Научная конференция «Суперкомпьютерные дни в России», 24 сентября 2024 г

Study of OpenCL-based Neural Network Convolutions on GPUs

Вахрушев В.Ю., магистр, Попова Н.Н.,доцент,
кафедра СКИ ВМК МГУ

Проблема эффективной реализации сверточных слоев для нейронных сетей

- Сверточные нейронные сети используются для решения широкого спектра задач - например, обработка изображений и сигналов.
- Стандартным решением является использование библиотеки cuDNN для NVIDIA видеокарт.
- Многие компании, такие как AMD и HUAWEI, разрабатывают собственные архитектуры GPU.
- Для non-NVIDIA видеокарт нет единой аппаратно-переносимой реализации сверточных слоев.

Цель работы

- **Цель:** реализация аппаратно-переносимых сверточных слоев для нейронных сетей; выявление наиболее эффективных методов свертки в зависимости от параметров слоя.
- **Задача:** разработка сверточных слоев с использованием технологии OpenCL; исследование влияния гиперпараметров алгоритмов на их эффективность.
- **Мотивация:** Наличие аппаратно-независимой реализации сверточных слоев позволит использовать эти методы на разных архитектурах GPU. Также сравнение реализованных методов друг с другом позволит выбрать наилучший алгоритм свертки в зависимости от входных параметров.

Используемые термины

- Warp - набор GPU нитей, выполняющих одинаковые инструкции (имеющих единый регистр команд).
- GEMM - метод выполнения свертки, основанный на сведении операции свертки к операции перемножения матриц (матрицы преобразованных входных объектов и матрицы фильтров).
- GEMM Implicit - модификация метода GEMM, в котором матрица преобразованных входных объектов не хранится в памяти, а конструируется “на лету” в ходе блочного перемножения матриц.
- Winograd - метод выполнения свертки, основанный на сведении операции свертки ко множеству операций перемножения матриц алгоритмом Винограда путем разбиения входных объектов на малые блоки определенного размера. Запись Winograd (A X A, B X B) обозначает метод Винограда с размером блока A X A и размером фильтра свертки B X B.

Существующие решения

- Многие современные фреймворки глубокого обучения, таких как Caffe, Keras и PyTorch, поддерживают OpenCL реализацию.
- В Keras реализована поддержка методов вычисления свертки GEMM и Winograd.
- В PyTorch поддержка OpenCL осуществляется с использованием библиотеки DLPrimitives, реализующей методы GEMM, GEMM Implicit и Winograd.
- Также существуют независимые от фреймворков реализации сверточных операций с использованием платформы OpenCL - например, CLBlast, поддерживающая реализации методов GEMM и GEMM Implicit.

GEMM

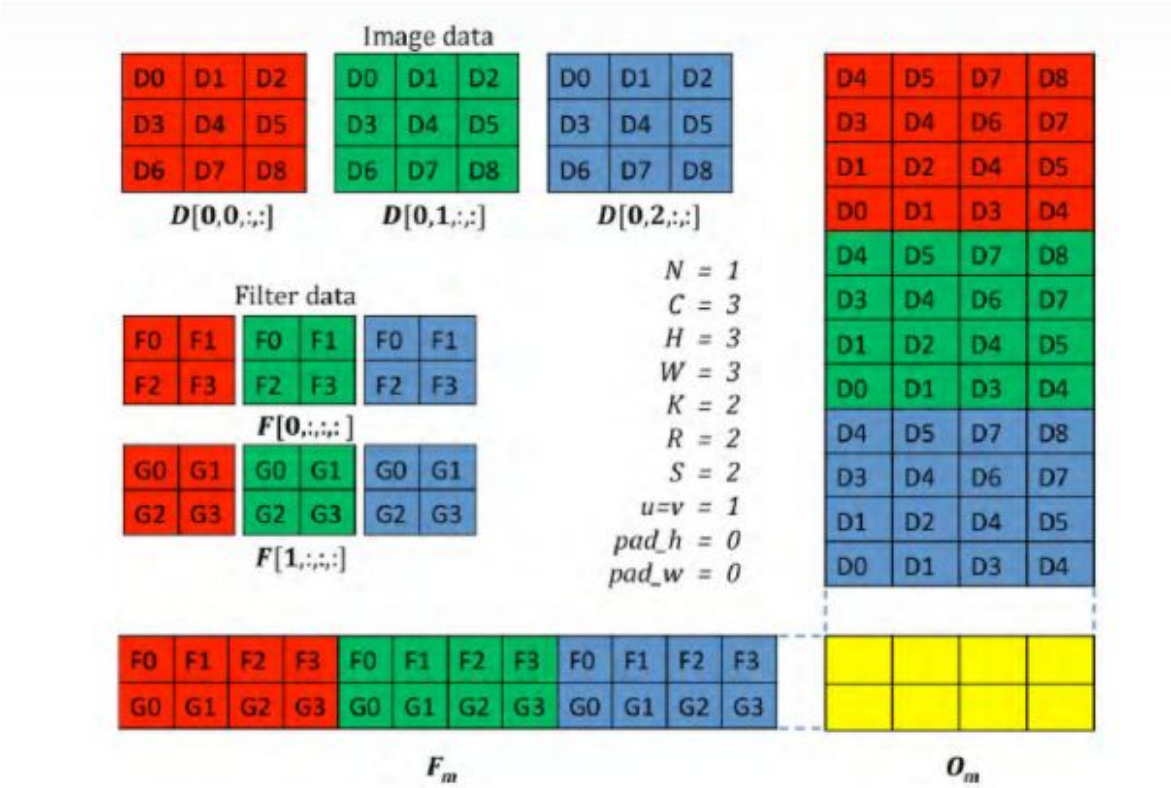
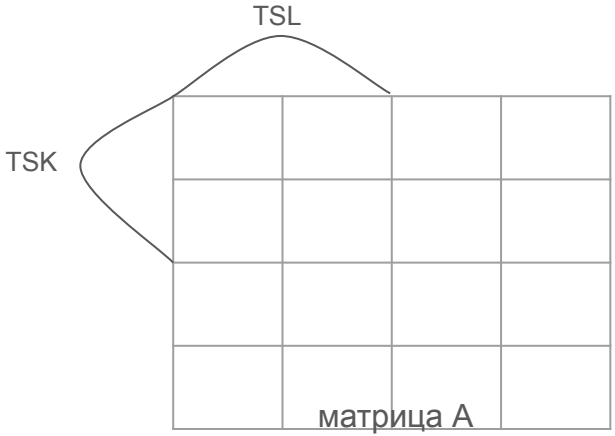
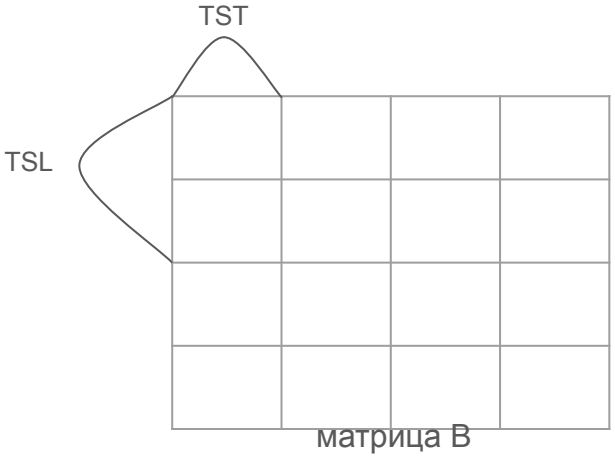


Image data
D0 D1 D2
D3 D4 D5
D6 D7 D8
D[0,0,:,:]
D[0,1,:,:]
D[0,2,:,:]
Filter data
F0 F1
F2 F3
F[0,:,:,:]
G0 G1
G2 G3
F[1,:,:,:]
N = 1
C = 3
H = 3
W = 3
K = 2
R = 2
S = 2
u=v = 1
pad_h = 0
pad_w = 0
D4 D5 D7 D8
D3 D4 D6 D7
D1 D2 D4 D5
D0 D1 D3 D4
F0 F1 F2 F3 F0 F1 F2 F3 F0 F1 F2 F3
G0 G1 G2 G3 G0 G1 G2 G3 G0 G1 G2 G3
F_m
O_m

Гиперпараметры метода GEMM



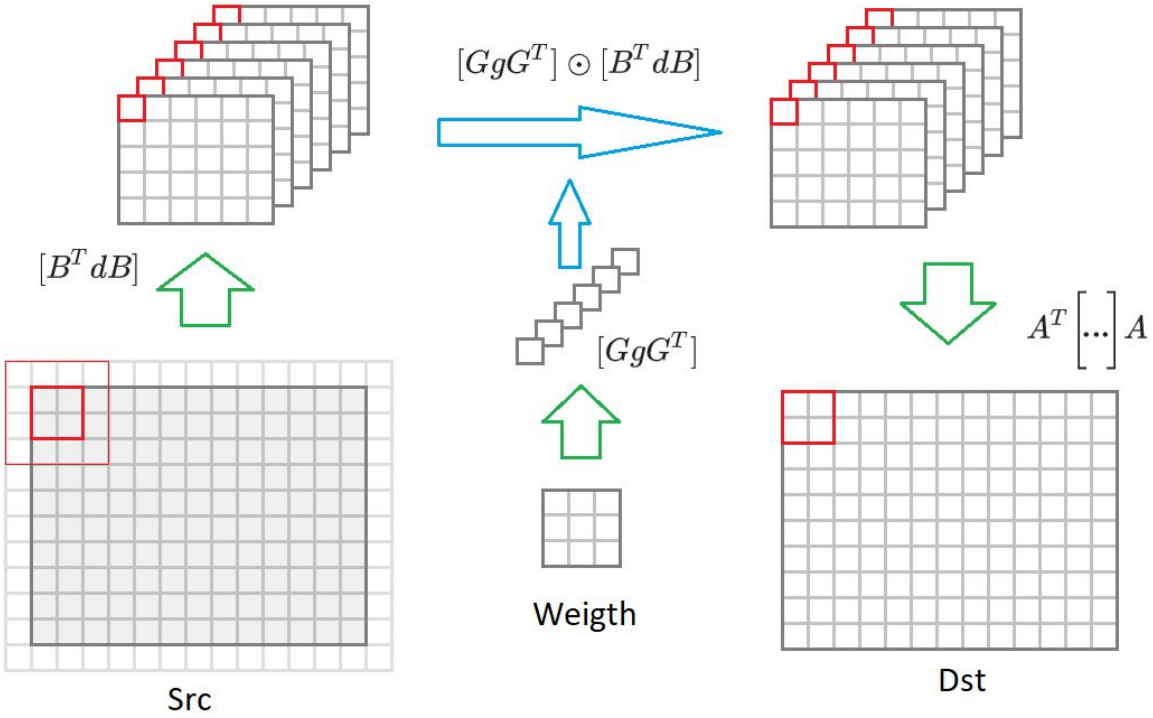


На данном слайде представлены основные гиперпараметры блочного алгоритма GEMM на GPU (матрица A размерности (K, L) умножается на матрицу B размерности (L, T)). Параметр **TSK** отвечает за то, сколько строк результата будет посчитано одной рабочей группой нитей, **TST** - сколько столбцов, **TSL** - сколько элементов будет загружено в локальную память рабочей группы за одну итерацию блочного метода. В данном примере **TSK** = 2, **TST** = 1, **TSL** = 2.

Особенности реализации GEMM

- Настройка параметра размера группы. В данной реализации он равен *64*, что кратно размеру warp.
- Настройка гиперпараметров GEMM. Параметры **TST** и **TSK** равны *64*, таким образом, рабочая группа обрабатывает выходной блок размером *64 X 64,* что позволяет равномерно распределить вычислительную нагрузку между нитями.
- Также при таком размере блока и рабочей группы каждая группа будет считывать *64 X **TSL*** элементов, где параметр **TSL** можно явно варьировать, а одна нить - *64 X **TSL** / 64 = **TSL*** параметров. По результатам экспериментов лучшие результаты показало значение **TSL** = *7*.
- Для метода GEMM Implicit вышеперечисленные пункты также справедливы.

Winograd



Особенности реализации Winograd

- Преобразование (домножение на соответствующие матрицы) фильтров заранее, в отдельном ядре.
- Наличие отдельных реализаций для методов Винограда с параметрами (2 X 2, 3 X 3), (4 X 4, 3 X 3), (2 X 2, 5 X 5).
- Настройка параметра размера группы. Рабочая группа имеет размер **tilesPerGroup** X **channelsPerGroup**, где **tilesPerGroup** - число блоков входных данных и число фильтров, обрабатываемых одной рабочей группой, а **channelsPerGroup** каналов, обрабатываемых за одну итерацию вычисления свертки методом Винограда.
- Оптимизация наиболее затратной части алгоритма - вычисления поэлементного произведения между блоками входного объекта и фильтров.

Оптимальные параметры для алгоритма Winograd

- Для метода Winograd(2 X 2, 3 X 3) лучше всего себя показали значения параметров **tilesPerGroup** = *32*, **channelsPerGroup** = *8*, учитывая что **transformedTileSize** = *16*.
- Для метода Winograd(4 X 4, 3 X 3) **transformedTileSize** = *36*, что существенно (в *36 / 16 = 2.25* раз) увеличивает размеры локального GPU буфера, необходимого для хранения данных, поэтому число блоков пришлось уменьшить, и оптимальные параметры оказались равны **tilesPerGroup** = *16*, **channelsPerGroup** = *9*.
- Для метода Winograd(2 X 2, 5 X 5) **transformedTileSize** также равен *36*, поэтому оптимальные значения параметров алгоритма для него оказались такими же.

Использование нескольких GPU

Реализованные методы поддерживают распараллеливание на несколько GPU. Способ распараллеливания зависит от входных данных:

- В случае, когда обрабатывается только один входной объект, распределение данных между GPU происходит по фильтрам. Фильтры помещаются в разделяемый между GPU буфер, каждый GPU вычисляет свертку с выделенными ему фильтрами и результат также помещается в разделяемый буфер.
- В случае обработки набора входных объектов применяется параллелизм по объектам. Каждый GPU обрабатывает объекты с выделенными ему номерами, и результат вычисления свертки записывается в разделяемый буфер.

Реализация разработанных методов – библиотека convCL

Предложенные методы были разработаны с использованием технологии OpenCL и языка Python 3.8 в рамках библиотеки ConvCL. Данная библиотека представляет собой набор классов, каждый из которых реализует один из предложенных методов.

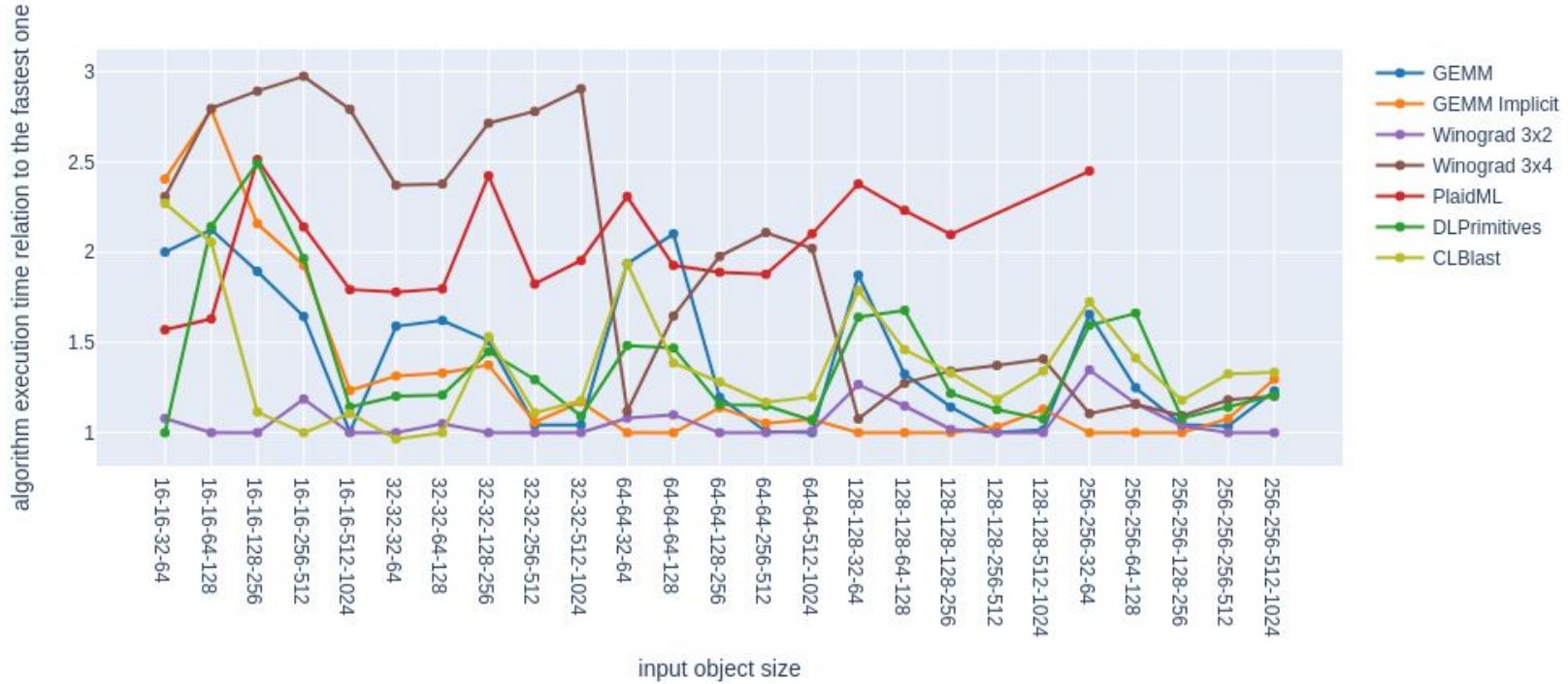
При создании экземпляра класса пользователь может указать различные параметры свертки, а также GPU контекст, в рамках которого будут выполняться методы.

Экспериментальное исследование разработанных методов

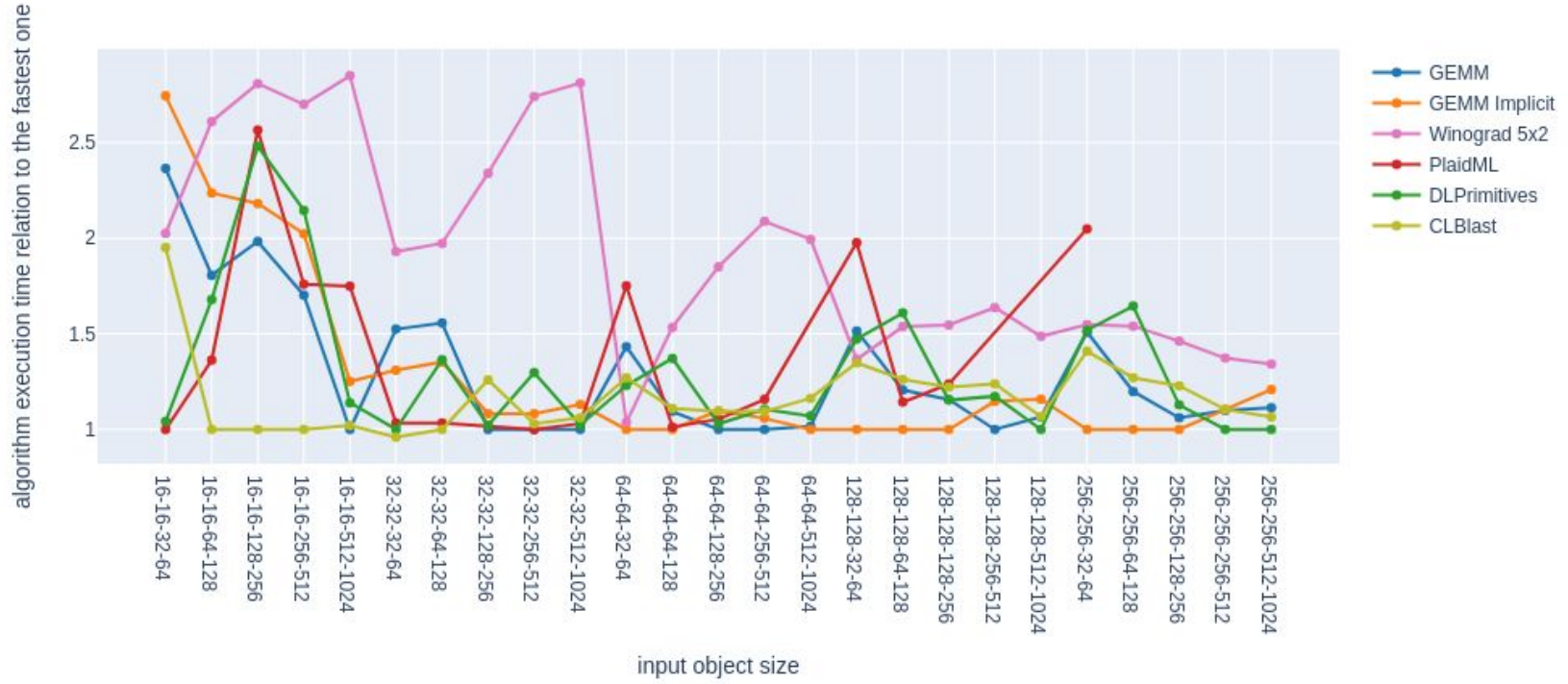
Экспериментальное исследование разработанных методов проведено с использованием случайно сгенерированных многомерных массивов определенных размеров, имеющих тип float32. Методы сравнивались друг с другом, а также с библиотеками DLPrimitives, PlaidML и CLBlast в случае экспериментов с одним GPU.

В качестве платформы для экспериментов использовался суперкомпьютер Ломоносов-2. Эксперимент проводился на одном узле(CPU - Intel Haswell, .6 GHz, 14 cores, . GPU - Nvidia P100, 16 GB). Все расчеты проводились 10 раз, а затем результаты усреднялись.

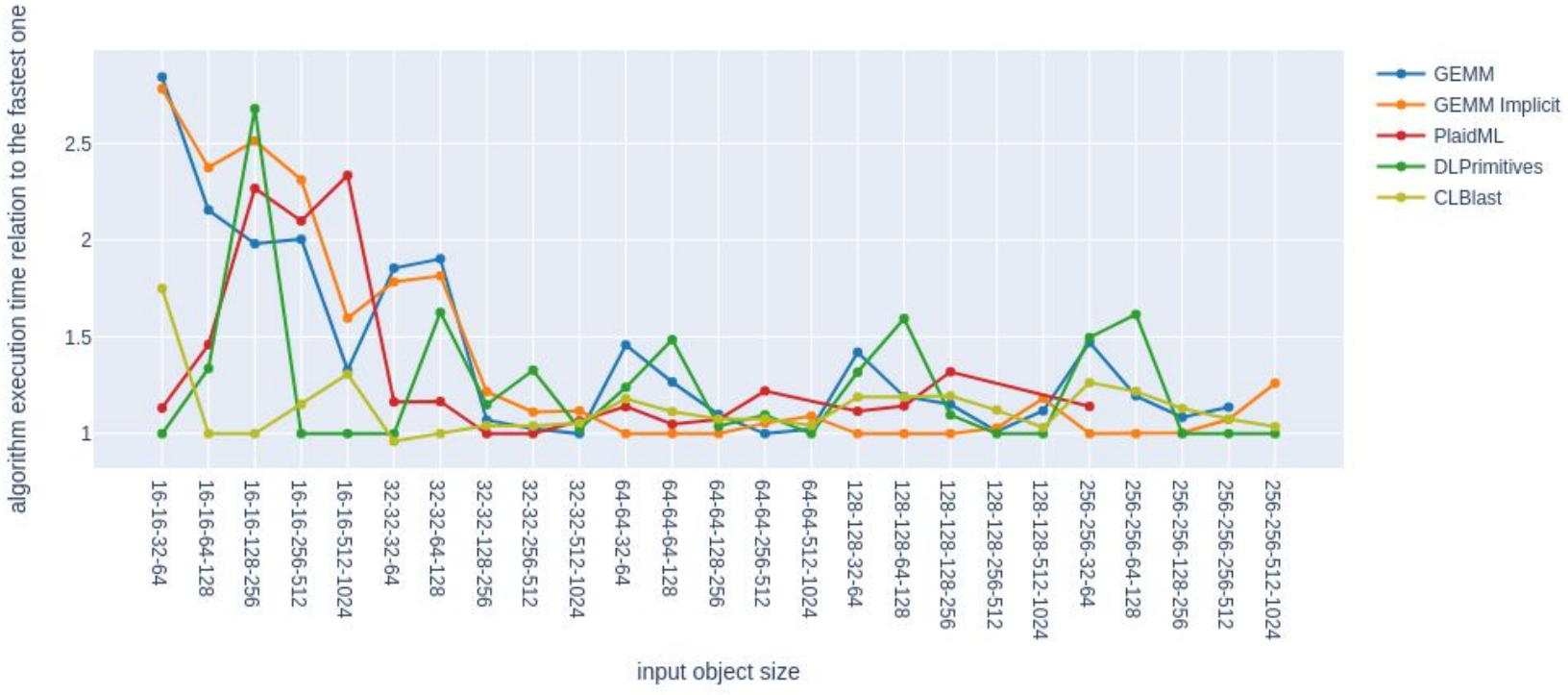
Сравнение эффективности реализованных методов; размер ядра свертки 3X3



Сравнение эффективности реализованных методов; размер ядра свертки 5X5



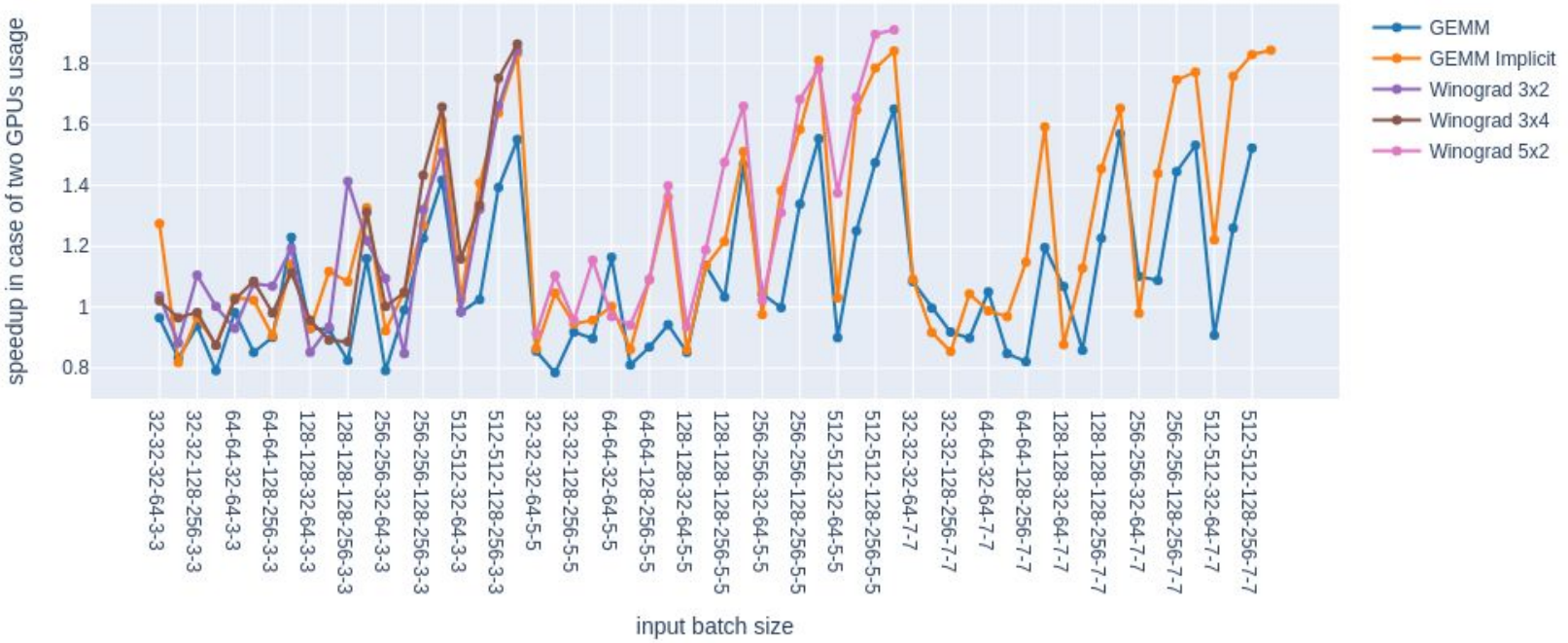
Сравнение эффективности реализованных методов; размер ядра свертки 7X7



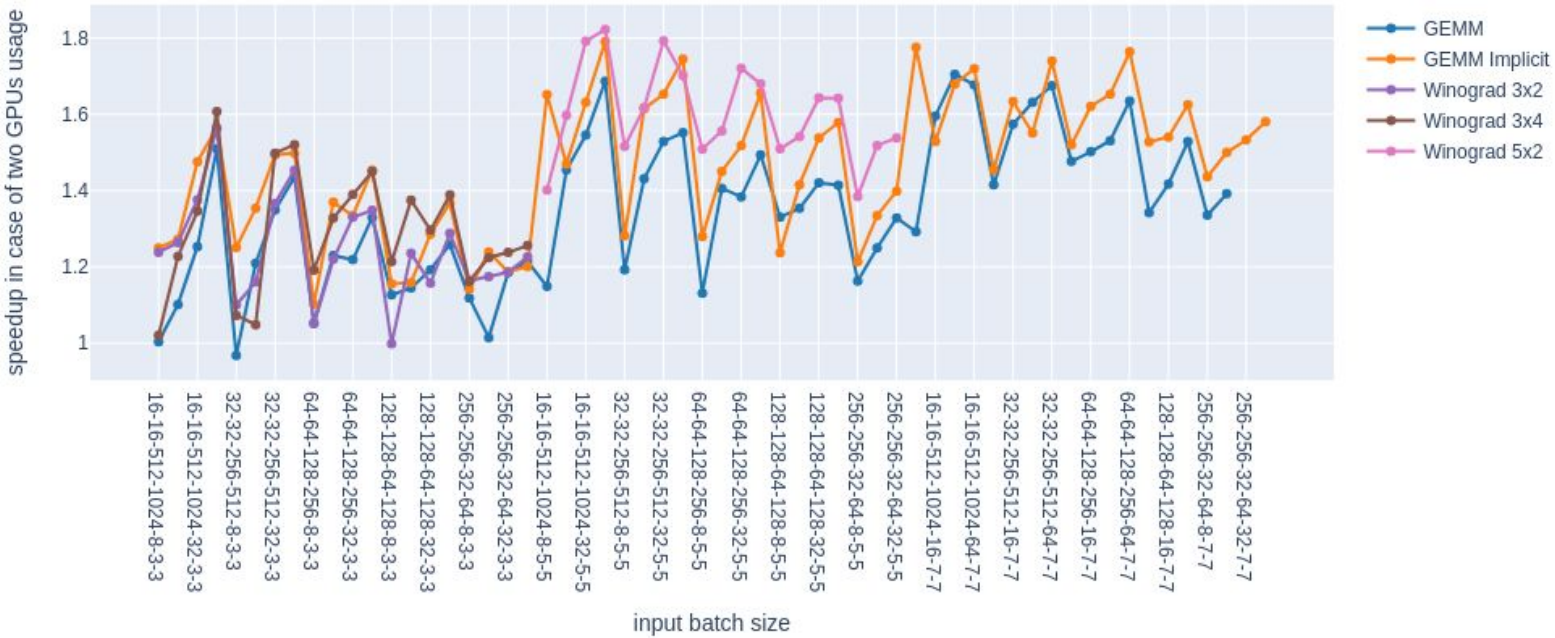
Выводы

- При размере ядра свертки 3 X 3 лучше всего себя показывают алгоритмы GEMM Implicit и Winograd(2 X 2, 3 X 3). Winograd опережает GEMM Implicit в случаях, когда размер входного изображения мал или число фильтров велико.
- При размере ядра свертки 5 X 5 Winograd (2 X 2, 5 X 5) несколько отстает от методов GEMM и GEMM Implicit, что объясняется тем, что этот метод требует гораздо больше локальной памяти GPU для хранения данных, тем самым уменьшая число результирующих блоков, которое может вычислить одна группа нитей и снижая вычислительную нагрузку на эту группу.
- При всех размерах ядер алгоритм GEMM Implicit показывает результаты лучшие, чем GEMM, что объясняется отсутствием дополнительной операции преобразования входного набора объектов (операции **im2col**).

Сравнение эффективности реализованных методов при использовании двух GPU; одно изображение



Сравнение эффективности реализованных методов при использовании двух GPU; набор изображений



Выводы

- При использовании параллелизма по фильтрам при малом размере входного объекта время вычисления свертки может даже увеличиваться из-за накладных расходов на передачу данных между GPU. Однако при большом размере изображения или при большом числе каналов видно существенное ускорение при использовании двух GPU.
- При использовании параллелизма по объектам ускорение от использования двух GPU тем более заметно, чем больше объектов во входном наборе данных.
- Также можно отметить, что ускорение существенно зависит от размеров ядра свертки. Чем больше размер ядра, тем больше вычислительной работы нужно провести для вычисления свертки и тем больше ускорение при использовании двух GPU.

Заключение

Основные результаты работы:

- Разработаны и исследованы методы реализации сверточных операций для нейронных сетей. Методы реализованы в виде программной библиотеки convCL.
- Проведенные эксперименты показали эффективность предложенных методов в сравнении с другими OpenCL реализациями.
- Библиотека не привязана к конкретному фреймворку и является инструментом с открытым исходным кодом. Она может быть использована как самостоятельно, так и в совокупности с другими библиотеками для обучения нейронных сетей с использованием платформы OpenCL.

СПАСИБО ЗА ВНИМАНИЕ!

Пример преобразующих матриц для метода Winograd с параметрами (4X4, 2X2)

$$B^T = \begin{bmatrix} 4 & 0 & -5 & 0 & 1 & 0 \\ 0 & -4 & -4 & 1 & 1 & 0 \\ 0 & 4 & -4 & -1 & 1 & 0 \\ 0 & -2 & -1 & 2 & 1 & 0 \\ 0 & 2 & -1 & -2 & 1 & 0 \\ 0 & 4 & 0 & -5 & 0 & 1 \end{bmatrix}$$

$$G = \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ -\frac{1}{6} & -\frac{1}{6} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{1}{6} & -\frac{1}{6} \\ \frac{1}{24} & \frac{1}{12} & \frac{1}{6} \\ \frac{1}{24} & -\frac{1}{12} & \frac{1}{6} \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 0 \\ 0 & 1 & 1 & 4 & 4 & 0 \\ 0 & 1 & -1 & 8 & -8 & 1 \end{bmatrix}$$

Оптимальные параметры для алгоритма Winograd (full)

- Для метода Winograd(2 X 2, 3 X 3) лучше всего себя показали значения параметров **tilesPerGroup** = *32*, **channelsPerGroup** = *8*. При таких параметрах, учитывая что **transformedTileSize** = *16*, размер подгруппы равен *32 X 8 / 16* = *16*, что также позволяет эффективно посчитать произведение выделенных подгруппе нитей элементов.
- Для метода Winograd(4 X 4, 3 X 3) **transformedTileSize** = *36*, что существенно (в *36 / 16 = 2.25* раз) увеличивает размеры локального GPU буфера, необходимого для хранения данных, поэтому число блоков пришлось уменьшить, и оптимальные параметры оказались равны **transformedTileSize** = *16*, **channelsPerGroup** = *9*. Размер подгруппы нитей равен *16 X 9 / 36 = 4*.
- Для метода Winograd(2 X 2, 5 X 5) **transformedTileSize** также равен *36*, поэтому оптимальные значения параметров алгоритма для него оказались такими же.

Пример вычисления произведения

1		2		3		4		1		2		3		4	
T1,E1,C1	T2,E1,C1	T1,E2,C1	T2,E2,C1	T1,E3,C1	T2,E3,C1	T1,E4,C1	T2,E4,C1	T1,E1,C2	T2,E1,C2	T1,E2,C2	T22,E2,2	T1,E3,C2	T2,E3,C2	T1,E4,C2	T2,E4,C2
F1,E1,C1	F2,E1,C1	F1,E2,C1	F2,E2,C1	F1,E3,C1	F2,E3,C1	F1,E4,C1	F2,E4,C1	F1,E1,C2	F2,E1,C2	F1,E2,C2	F2,E2,C2	F1,E3,C2	F2,E3,C2	F1,E4,C2	F2,E4,C2

На слайде приведен пример распределения вычисления поэлементного произведения элементов блоков входных объектов и фильтров. В данном примере **tilesPerGroup** = 2, **channelsPerGroup** = 2, **transformedTileSize** = 4. Соответственно, группа нитей разбивается на 4 подгруппы каждая размером 2 X 2 / 4 = 1 нить. Подгруппа вычисляет произведение каждого выделенного ей элемента из буфера входных объектов с каждым элементом из фильтров для каждого канала.