Art'Em – 絵画風加工を VR で実現する: パート 2

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「Art'Em – Artistic Style Transfer to Virtual Reality Week 4 Update」の日本語参考訳です。

Art'Em (英語) は、バーチャル・リアリティー (VR) で絵画風加工を利用できるようにするアプリケーションです。低精度ネットワークを利用することで、スタイライゼーションを高速化することを目的としています。

ニューラル・ネットワークとは、行列乗算とドット積にほかなりません。このプロジェクトを成功に導く鍵は、フォワード/バックワード・プロパゲーションです。パート 1 では、フォワード・プロパゲーションに含まれる多数の大きな行列乗算について説明しました。また、行列乗算の概念実証についても詳しく取り上げました。ここでは、これらを実装するための基本的なコードについて見ていきます。

コードの記述

```
#include<stdio.h>
#include<stdlib.h>
// printBits は渡された符号なし整数の 2 進表記を出力する
void printBits(size_t const size, void const * const ptr){
    unsigned char *b = (unsigned char*) ptr;
    unsigned char byte;
    int i, j;
    for (i=size-1;i>=0;i--)
        for (j=7;j>=0;j--)
        {
            byte = (b[i] >> j) & 1;
            printf("%u", byte);
        }
    puts("");    printf("\n");
}

int main() {
    // 配列の宣言と初期化
    float arra [32], arrb[32];
    for(int i = 0; i < 32; i++){
        double x = (double) rand()/RAND_MAX;
        arra[i] = ((x>0.3)?1:-1);
        arrb[i] = ((x>0.5)?-1:1);
    }
    // A と B を符号なし整数に変換
    // << は左ビットシフト演算子で符号が 001、
    // i が 2 の場合、sign<<i は 100 になる。
    // returnera は 0 に初期化され、sign<<i
    // により arra 行列を 32 ビットの符号なし
    // 整数に変換する。
    unsigned int returnera = 0, returnerb = 0, sign;
    for(int i = 0; i<32; i++){
        sign = (arra[i] >= 0);
        returnera = returnera | (sign<<i);
        sign = (arrb[i] >= 0);
        returnerb = returnerb | (sign<<i);
    }
    printBits(sizeof(returnera), &returnera);
    printBits(sizeof(returnerb), &returnerb);
```

```
    // 行列のドット積
    // 非常に重要な部分。XNOR 操作。
    unsigned int tempj = ~(returnera^returnerb);
    printBits(sizeof(tempj), &tempj);
    // pcnt として popcount 操作を実装
    // このイメージ† のように実行
    int jj = 2*(__builtin_popcount(tempj)) - 32;
    int sum = 0;
    for(int i = 0; i<32; i++)
        sum += arra[i]*arrb[i];
    printf("\nVerified sum: XNOR: %d and Normal: %d", jj, sum);
}
```

†イメージ

上記のコードは、次の結果を出力します。

```
00101011111010111111111110011110
11011100101111100010010101100011
00001000101010100010010100000010
Verified sum: XNOR: -14 and Normal: -14
```

上記から、どのように arra と arrb が符号なし整数に変換され、それぞれの XNOR 操作の後に pcnt 操作が行われたかが分かります。

これは、次元数の多い行列に拡張できます。

私は、この素晴らしい手法の適用法について調査するため、汎用行列乗算のベンチマークに 2 週間ほど費やしました。

ベンチマーク

適切に訓練すれば、XNOR-net アーキテクチャー向けに設計されたニューラル・ネットワークは、完全な精度のネットワークとほぼ同等に実行できると私は考えています。汎用ディープラーニングにおけるこのアーキテクチャーの有効性は未解明であり、さらなる調査が必要です。

ドット積と汎用行列乗算向けの XNOR を実装する前に、インテル® Xeon Phi™ プロセッサー・ベースのクラスターおよび GPU 上で Art'Em を実行する、簡単なケーススタディーを実施することにしました。これは、インテル® Xeon Phi™ プロセッサー・ベースのクラスターで XNOR-net が良い結果を達成するために重要です。

以下のケーススタディーは、完全な精度の行列の乗算をテストします。各行列のサイズは 2^n です。CUDA では、行列の次元がブロックサイズの倍数の場合に並列処理が効率良く実行されるため、行列サイズを 2^n にしました。CUDA を利用する行列乗算手法は、共有メモリーを使用します。任意のサイズの行列に対応するようにコードを変更したところ、行列乗算のパフォーマンスが大幅に低下しました。

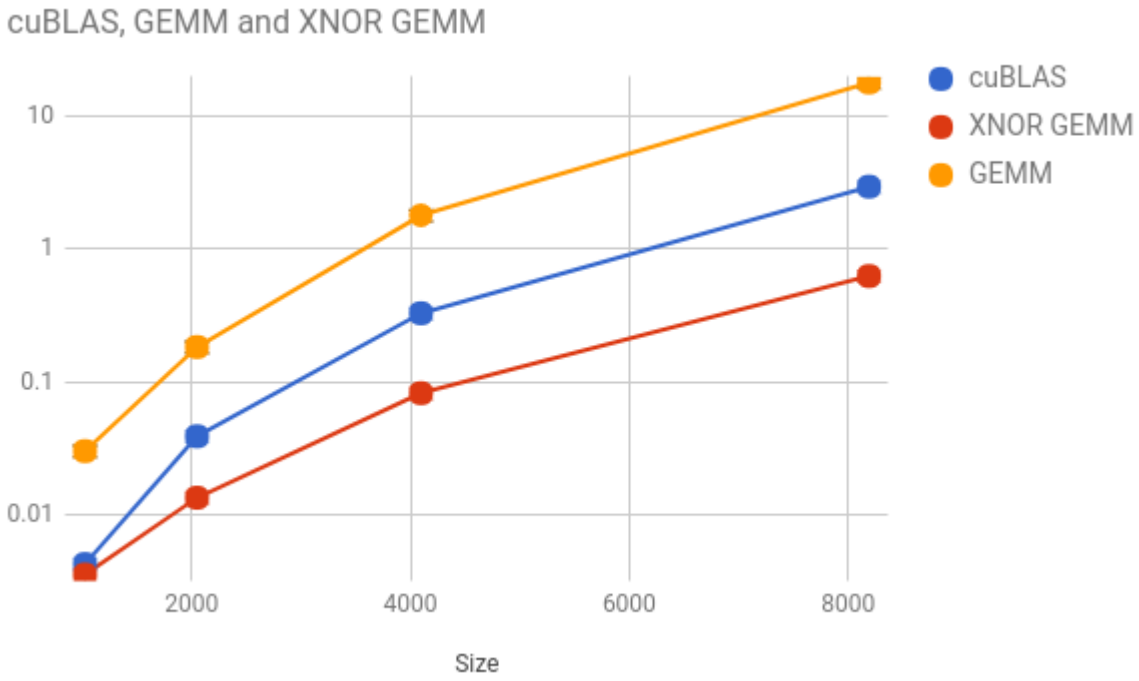
このケーススタディーにおける問題の 1 つは、私の GPU が 8192 を超えるサイズの行列乗算を実行できないことでした。より高性能な GPU では、より大きな行列を乗算できるでしょう。しかし、ケーススタディーでは 8192 以下に制限しました。

同じ行列乗算アルゴリズムをインテル® Xeon Phi™ プロセッサー・ベースのクラスターで実行しました。CPU では、インテル® マス・カーネル・ライブラリー (インテル® MKL) と OpenMP* を使用しました。また、固定行列サイズ向けの初歩的な CUDA XNOR GEMM コードのベンチマークも測定しました。インテル® Xeon Phi™ プロセッサー・ベースのクラスターと GPU の仕様は記事の最後にあります。

X 軸は両方の正方行列のサイズで、Y 軸は乗算の実行時間 (秒) です。

GPU

cuBLAS カーネル、CUDA GEMM カーネル、XNOR GEMM カーネルを使用しました。Y 軸は、**対数であることに留意してください。**



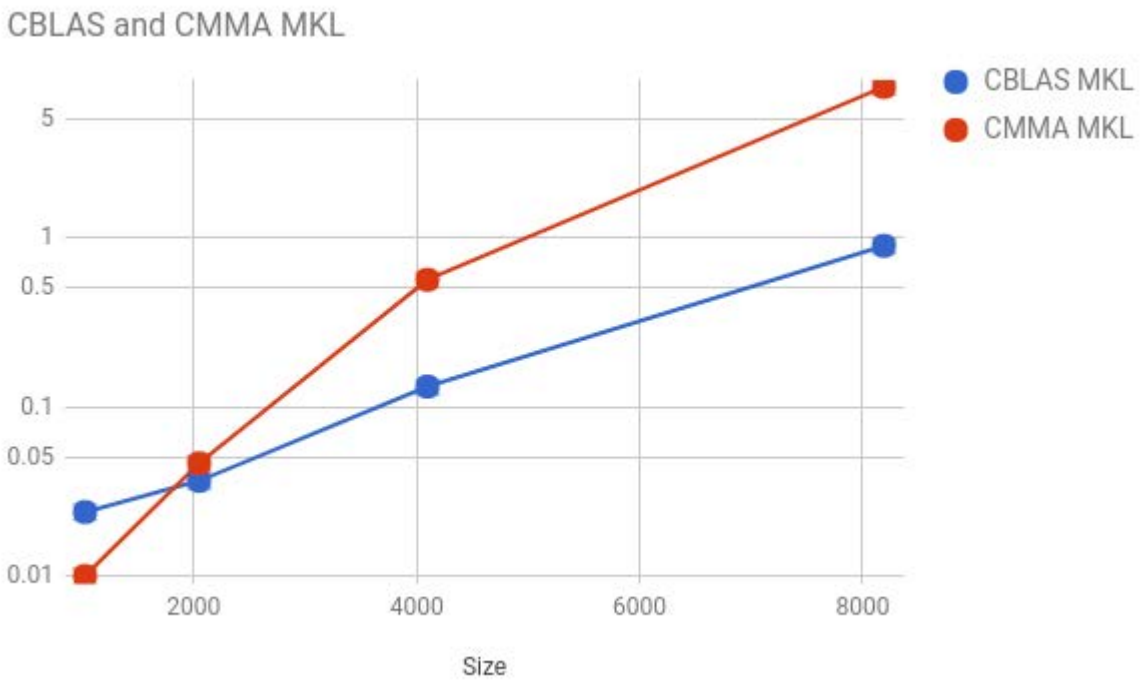
GPU 上では、XNOR GEMM カーネルが、高度に最適化された cuBLAS 関数のパフォーマンスを大きく上回りました。これは、XNOR-net が有望であることを示しています。

しかし、カスタムサイズの行列乗算に対応するようにコードを変更したところ、GEMM 関数のパフォーマンスが低下しました。これは、冗長な乗算が原因です。

XNOR アーキテクチャー向けにニューラル・ネットワークを設計すれば、これは問題にならないでしょう。このアーキテクチャーで動作するようにネットワークを訓練すれば、大幅なスループットの向上が見込めます。

CPU

インテル® MKL の最適化された 'cblas_sgemm' 関数と標準的な行列乗算関数を使用しました。標準的な行列乗算関数とその効率に関する詳細な解析は、こちら (英語) をご覧ください。Y 軸は、**対数であることに留意してください。**



XNOR GEMM コードは、汎用行列乗算にまだ対応していないため、完全な精度のネットワークの行列乗算のベンチマークを測定しました。高度に最適化されたインテル® MKL の CBLAS は、標準的な行列乗算コードのパフォーマンスを大きく上回りました。これは、驚くべきことではありません。

しかし、行列を符号なし整数にパックし、乗算をビット単位の演算に置き換えたことで、大幅なスピードアップを期待するようになりました。

プロジェクトの次のフェーズでは、任意のサイズの行列の XNOR 汎用行列乗算を実行する、高度に最適化された CUDA とインテル® MKL でサポートされる CPU および GPU 互換のコードを作成します。さらに、XNOR-net アーキテクチャーにおける畳み込み操作の最適化についても詳しく調査したいと考えています。

2^n サイズの行列でネットワークを作成することに専念したほうが良いのは分かっていますが、XNOR-nets を VGG 16、AlexNet などの既存のアーキテクチャーで利用できるようにすることも重要です。

使用した CPU:

プロセッサー: インテル® Xeon Phi™ プロセッサー 7210
コア数: 64
プロセッサー数 (CPU 数): 256
パッケージごとのコア数: 64
コアごとのスレッド数: 4
オンパッケージ・メモリー: 16GB 高帯域幅 MCDRAM (帯域幅 ~400GB/秒)
DDR4 メモリー: 96GB 6 チャネル (帯域幅 ~80GB/秒)
ISA: 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)、(ベクトル長 512 ビット)

使用した GPU:

製造元: NVIDIA*
GPU: NVIDIA* GeForce* 840M
コア速度:1029MHz
メモリー速度: 2000MHz
最大メモリー: 4096MB

前のステップ: パート 1

次のステップ: パート 3

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。