2A-02

組込み向け GPU を用いた畳み込み演算の高速化に関する検討

立見　駿介†　　山本　亮†　　岡田　尚也†　　小川　吉大†

三菱電機株式会社　情報技術総合研究所†

1. はじめに

深層学習のモデルの 1 つである Convolutional Neural Network (CNN)は、物体認識や物体検出など様々な応用で有望な手法であり、組込みにおいて需要がある。CNN は、畳み込み演算の演算量が非常に大きく、高速化のために GPU 利用が進んでおり、様々な深層学習向け OSS[1]を用いることで容易に GPU 実装を行える。一方で、OSS による実装は、汎用性重視のために特定の CNN 構造や GPU には最適でなかったり、実装先 GPU が限定されたりすることが問題であった。

そこで本稿では、OSS を使用せずに、CNN に含まれる畳み込み演算の GPU 実装を行い、チューニング方法を検討する。開発環境は、多くの組込み向け GPU が対応する OpenCL を用い、CNN 構造や GPU が変化しても対応できる実装について検討する。

2. 畳み込み演算

実装する畳み込み演算を以下に示す。Iは入力特徴マップ、Fはフィルタ(重み)、O は出力特徴マップ、bはバイアス値、W, Hは特徴マップの幅と高さ、Cは入力のチャネル数、Kはフィルタサイズ、Nは出力のチャネル数をそれぞれ表す。

$$O(w,h,n) = \sum_{m=0}^{C} \sum_{j=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} \sum_{i=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} \{I(w+i, h+j, m) * F(i,j,m,n)\} + b(n)$$

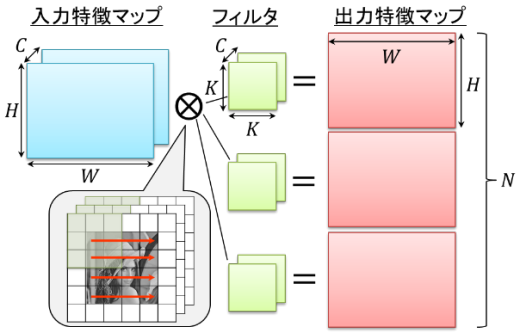

図 1　畳み込み演算のイメージ

A Study of the Acceleration of Convolution using an Embedded GPU

† Shunsuke Tatsumi, Ryo Yamamoto, Naoya Okada, Yoshihiro Ogawa, Mitsubishi Electric Corporation, Information Technology R&D Center

3. GPU 開発環境

GPU 実装には OpenCL を用いる。OpenCL は CPU・GPU・DSP 等を含むヘテロジニアス環境向けの汎用的な並列計算フレームワークであり、ベンダの異なる GPU 間で同一ソースコードを動作できるという利点を持つ。

GPU は、多数の PE を持ち、多数のスレッドを並列処理することで高速な処理を実現する。OpenCL のプログラミングモデルを図 2 に示す。ソースコードに 1 スレッドが行う処理を定義し、スレッド空間のサイズを定義することで、各スレッドは ID に応じた異なるデータを処理する。

GPU が持つ主なメモリには、容量が大きいがアクセス遅延の大きい外部 DDR メモリ、スレッド間のデータやり取りに使う共有メモリ、1 スレッドが占有するレジスタがある。一般に、外部 DDR へのアクセスはボトルネックになりやすいため、アクセスを減らすこと、アクセス効率を改善することが性能向上のポイントとなる。共有メモリとレジスタは高速であるため、データ再利用に活用することで、外部 DDR アクセスを減らし性能を向上できる。ただし、使い過ぎると、動作スレッド数が減少して性能低下を招く。そのため、処理や GPU の持つリソース量に応じて、1 スレッドの行う処理やメモリ使用量を調整することはチューニングのポイントの 1 つである。

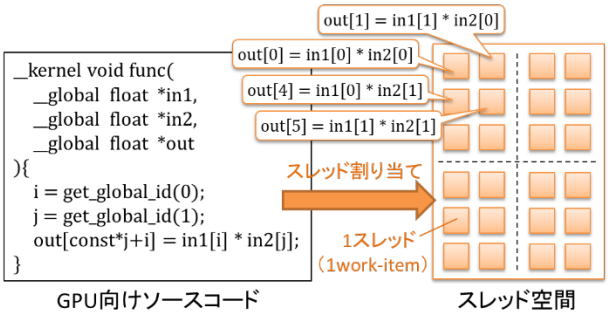

図 2　OpenCL プログラミングモデル

4. 実装方法の検討

今回、実施したチューニングを以下に示す。

(1)スレッド間データやり取りの無い処理割当て

(2)隣接ピクセルの導出を 1 スレッドにまとめる

(3)ベクトル化

(4)メモリアクセス効率を改善するデータ並び

(1)について、出力特徴マップの異なるピクセルは独立に導出できるため、各スレッドに1ピクセルの導出を処理の最小単位として割り当てる。これにより、スレッド間同期が不要となるため、高速化に繋がると考えられる。

(2)について、図 3 のように、出力特徴マップの空間方向に隣接するピクセルの導出処理では、フィルタに重複部分が生じる。そこで、本処理は1スレッドがまとめて行うとする。重複部分の入力特徴マップを再利用できるため、外部 DDR アクセスが減って高速化に繋がると考えられる。なお、T_h, T_wはまとめるピクセルの縦横幅を表す。

(3)について、一般に、複数データをベクトル化によって1データとして扱うと、メモリアクセスや演算の効率を改善できる。畳み込みでは、出力特徴マップとフィルタのチャネル方向に並ぶデータをベクトル化することで、ベクトル化するデータ数(ベクトル数)に依存せずに処理を定義でき、ベクトル化した出力ピクセルの導出処理間で同一の入力特徴マップを再利用できるため、今回はそのようにベクトル化する(図 4)。

(4)について、GPU はキャッシュを持つため、外部 DDR アクセス時は参照局所性を高めることでアクセス効率を向上できる。今回は、入力特徴マップのフィルタ重複部分を再利用するため、空間方向にデータを並べる。また、出力特徴マップとフィルタは、チャネル方向にベクトル化するため、チャネル方向にデータを並べる。詳細な並びを表1に示す。並び順は、最も優先してアクセスする順を右側に記載する。この並びの注意点として、入力と出力の並び順が異なるため、畳み込み1層ごとに並び替えが必要となる。

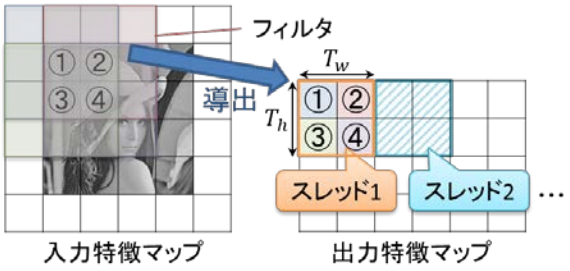

図 3　隣接ピクセル導出時のフィルタの重複

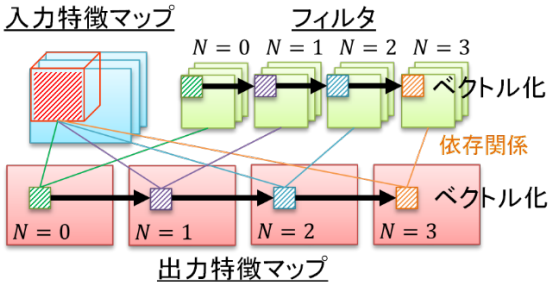

図 4　ベクトル化するデータとデータ依存関係

表 1　データの並び順

データ	並び順 (外ループ⇒内ループ)
入力特徴マップ	C⇒H⇒W
フィルタ	C⇒K⇒K⇒N
出力特徴マップ	H⇒W⇒N

5.　評価と考察

CNN の 1 つである VGG16[2]の畳み込み層とプーリング層を GPU 実装する。注意として、入力サイズは 300×300 ピクセルに変更し、プーリング層は前の畳み込み層とまとめて行うとする。GPU は、ルネサス社の車載向け SoC である R-CarH3 に搭載の PowerVR GX6650 を用いる。実装した VGG16 の構造と、実装結果を以下に示す。比較用に Caffe[1]による GPU 実装も併せて示す。

提案実装により、OSS 実装と比べて約 1.35 倍の高速化を達成できた。また、最速時の実装パラメータは層毎に異なった。そのため、これらは CNN 構造や GPU 毎に調整が必要と考えられる。今回、T_w, T_hは、全層でほぼ変わらなかったが、1 スレッドの処理量を変えるパラメータのため、GPU が変われば変化すると考えられる。

表 2　CNN 構造

#	層の種類	W	H	C	K	N	#	層の種類	W	H	C	K	N
1	conv3-64	300	300	3	3	64	10	maxpool(2x2)					
2	conv3-64	300	300	64	3	64	11	conv3-512	38	38	256	3	512
3	maxpool(2x2)						12	conv3-512	38	38	512	3	512
4	conv3-128	150	150	64	3	128	13	conv3-512	38	38	512	3	512
5	conv3-128	150	150	128	3	128	14	maxpool(2x2)					
6	maxpool(2x2)						15	conv3-512	19	19	512	3	512
7	conv3-256	75	75	128	3	256	16	conv3-512	19	19	512	3	512
8	conv3-256	75	75	256	3	256	17	conv3-512	19	19	512	3	512
9	conv3-256	75	75	256	3	256	18	maxpool(2x2)					

表 3　実装結果

#			1	2,3	4	5,6	7	8	9,10
提案実装	処理時間 [ms]	畳み込み	8.3	131.9	86.8	159.0	91.5	181.8	170.6
		並べ替え	9.4	11.9	6.8	9.5	8.0	9.6	3.5
		合計	17.7	143.8	93.6	168.5	99.5	191.3	174.1
	実装パラメータ	ベクトル数	8	8	8	8	16	16	8
		Tw	2	2	2	4	2	2	4
		Th	2	2	2	2	2	2	2
#			11	12	13,14	15	16	17,18	合計
提案実装	処理時間 [ms]	畳み込み	94.4	187.1	185.7	57.4	57.6	55.4	
		並べ替え	4.4	4.6	3.3	1.0	1.0	–	
		合計	98.8	191.7	189.0	58.4	58.6	55.4	1,540.3
	実装パラメータ	ベクトル数	16	16	16	16	16	16	–
		Tw	2	2	2	2	2	2	–
		Th	2	2	2	2	2	2	–
Caffe実装の処理時間 [ms]									2,083.4

6.　まとめ

本稿では、CNN の畳み込み演算の GPU 向け実装方法を検討した。今後は、CNN の畳み込み層以外の層の実装方法の検討と、異なる GPU に実装した際の性能評価を実施する予定である。

参考文献

[1] OpenCL Caffe, https://github.com/BVLC/caffe/tree/ opencl
[2] K. Simonyan, et al., “Very Deep Convolutional Networks for Large-Scale Image Recognition,” ICLR, 2015.