑えます。これは、コードをインテリジェントに折り畳み、圧縮することで実現され、メモリー要件が軽減されるだけでなく、計算中に処理および移動するデータ量が軽減されるため、実行速度が向上します。

AADC は、C++ と Python* の両方の分析で計算をトレースする「アクティブタイプ」を採用し、さまざまなプログラミング環境間で計算操作の一貫した正確な記録を維持します。これにより、すべての計算が、言語に関係なくキャプチャーされ、最適化されます。ユーザーは、トレースをトリガーし、入力データの 1 つのインスタンスで JIT コンパイルを使用することで、このプロセスを開始します。コンパイルされた DAG は、複数のサンプルを処理するカーネルとして機能し、高レベルのプログラミング構造のオーバーヘッドなしで、低レベルのマシンコード速度で実行されます。

## ケーススタディー：XVA パフォーマンスの強化

信用評価調整（XVA）計算は、金融市場における信用リスクと評価の管理に不可欠です。これらの計算はリソースを大量に消費することで有名で、リスクと価値を正確に評価するには堅牢なシミュレーションと価格設定モデルが必要です。このデモでは、MatLogica AADC を XVA シミュレーション・フレームワークに統合することで実現されるパフォーマンスの向上を示します。問題の XVA 計算では、Python* と C++ の組み合わせが使用され、Python* はオーケストレーション・レイヤーとして機能し、QuantLib ライブラリー（英語）をバックエンドとしてシミュレーション・モデルを価格設定関数に接続します。

元の Python* コードは必要な機能を備えていますが、ベクトルまたはマルチコア実行に対応していません。その結果、パフォーマンスは最適ではなく、リスク計算なしで 1 回の評価の実行を完了するのに約 35 秒かかります。これは、迅速な再計算や大規模なシミュレーションを必要とするシナリオでは実用的ではありません。AADC を組み込むことで、同じ計算タスクが大幅に改善されます。AADC は、既存の Python* および C++ コードから単一のモンテカルロ・パスの DAG を抽出し、インテル® AVX2/ インテル® AVX-512 とマルチスレッドを活用するカーネルを作成します。

AADC を使用すると、評価だけでなくすべての 1 次リスク感度を含め、XVA シミュレーションの実行時間が 35 秒から 1 秒へと大幅に短縮されます。比較については、xVA-QL-Original（英語）と xVA-QL-Example（英語）Jupyter* ノートブックを参照してください。

インテル® AVX-512 命令とマルチスレッドを使用すると、パフォーマンスがさらに向上し、シミュレーションの効率がさらに高まります（**図 2**）。これらの機能強化は、スピードと精度が意思決定と財務結果に直接影響する金融分野では非常に重要です。

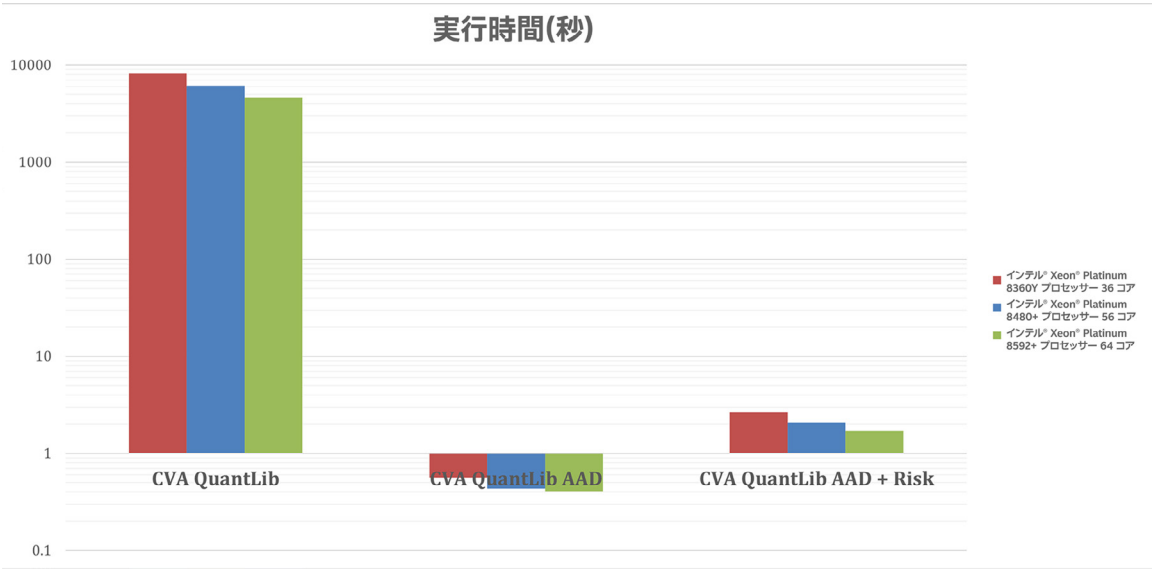


**図 2. パフォーマンスの向上**

## C++ ライブラリーのインストルメンテーション

AADC を利用する主な技術要件の 1 つは、ユーザープログラムをトレースして DAG を抽出するのに不可欠な「アクティブ」な `idouble` 型を使用するため、基礎となる C++ ライブラリーをインストルメンテーションすることです。これは通常、単純な検索と置換操作で実現できますが、多くの場合、煩わしい C++ コンパイルエラーが発生します。これらのエラーは管理可能であり、スムーズな統合と操作を確実にするには対処する必要があります。このプロセスを容易にするため、AADC には、必要な修正を自動的に行うように設計された LLVM/Clang ベースのツールが含まれています。このツールは、変更プロセスを合理化し、必要な手作業を減らし、インストルメンテーション・フェーズでエラーが発生する可能性を最小限に抑えるのに役立ちます。

## コード内の分岐の処理

AADC の記録 / 再生パターンが正しく機能するには、コード内の分岐がミスしないことが不可欠です。分岐ミスは、記録された計算に不整合をもたらし、任意の入力に対して記録された実行を適用する能力を制限します。この設定は線形問題では機能しますが、条件文を含むアルゴリズムでは困難な場合があります。

実際には、`if (x < K) return 0; else return (x-K);` などの文は、条件分岐を排除するため `max(0, x-K)` に変換する必要があります。AADC は、すべての分岐ミスを追跡し、記録に分岐ミスがない場合は一貫性があると見なされ、任意の入力に対して安全に使用できます。さらに、AADC は、ユーザーコード内で必要な変更の正確な場所を特定できるため、開発者はコードを最適化してコンパイラーとの互換性を高めることができます。

## QuantLib およびほかのライブラリーとの互換性

オープンソースの QuantLib ライブラリーを使用したデモでは、AADC の記録要件と互換性を持たせるため修正が必要でした。しかし、QuantLib 内のすべてのインストルメント・タイプが現在サポートされているわけではなく、事前の修正なしに AADC を直接適用できる範囲には限界があります。これは、引き続きより多くの分析を完全微分可能なコードと互換性のある形式に変換する必要があることを示しています。

## まとめ

ここで紹介したケーススタディーは、ベクトル化と効率的なコンパイル手法、および随伴微分法を活用することで、AADC がいかに計算金融タスクを変革できるかを明確に示しています。実行時間の劇的な短縮は、より複雑な分析を迅速に実行する能力とともに、AADC を金融機関や計量アナリストの重要な武器として位置付けています（**図 3**）。この実用的なデモは、AADC の能力を明確に示すだけでなく、金融工学やそれ以外の分野でパフォーマンス・ベンチマークを再定義する可能性を示しています。

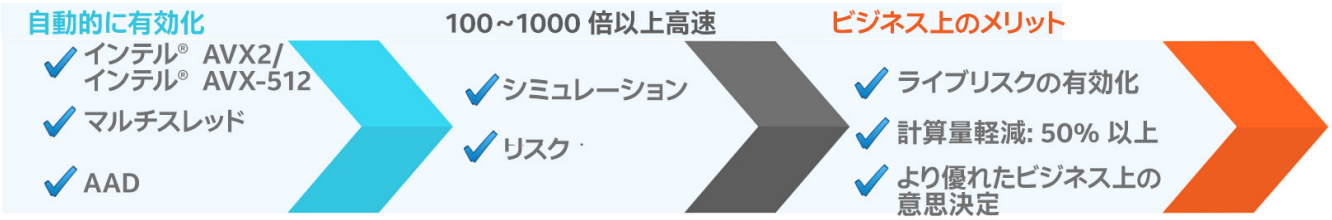


**図 3. 技術的な側面からビジネス上のメリットまで**