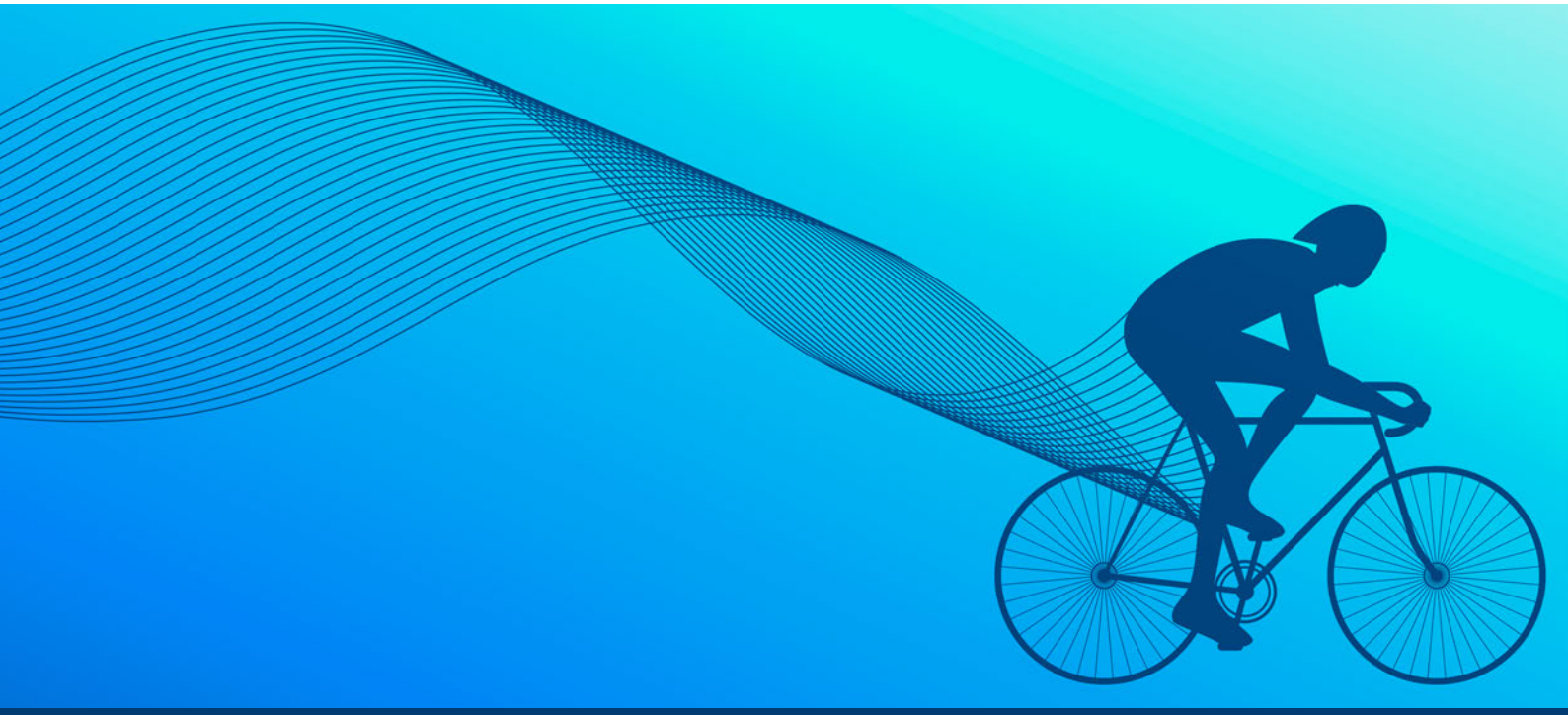

生産性と C++ パフォーマンスの向上

インテル® SIMD Data Layout Template (インテル® SDLT) でベクトル化された C++ コードの効率を高める

Udit Patidar インテル コーポレーション 開発製品部門 プロダクト・マーケティング・エンジニア

インテルでは、コードの hotspot を特定したり (インテル® VTune™ Amplifier XE など)、コードの最適化についてのアドバイスを表示する (インテル® Advisor XE など) さまざまなツールを提供しています。**インテル® C++ コンパイラー**は、特定の C++ ループの最適化が成功したかどうかを知らせる詳細な最適化レポートを生成します。[1] このたび、インテルは、インテル® Parallel Studio XE やインテル® System Studio のコンポーネントであるインテル® C++ コンパイラーに新しいテンプレート・ライブラリーを追加しました。C++ を使用しているアプリケーション開発者なら、メモリーアクセスのレイアウトを手動で最適化することがいかに大変であるかご存じでしょう。**インテル® SIMD Data Layout Template** (インテル® SDLT) ライブラリーは、プログラマーが最小の労力/オーバーヘッドで構造体配列 (AoS) 表現から配列

構造体 (SoA) 表現に変更できるようにして C++ コードを最適化します。SOA 表現は、メモリー使用量を抑え、データ構造レイアウトの変換を避けてベクトル化を容易にします。

最新のインテル® プロセッサーやコプロセッサーは、ベクトル命令に対応しており、SIMD (Single Instruction Multiple Data) プログラミング・モデルをサポートしています。インテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512) 命令では、ベクトル化されたコードの浮動小数点演算のピーク・パフォーマンスは、ベクトル化されていないコードの 8 倍 (倍精度) または 16 倍 (単精度) になります。残念ながら、特に**ベクトル化**を考慮しないで書かれたレガシーコードでは、理論性能の限界に達することはまずありません。データのメモリーレイアウトを適切に行わなければ、多くのパフォーマンスは引き出せないままです。

C++ では、データのメモリーレイアウトが、効率的なベクトル化を達成できるかどうかを左右します。構造体および配列を扱う場合は特に重要です。開発者が **`std::vector`** のように C++ 標準テンプレート・ライブラリーのコンテナーで配列を表現することは一般的です。[2]

```
struct Point3s {

  float x;
  float y;
  float z;
};

std::vector<Point3s> inputDataSet(count);
std::vector<Point3s> outputDataSet(count);

for(int i=0; i < count; ++i) {
  Point3s inputElement = inputDataSet[i];
  Point3s result = // inputElement を変換するアルゴリズムに依存しないループ反復

  // アルゴリズムは高レベルのオブジェクトとヘルパーメソッドを使用
  outputDataSet[i] = result;
}
```

そのようなデータレイアウトは C++ プログラマーにとって当たり前の表現かもしれませんが、この AOS データセットをベクトルレジスターにロードするオーバーヘッドはベクトル化のパフォーマンス・ゲインを相殺してしまいます。AOS 表現よりも SOA 表現のほうがベクトル化に適していますが、C++ プログラマーの直観には反しています。[3]

インテル® SDLT の説明に移りましょう。

インテル® SDLT ライブラリーは、ユーザーには AOS インターフェイスを提供しますが、SOA 形式でデータをメモリーに格納します。プログラマーが **SIMD** に適したデータレイアウトに変換する必要はありません。インテル® SDLT は、標準 ISO C++11 機能を使用した高レベルのインターフェイスを提供します。特別なコンパイラーのサポートは必要ありません。インテル® SDLT のレイアウトは SIMD に適しているため、OpenMP* SIMD 拡張やインテル® Cilk™ Plus SIMD 拡張のようなインテル® コンパイラーのパフォーマンス機能を活用できます。

インテル® SDLT ライブラリーを使用するには、データ型をプリミティブとして宣言します。次に、(**`std::vector`** の代わりに) ジェネリックのインテル® SDLT コンテナーでプリミティブを使用します。配列ポインターやイテレーターの代わりに、コンテナーのデータアクセサーを使用します。明示的なベクトル・プログラミングで使用すると、インテル® SDLT コンテナーのアクセサーによりデータが変換され、コンパイラーにより効率的な SIMD コードが生成されます。

```
SDLT_PRIMITIVE(Point3s, x, y, z)

sdlt::soa1d_container<Point3s> inputDataSet(count);
sdlt::soa1d_container<Point3s> outputDataSet(count);

auto inputData = inputDataSet.const_access();
auto outputData = outputDataSet.access();

#pragma forceinline recursive
#pragma omp simd
for(int i=0; i < count; ++i) {
  Point3s inputElement = inputData[i];
  Point3s result = // inputElement を変換するアルゴリズムに依存しないループ反復

  // オブジェクト・ヘルパー・メソッドを使用してアルゴリズムを高レベルに保つ
  outputData[i] = result;
}
```

アクセサーを (ループ本体内の) ローカル変数へエクスポートする (またはローカル変数からインポートする) ループのインデックスとともに使用すると、コンパイラーのベクトライザーは SIMD に適したデータ形式にアクセスし、可能な場合はユニットストライド方式のロードを実行します。多くの場合、コンパイラーはループ内部のローカル・オブジェクトのプライベート表現を SOA に最適化することができます。上記の例では、コンテナーのメモリーレイアウトがコンパイラーのローカル・オブジェクトのプライベート表現と同様に SOA であるため、コンパイラーは効率的なユニットストライド方式のロードを生成することができます。

メモリーのデータを SOA で表現すると、コンパイラーは配列要素をベクトルレジスターに直接ロードできるようになります。その結果、SIMD ベクトル化の能力を引き出すことができます。対照的に、AOS レイアウトで直接計算を行うと、実行スロットを消費するにもかかわらず、結果に寄与しない余分な命令がベクトルレジスターを占有します。

要約すると、以下の表に示すように、インテル® SDLT を使用することで、最適なメモリーレイアウトが選択されて生産性が向上し、SIMD 対応 C++ コードのパフォーマンスが向上します。

C++ を使用している開発者の生産性を向上	C++ オブジェクトの使用を中止したり、C 配列に戻す代わりに、最小限の労力で済むジェネリック・コンテナーを使用してコードを SIMD 対応にします。データレイアウトの変換はインテル® SDLT に任せます。
パフォーマンスを向上	メモリーアクセスを連続させることにより、コンパイラーがより効率的な SIMD コードを生成できるようにします。また場合によっては、オーバーヘッドが大きいとされた個所でベクトル化が利用できるようになります。
統合が簡単	インテル® SDLT のコンテナーは、`std::vector` に似たインターフェイスを提供します。データアクセサーは、既存のインテルのベクトル・プログラミング・モデルと互換性があり、ほかのインテル® パフォーマンス・ライブラリーと調和します。

インテル® SDLT が適しているかどうか評価する場合は、次の言葉を考慮すると良いでしょう:[4] 「ほとんどの場合、インテル® SIMD 対応のハードウェアではベクトル化しないよりもベクトル化するほうがよい」。

データレーン数が増加した最新のインテル® コプロセッサーでは、インテル® SDLT を考慮に入れないことによりコードの現代化が妨げられ、従来の C++ アプリケーションの欠点や短所が強調される可能性があります。C++ アプリケーションで SOA レイアウトを可能にする、新しいインテル® SDLT を試してみてください。SIMD の効率が向上し、思いがけない喜びが得られるでしょう。

インテル® SDLT が DreamWorks Animation の最先端の技術に貢献

AOS/SOA 問題は広く研究されています。そして、インテル® SDLT はすでに多くの業界で素晴らしい評価を得ています。例えば、DreamWorks Animation には、標準 C++ プログラミングの原理を利用して (つまり、明示的なベクトル化を考慮しないで) 記述された数学ライブラリーがあります。そのような大規模なコードのデータ構造レイアウトを手動で SOA に変換することは非常に困難です。数学ライブラリーはキャラクター・アニメーションのほぼすべてのエリアに影響するため、SIMD 対応の大きな障害となっていました。インテル® SDLT でコードを変換したところ、既存のコードを (変更なしで) コンパイルし、ループをベクトル化して、パフォーマンスを大きく向上できました。

「DreamWorks Animation では、[インテル® SDLT] を使用して社内アニメーション・ツール Premo* の変形コードをベクトル化しました。その結果達成できたパフォーマンスの向上は劇的なものでした。将来の映画に登場するキャラクターは、これまでになく高品質になることでしょう。ライブラリーは使いやすく、既存のコードベースにも簡単に統合することができました。」

DreamWorks Animation
Martin Watt 氏

サンプルコードとコンパイラーのドキュメント

1. インテル® SDLT を使用した平均化フィルターの実装 (英語)
2. インテル® SDLT のサンプルコード (英語)
3. インテル® SDLT のサンプルコード (その 2) (英語)

参考文献

1. 「ベクトル化アドバイザーのヒントの活用」、The Parallel Universe Issue 23
2. インテル® SIMD Data Layout Templates (インテル® SDLT) 入門 (英語)
3. 32 ビット インテル® アーキテクチャーのメモリー利用を最適化するデータ構造の操作方法 (英語)
4. ケーススタディー: 計算負荷の高いループにおける構造体配列と配列構造体の比較