МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №4

по дисциплине: Параллельное программирование
тема: «Параллельное программирование с использованием OpenCL»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
доц. Островский Алексей Мичеславович

Белгород 2025 г.

Цель работы: Изучить основы параллельного программирования с использованием OpenCL, реализовать вычислительные задачи с применением графического ускорителя (GPU), оценить производительность и масштабируемость решений при выполнении вычислений.

Условие индивидуального задания:

Реализовать параллельные алгоритмы для обучения и предсказания с помощью логистической регрессии. Дан набор данных dataset[N][D], где N — количество обучающих примеров, D — размерность признакового пространства (загрузка из файла). Даны метки labels[N] (выходные значения 0 или 1). Модель: логистическая регрессия с функцией активации sigmoid. Предсказание вычисляется по формуле:

$$y_{pred} = sigmoid(\sum_{i=1}^{D} w_i x_i + b)$$

где w_i — веса модели, b — смещение. Требуется: реализовать параллельный прямой проход (forward pass), где каждая параллельная нить (thread) вычисляет предсказание y_{pred} для одного обучающего примера. Реализовать параллельное вычисление локальных градиентов по каждому примеру. Выполнить редукцию локальных градиентов для обновления весов и смещения. Обеспечить эффективную работу на GPU с использованием OpenCL. После обучения вывести предсказания на обучающем наборе данных и на новом тестовом примере. Сравнить производительность (по времени выполнения) между реализациями на GPU и CPU. Все данные использовать в формате float. Загрузка данных из файла.

Ход выполнения работы

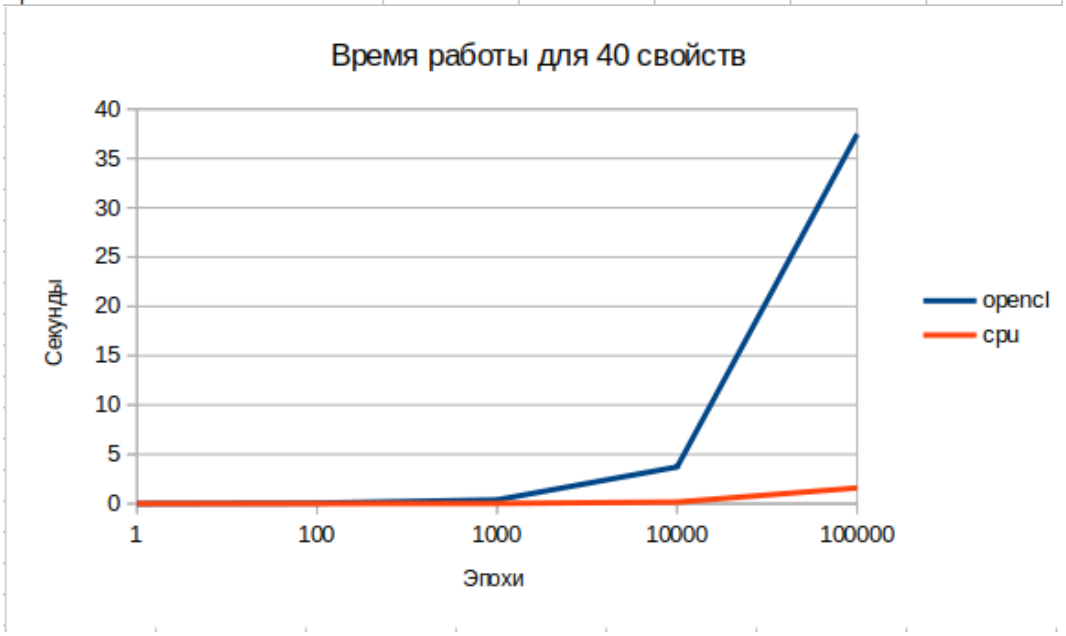
Описание архитектурных решений Шейдер для вычисления предсказаний использует разбиения по группам для просчёта предсказания для одного объекта. Размер группы представляет собой количество свойств объекта наиболее близкое к 2^N (в большую сторону). В конце используется редукция сумма для получения итогового предсказания. Шейдер может работать как со множествами объектами, так и с одним объектом.

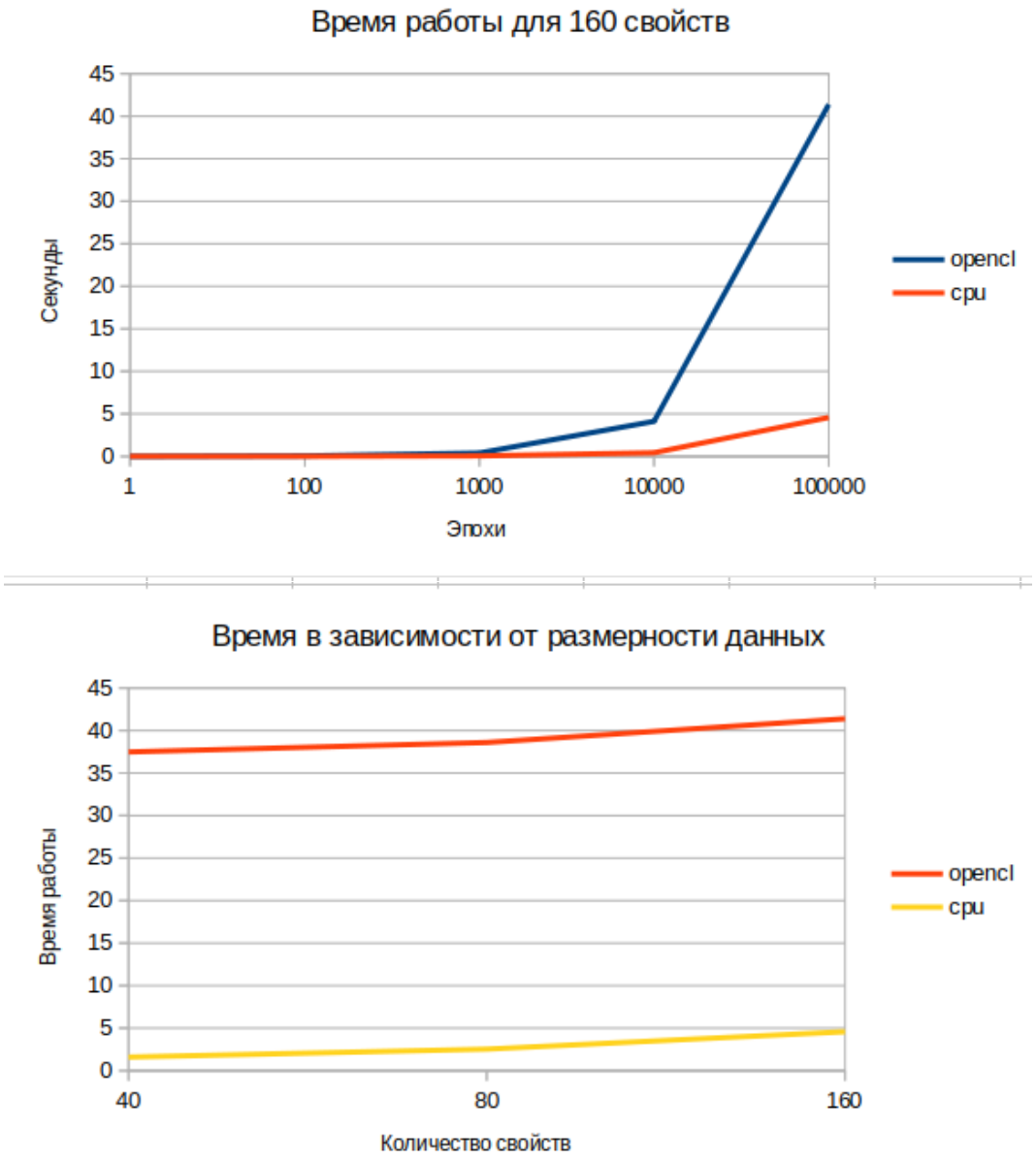
Шейдер для перерасчёта смещения и весов использует несколько другую композицию. В нём в группе объединяется один признак объекта, a local_id представляет собой номер объекта. Соответственно размер группы будет количеством объектов наиболее близкое к 2^N (в большую) сторону. В конце производится редукция градиентов и производится перерасчёт весов. А также производится редукция для смещения и производится его перерасчёт.

Для пересчёта градиентов использовался другой подход, более подходящий под задачу параллелизации. (Asynchronous SGD) Так, в примере градиент пересчитывался каждый раз с новыми данными и веса обновлялись на лету. При новом подходе градиенты считаются для всех предсказаний сразу и редуцируются. Такой подход может привести к потенциально худшим результатам, однако позволяет распараллелить задачу.

Графики ускорения и зависимостей. Ниже приведено время выполнения программы при разных количествах эпох и свойств:

	Эпохи				
Устройство	1	100	1000	10000	100000
OpenCL (AMD Radeon 6800 XT)	0,003	0,05	0,382	3,72	37,48
Процессор (Intel Core i5-12600k)	0,00003	0,002	0,02	0,16	1,59
При 40 свойствах					
	Эпохи				
Устройство	1	100	1000	10000	100000
OpenCL (AMD Radeon 6800 XT)	0,003	0,05	0,4	3,87	38,59
Процессор (Intel Core i5-12600k)	0,00005	0,002	0,036	0,253	2,53
При 80 свойствах					
	Эпохи				
Устройство	1	100	1000	10000	100000
OpenCL (AMD Radeon 6800 XT)	0,003	0,06	0,4	4,12	41,38
Процессор (Intel Core i5-12600k)	0,00007	0,004	0,058	0,425	4,58
При 160 свойствах					





Вывод: в ходе лабораторной работы изучили основы параллельного программирования с использованием OpenCL, реализовать вычислительные задачи с применением графического ускорителя (GPU), оценить производительность и масштабируемость решений при выполнении вычислений.

В данной задаче победить реализации на OpenCL процессор не удалось, однако можно заметить, что время работы при выполнении на OpenCL при увеличении количества свойств практически не растёт по сравнению с однопоточной реализацией. Можно сделать вывод, что реализация OpenCL имеет смысл, если будут использоваться многомерные данные, при маломерных данных с небольшим количеством нейронов, можно использовать и однопоточный подход.

Также для обучения нейронных моделей можно использовать специализированные устройства, которые сегодня разрабатываются компанией Nvidia в частности. Эти устройства могут отличаться повышенным количеством ядер, памяти, возможности использования более широких групп с увеличенной локальной памятью.

Не стоит на месте и развитие стандартов. OpenCL, пусть и является одним из широкоподдерживаемых стандартов, на сегодняшний день находится в стадии ”угасания”. Рекомендуется использовать более современные поддерживаемые технологии, такие как OpenGL, Vulkan. Производители устройств предлагают специализированные решения

для конкретных устройств, так для AMD существует ROCm, для Nvidia существует CUDA. Однако возникает проблема совместимости, но есть решение и для неё. Так AMD HIP предлагает компиляцию как для CUDA, так и для ROCm (C++). Можно использовать Rust - более безопасный язык программирования - для написания шейдеров, это возможно благодаря компиляции Rust в промежуточный шейдерный код SPIR-V, поддерживаемый многими технологиями при помощи пакета rust-gpu. Пакет всё ещё развивается, так в нём пока что ещё не поддерживается компиляция в OpenCL, поэтому данная лабораторная работа и была выполнена с применением традиционного шейдера.

Современные тенденции развития графических ускорителей направлены на создание универсального языка для написания программ для графических ускорителей а также на увеличения производительности для решения актуальных задач: использование нейронных сетей а также их обучение.