サンプルコード: インテル® ディープラーニング・ブーストの新しいディープラーニング命令 bfloat16

この記事は、インテル® デベロッパー・ゾーンに公開されている「Code Sample: Intel® Deep Learning Boost New Deep Learning Instruction bfloat16 - Intrinsic Functions」の日本語参考訳です。

ファイル	ダウンロード
ライセンス	3 条項 BSD ライセンス (英語)

動作環境	
オペレーティング・システム:	Linux*
ハードウェア:	第 3 世代インテル® Xeon® スケーラブル・プロセッサー
ソフトウェア: (プログラミング言語、ツール、IDE、フレームワーク)	インテル® Parallel Studio XE 2019 に含まれる インテル® C++ コンパイラー 19.0
必要条件:	C++ の使用経験

このサンプルコードは、第 3 世代インテル® Xeon® スケーラブル・プロセッサーのインテル® ディープラーニング・ブースト (インテル® DL ブースト) を利用するインテル® 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) の使用法を示します。インテル® DL ブーストを利用するインテル® AVX-512 では、bfloat16 (Brain 浮動小数点) 形式を使用して、ディープラーニングのトレーニング・タスクのパフォーマンスを向上できる新しい命令が追加されました。

ここで紹介するサンプルコードは、組込み関数を使用して新しい命令を検証します。

インテル® DL ブースト: AVX-512_BF16 拡張

bfloat16 (BF16) は、マシンラーニング (特に、ディープラーニングのトレーニング) アルゴリズムを高速化できる新しい浮動小数点形式です。

第 3 世代インテル® Xeon® スケーラブル・プロセッサーには、BF16 形式を使用して AI 処理を高速化する、**AVX-512_BF16** と呼ばれるインテル® AVX-512 拡張 (インテル® DL ブーストの一部) が含まれます。

AVX-512_BF16 には、BF16 ペアのドット積を計算して単精度 (FP32) に集計する命令 (**VDPBF16PS**) と、パックド単精度データ (FP32) をパックド BF16 データに変換する命令 (**VCVTNE2PS2BF16**、**VCVTNEPS2BF16**) があります。

AVX-512_BF16 命令の詳細は、『インテル® アーキテクチャー命令セット拡張と将来の機能のプログラミング・リファレンス』 (英語) を参照してください。

BF16 を使用する理由

図 1 は、2 つの 16 ビット浮動小数点形式 (FP16 と BF16) と FP32 形式を比較したものです。FP16 形式には 5 ビットの指数部と 10 ビットの仮数部があり、BF16 には 8 ビットの指数部と 7 ビット の仮数部があります。FP32 と比較すると、BF16 は精度を下げているものの (7 ビットの仮数部)、FP32 と同様の範囲を保持しており、ディープラーニングのトレーニングに適していることが分かります (bfloat16 の詳細と bfloat16 がディープラーニングに適している理由については、「高精度の計算に bfloat16 人工知能データ型を利用する」(英語) を参照)。

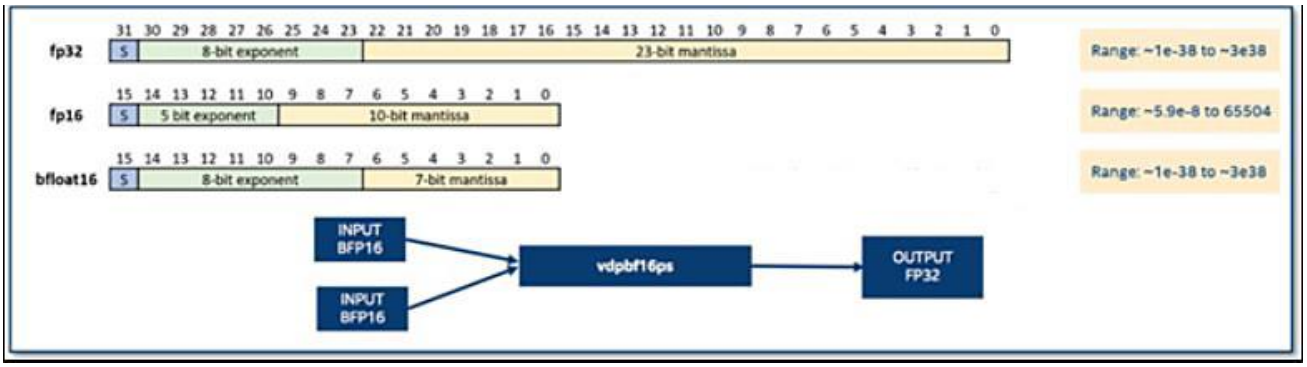

図 1. 異なる浮動小数点形式の比較。新しい **vdpbf16ps** 命令は、BF16 ペアのドット積を計算して結果を単精度 (FP32) に集計します。(画像の出典: David Mulnix)

サンプルコードの説明

このサンプルコードは、インテル® AVX-512 組込み関数を使用して BF16 命令の使用法を示します。

最初に、immintrin.h ヘッダーファイルにあるインテル® AVX-512 組込み関数のプロトタイプをインクルードします。

```
#include <immintrin.h>
```

次にメモリー変数とレジスター変数を宣言します。インテル® AVX-512 の組込み関数は、512 ビット・レジスターを表すオペランドとして C データ型を使用します。**__m512** データ型は 16 個の FP32 値を、**__m512bh** データ型は 32 個の BF16 値をそれぞれ保持できます。

```
int main() {
  float op1_f32[16];
  float op2_f32[16];
  float op3_f32[16];
  float res_f32[16];
  float res_comp_f32[16];

  // register variables
  __m512 v1_f32;
  __m512 v2_f32;
  __m512 v3_f32;
  __m512bh v1_f16;
  __m512bh v2_f16;
  __m512 vr_f32;
```

float 配列を初期化します。このサンプルコードではドット積演算を説明するため、異なる精度のオペランドの乗算結果が明確になるように、オペランドを 2.0 の平方根にしています。

```
// 配列のサンプル値を選択
   float v = sqrt(2);
   for (int i = 0; i < 16; i++)
   {
      op1_f32[i]  = v;
      op2_f32[i]  = v;
      res_f32[i]  = 1.0;
      // float32 を使用したドット積の結果 (bf16 との比較に使用)
      res_comp_f32[i] = 2.0 * op1_f32[i] * op2_f32[i] + res_f32[i];
   }
```

上記の初期化ループでは、後で BF16 命令の結果と比較するため、通常の FP32 命令でもドット積を計算しています。

メモリー内の配列が初期化されたら、**_mm512_loadu_ps** 関数を使用してレジスターにデータをロードできます (データは特定の境界でアライメントする必要はありません。データが 64 バイト境界でアライメントされている場合は、代わりに **_mm512_load_ps** 関数を使用できます)。

```
// 16 個の float32 値をレジスターにロード
// (データは特定の境界でアライメントされている必要はありません)
   v1_f32 =_mm512_loadu_ps(op1_f32);
   v2_f32 =_mm512_loadu_ps(op2_f32);
   vr_f32 =_mm512_loadu_ps(res_f32);
   (…)
```

次に、レジスターにある FP32 データを BF16 形式に変換します。前述のとおり、**VCVTNE2PS2BF16** 命令は、2 つのパックド FP32 データを (2 倍の数の値を保持する) 1 つのパックド BF16 データに変換します。次のインテル® C/C++ コンパイラーの組込み関数は、上記の命令と同等です。

```
__m512bh _mm512_cvtne2ps_pbh (__m512, __m512)
```

(詳細は、『インテル® アーキテクチャー命令セット拡張と将来の機能のプログラミング・リファレンス』(英語) を参照)。このサンプルコードでは、(結果を分かりやすくするため) 2 つのレジスターを同一の BF16 データで初期化します。

```
// 2 つの float32 レジスター (それぞれ 16 個の値を格納) を
// 1 つの BF16 レジスター #1 (32 個の値) に変換
v1_f16 = _mm512_cvtne2ps_pbh(v1_f32, v2_f32);

// 2 つの float32 レジスター (それぞれ 16 個の値を格納) を
// 1 つの BF16 レジスター #2 (32 個の値) に変換
v2_f16 = _mm512_cvtne2ps_pbh(v1_f32, v2_f32);
```

これで、**VDPBF16PS** 命令を使用して、ドット積演算を実行できます。以下は、同等のインテル® C/C++ コンパイラーの組込み関数です。

```
__m512 _mm512_dpbf16_ps(__m512, __m512bh, __m512bh)
```

上記の組込み関数では、BF16 の 2 つのレジスターのほかに、2 つの BF16 ペアのドット積演算の結果の集計に使用される **__m512** 型 (FP32 型) のパラメーターがあります。BF16 値の 2 つのペアと 1 つの FP32 値を使用するこの演算は、次の式で表すことができます。

$$VR_{32}[1] = V1_{16}[1] \times V2_{16}[1] + V1_{16}[2] \times V2_{16}[2] + V3_{32}[1]$$

このケースでは、単純にするため、FP32 入力にパックド FP32 結果を書き込んできますが、必ずしもそうする必要はありません。

```
// FMA: BF16 レジスター #1 と #2 のドット積を計算
// 結果を 1 つの float32 レジスターに累積
vr_f32 = _mm512_dpbf16_ps(vr_f32, v1_f16, v2_f16);
```

最後に、結果データをレジスターからメモリーへコピーします。

```
// レジスターの値をメモリーにコピー
// (メモリーアドレスは特定の境界にアライメントされている必要はありません)
_mm512_storeu_ps((void *) res_f32, vr_f32);
```

サンプルコードをコンパイルして実行すると、次のような結果が得られます。

```
icpc testBF16.cpp -o testBF16
./testBF16
INPUT TO BF16 INSTRUCTION:
Two float32 vectors, each containing values:
1.41421   1.41421   1.41421   1.41421   1.41421   1.41421   1.41421   1.41421
1.41421   1.41421   1.41421   1.41421   1.41421   1.41421   1.41421   1.41421

One float32 vector (input/output vector), containing values:
1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1

RESULTS OF DOT PRODUCT USING BF16 INSTRUCTION:
4.99915 4.99915 4.99915 4.99915 4.99915 4.99915 4.99915 4.99915 4.99915
4.99915 4.99915 4.99915 4.99915 4.99915 4.99915 4.99915

RESULTS OF DOT PRODUCT USING FLOAT32 INSTRUCTIONS :
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
```

FP32 命令を使用した結果と比較すると、BF16 命令を使用した結果では精度の低下が見られます。また、FP32 命令を使用した結果はそれぞれ 16 個の値を含む 2 つのレジスターのドット積演算の結果であるのに対し、BF16 命令を使用した結果はそれぞれ 32 個の値を含む 2 つのレジスターのドット積演算の結果です。

関連情報

インテル® AVX-512 組込み関数については、「インテル® 組込み関数ガイド」(英語) で詳しく説明されています。BF16 形式と命令の詳細は、ホワイトペーパー「BFLOAT16 – ハードウェア数値定義」(英語) を参照してください。

製品とパフォーマンス情報

[1] インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。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® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。

注意事項の改訂 #20110804