インテル® AVX-512 で向上したベクトル化のパフォーマンス

インテル® コンパイラーでループをベクトル化してスピードアップするさまざまな例

Martyn Corden インテル コーポレーション ソフトウェア・テクニカル・コンサルティング・エンジニア

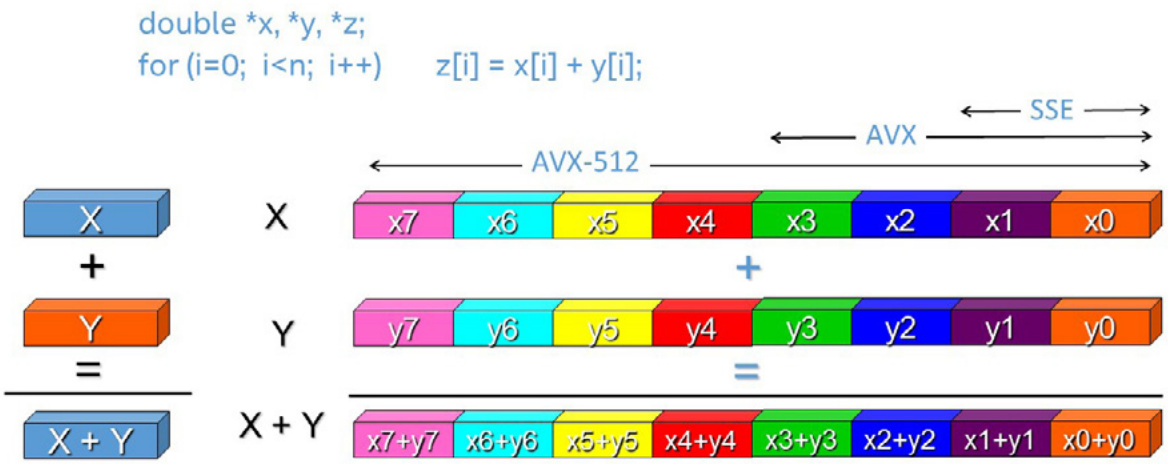
編集者から : **前号**の The Parallel Universe では、インテル® Parallel Studio XE 2017 のベクトル化サポートについて取り上げました。今号では、Martyn Corden が、インテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512) 命令についてさらに詳しく取り上げ、以前は不可能だったベクトル化を開発者が利用する方法を説明します。

クロック速度を上げるだけで簡単にパフォーマンスを向上できる時代は遠い過去のものとなりました。ムーアの法則は、代わりに、追加のコアと SIMD レジスターの帯域幅の増加による並列処理の向上に適用されます。**インテル® AVX-512** (英語) では、SIMD ベクトル幅が 512 ビットに拡張され、以前の命令セットではベクトル化できなかったループの**ベクトル化**や、より効率的なベクトル化を行うことができる新しい命令が含まれています。

この記事では、潜在的なアドレス競合の問題がある、配列を圧縮 / 展開するループや、ヒストグラムの生成とスキャッターを実行するループを、**インテル® C/C++ コンパイラーおよびインテル® Fortran コンパイラー**でベクトル化してスピードアップする方法を、例とともに説明します。さらに、構造体配列の特定のループ形式について、コンパイラーでストライドロードやギャザーをより効率的なユニットストライド方式の SIMD ロードに変換する方法を説明します。最後に、**インテル® Parallel Studio XE 2017** に含まれるインテル® コンパイラー 17.0 の最適化レポートの新しい機能を使用して、この変換を認識する方法を説明します。これらの最適化は、**インテル® Xeon Phi™ プロセッサー** x200 製品ファミリーだけでなく、インテル® AVX-512 命令セットをサポートする将来のインテル® Xeon® プロセッサーで動作するさまざまなアプリケーションにも役立ちます。

ベクトル化の利点

図 1 は、単純な倍精度浮動小数点ループを示しています。スカラーモードでは、1 つの命令で 1 つの結果が生成されます。ベクトル化を行うと、1 つのインテル® AVX-512 命令で 8 つ (インテル® AVX では 4 つ、インテル® ストリーミング SIMD 拡張命令 (インテル® SSE) では 2 つ) の結果が生成されます。インテル® Xeon Phi™ プロセッサー x200 製品ファミリーは、32 ビットおよび 64 ビットの整数と浮動小数点データを扱うさまざまな演算でインテル® AVX-512 命令をサポートします。将来のインテル® Xeon® プロセッサーでは、8 ビットおよび 16 ビット整数にも対応する予定です。



1 スカラーおよびインテル® SSE、インテル® AVX、インテル® AVX-512 でベクトル化したループ

インテル® コンパイラーは、デフォルトで自動ベクトル化を有効にしますが、インテル® AVX-512 命令をターゲットにするには、**図 2** のいずれかのコンパイラー・オプションを指定する必要があります。

Linux* および OS X*	Windows®	ターゲット
-xmic-avx512	/Qxmic-avx512	インテル® Xeon Phi™ プロセッサー x200 製品ファミリー
-xcore-avx512	/Qxcore-avx512	将来のインテル® Xeon® プロセッサー
-xcommon-avx512	/Qxcommon-avx512	両方に共通のインテル® AVX-512 のサブセット。 ファットバイナリーではありません。
-axmic-avx512	/Qaxmic-avx512	インテル® Xeon Phi™ プロセッサー x200 製品ファミリー およびインテル® Xeon® プロセッサー。 ファットバイナリーです。
-xhost	/Qxhost	コンパイルを行うホストマシンに搭載されるプロセッサー

表 1. インテル® AVX-512 命令を有効にするコンパイラー・オプション

ループの圧縮と展開

図 2A の Fortran の例は、配列を圧縮します。大きなソース配列から条件を満たす要素のみ、小さなターゲット配列にコピーされます。**図 2B** の C の例は、逆の操作 (つまり、配列の展開) を行います。小さなソース配列の要素が大きなスパース配列にコピーされます。

```
nb = 0
do ia=1, na                ! 行 23
  if (a(ia) > 0.) then
    nb = nb + 1            ! 依存関係
    b(nb) = a(ia)          ! 圧縮
  endif
enddo
```

2A 配列の圧縮

```
int j = 0
for (int i=0; i <N; i++) {
    if (a[i] > 0) {
        c[i] = a[k++];    // 展開
    }
}
// j と k の繰り返し間の依存関係
```

2B 配列の展開

密配列インデックスの条件付きインクリメントを行うと、ループ反復間の依存関係が発生します。以前は、この依存関係が自動ベクトル化を妨げていました。例えば、インテル® AVX2 向けに**図 2A** のループをコンパイルすると、次のような最適化レポートが表示されます。

```
ifort -c -xcore-avx2 -qopt-report-file=stderr -qopt-report=3 compress.f90
…
ループの開始 compress.f90(23,3)
   リマーク #15344: ループはベクトル化されませんでした: ベクトル依存関係がベクトル化を妨げています。
   リマーク #15346: ベクトル依存関係: ANTI の依存関係が nb (25:7) と nb (25:7) の間に仮定されました。
ループの終了
```

図 2B の C の例も同じようなレポートが出力されます。依存関係により不正な結果が引き起こされる可能性があるため、OpenMP* SIMD ディレクティブは使用できません。

インテル® AVX-512 では、1 つの SIMD レジスターの特定の要素を別の SIMD レジスターの連続する要素またはメモリーに書き込む新しい vcompress 命令により、この依存関係の制約を解消しました。同様に、vexpand 命令は、ソースレジスターまたはメモリーの連続する要素をディスネーション SIMD レジスターの特定の (スパース) 要素に書き込みます。これらの新しい命令により、インテル® AVX-512 が有効な場合、コンパイラーは圧縮の例 (**図 2A**) をベクトル化することができます。

```
ifort -c -xmic-avx512 -qopt-report-file=stderr -qopt-report=3 compress.f90
…
ループの開始 compress.f90(23,3)
   リマーク #15300: ループがベクトル化されました。
   リマーク #15450: マスクなし非アライン・ユニット・ストライド・ロード : 1
   リマーク #15457: マスク付き非アライン・ユニット・ストライド・ストア : 1
…
   リマーク #15497: ベクトル圧縮 : 1
ループの終了
```

3 インテル® AVX-512 を有効にした場合の配列の圧縮

ソース配列のすべての要素がロードされるため、ロードはマスクなしです。選択された要素のみストアされるため、有効なストアはマスク付きです。最適化レポートのリマーク #15497 は、圧縮表現が認識されベクトル化されたことを示しています。-S オプションを指定して取得できるアセンブリー・リストには、次のような命令が表示されます。

```
vcompressps %zmm4, -4(%rsi,%rdx,4){%k1}
```

図 2B の配列展開ループについても同様の結果が得られます。

1,000,000 の乱数を含む単精度配列を 1/2 に圧縮する操作をインテル® Xeon Phi™ プロセッサー 7250 で 1000 回繰り返した結果、インテル® AVX-512 はインテル® AVX2 よりも約 16 倍高速でした。このスピードアップは SIMD レジスターと命令の幅の違いによるものと言えます。

インテル® AVX-512 競合検出命令

間接アドレス指定のストアを含むループには、ベクトル化を妨げる潜在的な依存関係があります。次に例を示します。

```
for (i=0; i<n; i++)  a[index[i]] = …
```

2 つの異なる `i` の値で取得される **`index[i]`** の値が同じ場合、データ競合が発生し、同時に (安全に) 実行することはできません。インテル® AVX-512 の **`vpconflict`** 命令は、競合が発生しない (**`index[i]`** の値が重複しない) SIMD レーンのマスク (**`i`** の値) を提供することにより、この競合を回避します。これらのレーンで SIMD 計算が安全に実行された後、マスクアウトされたレーンでループが再実行されます。

ヒストグラム

ヒストグラムの生成は、多くのアプリケーション (画像処理など) で一般的な操作です。**図 4** のコード例は、入力値の配列 **x** について、配列 **h** で **sin(x)** のヒストグラムを生成します。2 つの入力値が同じヒストグラム・ビン **ih** に関連付けられ、依存関係が発生する可能性があるため、インテル® AVX2 では、このループはベクトル化されません。

```
for (i=0; i<n; i++) {
      y     = sinf(x[i]*twopi);
      ih    = floor((y-bot)*invbinw);
      ih    = ih > 0     ? ih : 0;
      ih    = ih < nbin-1 ? ih : nbin-1;
      h[ih] = h[ih] + 1;                        //  行 25
   }
```

4 sin(x) のヒストグラムを生成するループ

```
icc -c -xcore-avx2 histo.c -qopt-report-file=stderr -qopt-report-phase=vec
…
ループの開始 histo2.c(20,4)
      リマーク #15344: ループはベクトル化されませんでした: ベクトル依存関係がベクトル化を妨げています。 …
      リマーク #15346: ベクトル依存関係: FLOW の依存関係が h[ih] (25:7) と h[ih] (25:7) の間に仮定されました。
ループの終了
```

インテル® AVX-512 競合検出命令を使用すると、このループを安全にベクトル化できます。

```
ifort -c -xmic-avx512 histo2.f90 -qopt-report-file=stderr -qopt-report=3 -S
…
ループの開始 histo2.c(20,4)
   リマーク #15300: ループがベクトル化されました。
   リマーク #15458: マスク付きインデックス (集約) ロード : 1
   リマーク #15459: マスク付きインデックス (分散) ストア : 1
   リマーク #15499: ヒストグラム : 2
ループの終了
```

アセンブリー・コード (この記事には含まれていません) では、**x** および **index** がロードされ、すべての SIMD レーンについて **ih** が計算されます。そして、**vpconflict** 命令を実行し、ユニークなビン (つまり、**h** の要素) をレジスターに格納するマスク付きギャザーを行ってインクリメントします。**ih** の値に重複があった場合、コードはループを巻き戻して、対応するビンを再度インクリメントします。最後に、インクリメントされたビンを **h** に戻すマスク付きストア (スキャッター) を行います。

0 ～ 1 の 100,000,000 の乱数値を含む単精度配列から 200 ビンのヒストグラムの生成を 10 回繰り返す単純なテストを、インテル® AVX2 向けとインテル® AVX-512 向けにコンパイルして、インテル® Xeon Phi™ プロセッサー 7250 で実行したところ、インテル® AVX-512 はインテル® AVX2 と比べて約 9 倍のスピードアップを達成しました。この結果は、問題の詳細に大きく依存します。スピードアップのほとんどは、ギャザーやスキャッターではなく、この例の **sinf** 関数のような、ループ計算のベクトル化によるものです。競合が比較的少ない一般的なケースでは、より大きなスピードアップが得られます。しかし、特異点や狭いスパイクを含むヒストグラムのように、多くの競合がある場合、スピードアップは小さくなります。

ギャザーシャッフル最適化

小さな構造体またはショートベクトルの大きな配列はよく利用されますが、これらのベクトル化は効率的ではありません。構造体要素またはショートベクトルのベクトル化は、トリップカウントが低いため、不可能または非効率です。大きな配列インデックスのベクトル化も、連続する反復で使用されるデータがメモリーで隣接していない (つまり、連続するデータの効率的な SIMD ロードを使用できない) ため、非効率です。例えば、**図 5** の例は、3 つのベクトルを含む配列の要素の 2 乗和を計算します。

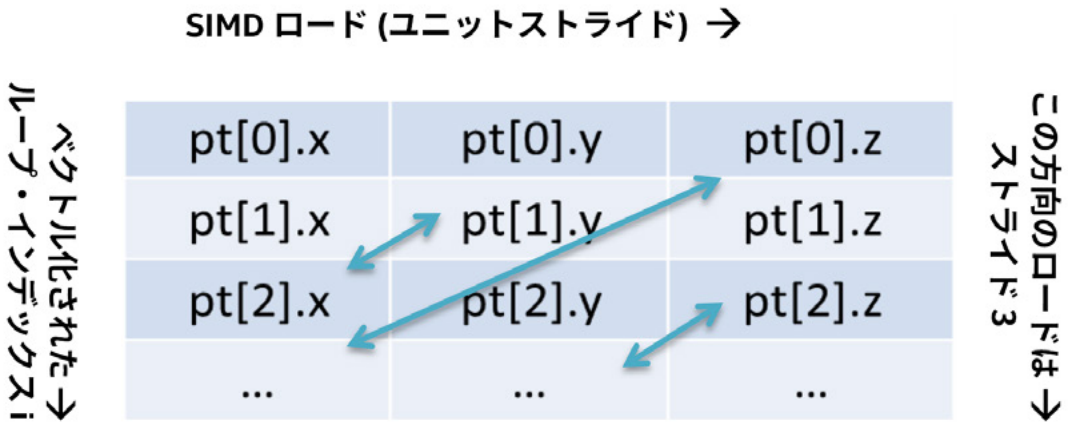
```
struct Point { float x; float y; float z; };

float sumsq( struct Point *ptvec, int n) {
    float   t_sum = 0;

    for (int i=0; i<n; i++) {
        t_sum += ptvec[i].x * ptvec[i].x;
        t_sum += ptvec[i].y * ptvec[i].y;
        t_sum += ptvec[i].z * ptvec[i].z;
    }
    return  t_sum;
}
```

5 小さな構造の大きな配列の制御

インテル® コンパイラーの旧バージョン 15 は、構造体の各要素をロードするため、ストライドロードまたはギャザーを使用してこのループをベクトル化します。

```
icc -std=c99 -xmic-avx512 -qopt-report=4 -qopt-report-file=stderr -qopt-report-phase=vec,cg
…
ループの開始 g2s.c(6,5)
   リマーク #15415: ベクトル化のサポート : 集約 (gather) が生成されました (変数 ptvec): ストライド 3
…
   リマーク #15300: ループがベクトル化されました。
   リマーク #15460: マスク付きストライドロード : 6
```

しかし、構造体の **x**、**y**、**z** 要素がメモリーで隣接していることが分かっていれば、コンパイラーはより適切な処理を行うことができます。**図 6** に示すように、コンパイラーは、コンポーネントの SIMD ロードを実行した後、置換やシャッフルを行ってデータを転置し、**i** のループがベクトル化されるように配置します。



6 ベクトル化が可能になるようにデータを転置する

インテル® コンパイラーの最新バージョン 17 でコンパイルすると、最適化レポートに「コード生成」項目が表示されます。

```
レポート： コード生成の最適化 [cg]
sumsq.c(10,22): リマーク #34030: 隣接するスパース （ストライド） ロードは速度の最適化が行われました。
詳細： ストライド { 12 }、型 { F32-V512, F32-V512, F32-V512 }、要素数 { 16 }、マスク { 0x000000007 }。
```

これは、コンパイラーがオリジナルのストライドロードを連続する SIMD ロード (およびシャッフルと並べ替え) に変換できたことを示します。10,000 の乱数に対してループを 100,000 回繰り返す単純なテストを、インテル® AVX-512 向けにコンパイルして、インテル® Xeon Phi™ プロセッサー 7250 で実行したところ、インテル® コンパイラー 17 を使用した場合の速度は、インテル® コンパイラー 15 を使用した場合の 2 倍以上でした。**pt** が 2 次元配列 **pt[10000][3]** の場合 (Fortran では、**pt** が 2 次元配列 **pt(3,10000)** または派生型配列の場合)、同じ最適化が行われます。

この「ギャザーシャッフル」最適化は、ほかの命令セット向けにも行われることがあります。しかし、インテル® AVX-512 をターゲットにする場合、強力な新しいシャッフル命令と置換命令が生成される可能性が高まります。

まとめ

インテル® AVX-512 の強力な新しい命令は、以前はベクトル化できなかったループのベクトル化や、より効率的なベクトル化により、アプリケーションのパフォーマンスを向上します。コンパイラーの最適化レポートは、これらの最適化がいつどこで行われたかを示します。

関連情報

インテル® Xeon Phi™ プロセッサー

インテル® C++ コンパイラーによるベクトル化

Fortran の明示的なベクトル・プログラミング (英語)

ベクトル化の基本 (インテル® Xeon Phi™ コプロセッサー x100 製品ファミリー向けに記述された記事ですが、ほかのプロセッサーにも当てはまります)

Fortran の配列データおよび引数とベクトル化

インテル® コンパイラー・ユーザー・フォーラム (英語)

ウェビナー (英語)

インテル® コンパイラーに導入された新しい最適化レポートを最大限に活用する

インテル® コンパイラーの新しいベクトル化機能

シリアルからその先へ : 高度なコードのベクトル化と最適化

データのアライメント、パディング、およびピール / リマインダー・ループ

BLOG HIGHLIGHTS

ダイレクト N 体シミュレーション

MIKE P. (INTEL) ›

インテル® Xeon Phi™ プロセッサーを含む、インテル® アーキテクチャーでのパフォーマンス最適化の演習

注 : この演習は、『Parallel Programming and Optimization with Intel Xeon Phi Coprocessors Second Edition (2015)』の第 4 章で説明されているさまざまな最適化の概要です。
この書籍は、**xeonphi.com/book** (英語) から入手できます。

このステップでは、サンプル・アプリケーションを利用してコードを現代化する方法を紹介します。提供されるソースコードは、N 体シミュレーション (重力的または静電的に互いに影響する粒子のシミュレーション) です。各粒子の位置と速度の追跡には、構造体 "Particle" を使用します。シミュレーションは時間ステップに離散化されます。各時間ステップで、最初に、ダイレクト All-to-all アルゴリズム (複雑性 O(n^2)) を使用して、(構造体に格納された) 各粒子の力を計算します。次に、明示的なオイラー法を使用して各粒子の速度を変更します。最後に、明示的なオイラー法を使用して粒子の位置を更新します。

この記事の続きはこちら (英語) でご覧になれます。›