1H-04

畳み込みニューラルネットワークの行列演算における Tensor コアを用いた並列処理

Parallelization Using Tensor Core for Matrix Computation in Convolutional Neural Network

大内 佑一朗† 吉田 明正†
Yuichiro Ouchi Akimasa Yoshida

1 はじめに

物体検出のディープラーニングに用いられる畳み込みニューラルネットワーク（CNN）では多くの行列演算が行われており，学習時間を短縮するためには行列演算の高速化が不可欠である．行列演算の高速化には，GPU の活用が広く行われているが，さらなる速度向上を実現するためには，単精度浮動小数点数 FP32 の代わりに，半精度浮動小数点数 FP16 を利用して，GPU の演算性能を高めることが挙げられる．加えて，Tensor コア [1] を活用することにより，行列演算の高速化が可能になる．そこで本手法では，FP16 および Tensor コアを利用する CUDA プログラムを開発しており，CNN に用いられる行列演算に適用した． NVIDIA Quadro RTX 6000 上で行った性能評価の結果，提案手法の有効性が確認された．

2 CNN を用いた物体検出

物体検出とは，画像から物体の位置と大きさ，カテゴリを抽出するものである．近年，CNN を用いたディープラーニング技術により，精度が向上している．本研究では，物体検出のフレームワークとして，YOLOv3[2] を用いる．

2.1 YOLOv3 フレームワーク

YOLO は，2016 年に発表された物体検出のフレームワークで， C 言語で作成された機械学習用のフレームワーク Darknet[3] の一機能である．2018 年に発表された YOLOv3 では，さらにニューラルネットワークを多層化し，畳み込み層を 75 層にすることで，より高精度な検出が可能になっている．さらに，コンパイルオプションを指定することで OpenMP や CUDA を利用でき，それらを用いた並列実行が可能である．また，複数 GPU 環境での実行も可能であり，その有効性が確認されている [4][5].

2.2 YOLOv3 における行列演算

YOLOv3 のプログラムの流れは図 1 のようになっており，ループ中の forward 処理と backward 処理では全 75 層の畳み込み層により，行列積を求める計算が多く行われている．forward 処理で 75 回，backward 処理で 149 回の合計 224 回行列演算が行われており，4.1 節の環境で測定したところ，逐次実行で画像 1 枚の学習時，全学習時間のうち約 92 %が行列演算に利用されていることが確認された．これは学習データ数が増加しても大きくは変わらないため，CNN 全体の速度向上のためには，行列演算の速度を上げることが不可欠だとわかる．

†明治大学大学院先端数理科学研究科ネットワークデザイン専攻
Department of Network Design, Graduate School of Advanced Mathematical Sciences, Meiji University

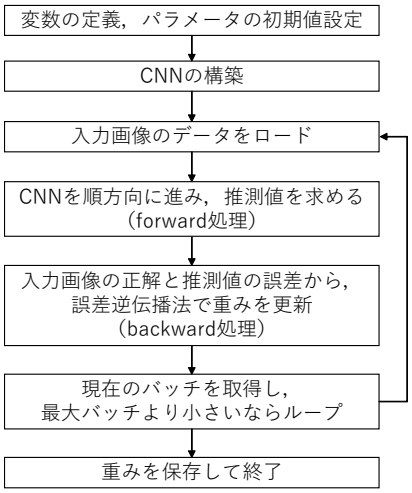

図 1 YOLOv3 のプログラムの流れ．

3 GPU における行列演算の高速化手法

本稿では行列積演算に対し，FP16 および，Tensor コアを利用した手法を提案する．

3.1 半精度浮動小数点数 FP16 の利用

半精度浮動小数点数 FP16 とは，32 ビットを利用する float 型の単精度浮動小数点数 FP32 に対して，半分の 16 ビットを利用するデータ型であり，精度を多少犠牲にする代わりに，データ量が半分になったものである．物体検出などに用いられる畳み込みニューラルネットワークでは，多くの場合，計算精度は推論結果に大きくかかわらないため，学習時間の短縮が可能である． また，データ量が半分になっているため，CPU/GPU 間の通信時間を削減できることが期待される．

3.2 Tensor コアの利用

Tensor コア [1] は Titan V 以降の NVIDIA GPU に搭載されている行列演算ユニットで，4×4 の行列 A，B，C において，AB + C を 1 サイクルで高速に計算できるものである．また A，B の行列は FP16 で入力し，出力は FP16，FP32 のどちらかを選ぶことができる．Tensor コアは，WMMA API を利用してカーネル関数を記述するか，cuBLAS などのライブラリを用いることで利用できる．

WMMA API は Tensor コアを扱うための API であり，16×16 など決められた大きさの行列を fragment と呼ばれる構造体にスレッドごとにコピーし，行列積を求めるものである．WMMA API を利用すると，ある程度自由にカーネル関数を記述することができる．

一方で，cuBLAS ライブラリは，NVIDIA GPU 向けの

Copyright ©2021 Information Processing Society of Japan.
All Rights Reserved.

行列演算ライブラリであり，いくつかの関数で Tensor コアの利用がサポートされているため，関数を呼び出すだけで Tensor コアを利用することができる．行列積演算において，Tensor コアを利用する場合は cublasGemmEx() という関数を用い，オプションで Tensor コアの利用を指定することで可能である．本研究では cuBLAS ライブラリを用いて性能評価を行う．

4 行列演算の性能評価

本性能評価では表 1 のマシンを利用し，性能評価を行った．

表 1 性能評価に用いる RTX6000 搭載サーバ．

CPU	Intel Xeon 2265（3.50GHz 12Core）
メモリ	128GB
GPU	NVIDIA Quadro RTX 6000 × 2
OS	Ubuntu18.04LTS
処理系	CUDA10.2

4.1 複数 GPU 上での YOLOv3 の性能評価

本性能評価では，YOLOv3 においてマルチ GPU 上で学習を行い，性能評価を行った．データセットは Pascal VOC を利用し，学習回数は batch=64, subdivision=16, maxbatch=2000 で行った．性能評価結果は表 2 の通りとなり，1GPU 実行の 5285.844[s] に対して，2GPU 実行では 2713.979[s] であり，1.95 倍の速度向上が得られた．

表 2 複数 GPU 上での YOLOv3 の性能評価．

GPU 数	実行時間 [ms]	1GPU 実行に対する速度向上率 [倍]
1	5285.844	1.000
2	2713.979	1.948

4.2 FP16 を用いた行列演算の性能評価

FP16 を利用した行列演算を行い，評価を行った結果，行列演算の計算時間は表 3 のようになり，256 × 256 で FP32 の 3.993[ms] に対し，FP16 は 3.732[ms] で 6.5%，4096 × 4096 で FP32 の 7932.152[ms] に対し，FP16 は 7832.649[ms] で 1.3% の高速化がそれぞれ確認された．

一方で，CPU/GPU 間の通信時間は表 4 のようになり，256 × 256 で FP32 の 0.522[ms] に対し，FP16 の 0.143[ms] で 47.8%，4096 × 4096 で FP32 の 50.039[ms] に対し，FP16 は 25.058[ms] で 49.9% の高速化がそれぞれ確認された．

4.3 Tensor コアを用いた行列演算の性能評価

本節では，行列積演算に Tensor コアを利用して，性能評価を行った．本性能評価では，1024 × 1024，2048 × 2048，4096 × 4096，8192 × 8192 の行列に対し，cuBLAS ライブラリを用いて Tensor コアの利用有無での演算時間の比較を行った．YOLOv3 で行われる行列積演算の演算回数は，一つの畳み込み層で最大約 1.7×10^9 回であり，1 枚の画像を学習する際に行う演算数の合計は約 2.1×10^{11} である．ここで，1 つの畳み込み層の演算回数は，1024 × 1024 の行列積の演算回数に相当し，1 枚の画像の学習の演算回数は，8192 × 8192 の行列積の演算回数に相当する．

表 3 FP16 を利用した行列演算の計算時間．

行列サイズ	FP16 の計算時間 [ms]	FP32 の計算時間 [ms]	FP16/FP32 比
256 × 256	3.732	3.993	0.935
4608 × 4608	7832.649	7932.152	0.987

表 4 FP16 を利用した行列演算の CPU/GPU 間の通信時間．

行列サイズ	FP16 の通信時間 [ms]	FP32 の通信時間 [ms]	FP16/FP32 比
256 × 256	0.143	0.274	0.522
4608 × 4608	25.058	50.039	0.501

性能評価結果は表 5 の通りとなり，1024 × 1024 では Tensor コア無しで 0.242[ms] だったのに対し，有りの場合 0.058[ms] で 4.17 倍，8192 × 8192 では Tensor コア無しで 84.483[ms] だったのに対し，有りの場合 10.710[ms] で 7.89 倍の高速化が得られた．

表 5 Tensor コアを利用した行列演算の性能評価．

行列サイズ	Tensor コア無計算時間 [ms]	Tensor コア有計算時間 [ms]	速度向上率 [倍]
1024 × 1024	0.242	0.058	4.172
2048 × 2048	1.454	0.289	5.031
4096 × 4096	10.752	1.739	6.183
8192 × 8192	84.483	10.710	7.888

5 おわりに

本稿では，物体検出に用いられる YOLOv3 フレームワークにおいて，FP16 を用いた高速化と Tensor コアを用いた高速化手法を提案した．

性能評価では，YOLOv3 の 2GPU による実行により，1.95 倍の速度向上が得られた．また，部分評価として行列演算を取り上げ，FP16 による性能評価により 256 × 256 行列で 6.5% の計算時間短縮，49.9% の通信時間短縮が達成された．また，Tensor コアの利用により，8192 × 8192 の行列演算では 7.89 倍の高速が達成された

そのため，YOLOv3 などの CNN における行列演算に本手法を適用することで，学習の高速化を達成できると思われる．

参考文献

[1] Akira Naruse. VOLTA AND TURING: ARCHITECTURE AND PERFORMANCE OPTIMIZATION, https://www.nvidia.com/content/apac/gtc/ja/pdf/2018/2051.pdf, 2018.

[2] Joseph Redmon, Ali Farhadi. YOLOv3: An Incremental Improvement, arXiv:1804.02767, 2018.

[3] Joseph Redmon. Darknet: Open Source Neural Networks in C, http://pjreddie.com/darknet/, 2013-2016.

[4] 西川 由理，佐藤 仁，小澤 順．Darknet53 を用いた ImageNet による一般物体認識と YOLOv3 を用いた物体検出の分散深層学習，研究報告ハイパフォーマンスコンピューティング（HPC），Vol.2019-HPC-168，No.24，pp.1-6，2018.

[5] 西川 由理，佐藤 仁，小澤 順．一般物体検出 YOLO の分散深層学習による性能評価，研究報告ハイパフォーマンスコンピューティング（HPC），Vol.2018-HPC-166，No.12，pp.1-6，2018.

Copyright ©2021 Information Processing Society of Japan.
All Rights Reserved.