Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

Компиляторы для глубоких нейросетевых моделей

Общая схема построения тензорных компиляторов

При поддержке компании YADRO

Сысоев А.В.,
Кустикова В.Д.

Содержание

ОБЗОР ОБЩИХ ПОДХОДОВ К ОПТИМИЗАЦИИ

Цель оптимизации программ

- ***Цель оптимизации программ*** – получение из работающего* варианта программы другого работающего* варианта, удовлетворяющего некоторым критериям

* принципиальный момент

Рассматриваемые критерии

Основными ***критериями*** при оптимизации программ являются:

- **Скорость работы**

  Миф: компьютеры работают все быстрее. Ничего делать не нужно ☺
  // Большие задачи; Real-time; Финансы и др. области

- **Объем используемой памяти**

  Миф: много дешевой памяти и будет еще больше. Ничего делать не нужно ☺
  // Большие задачи; ограничения на ускорителях

- **Объем места, занимаемого на диске**

  Выглядит сомнительным, но вспоминаем про инженерные расчеты
  и ограничения на кластерах

Рассматриваемые критерии. Взаимодействие

Скорость (performance) vs. **Память** vs. **Место на диске**

Часто критерии ***противоречивы***:

- Используем дополнительную память – ускоряем работу
- Используем место на диске для однократного расчета и последующего хранения и использования – ускоряем работу
  (**пример**: кратчайшие пути, см. Renato Werneck, 2010)

Обычно ***один из критериев является основным!***

- В современных условиях это чаще **скорость работы**, но есть и другие случаи

Требования к оптимизации*

- ***Оптимизация ради оптимизации не только бесполезна, но и вредна!***

- **Требования**:
  - Переносимость
  - Небольшая трудоемкость
  - Значимый выигрыш
  - Понятность программы
  - Возможность развития

* По материалам *К. Касперски. Техника оптимизации программ. Эффективное использование памяти. БХВ-Петербург, 2003.*

Требования к оптимизации. Переносимость

- Избегаем ассемблерных вставок
- Избегаем недокументированных возможностей аппаратуры и средств разработки

Основная причина – совместимость (средства разработки и аппаратура развиваются)

Требования к оптимизации. Небольшая трудоемкость

- Оптимизация не должна существенно увеличивать трудоемкость разработки и тестирования!

Основная причина – необходимо уложиться в сроки и бюджет и выпустить продукт

Требования к оптимизации. Значимый выигрыш

- Оптимизация должна приносить значимый* выигрыш

Основная причина – соразмерность затрат и результата (избегаем оптимизации ради оптимизации)

* Есть области, в которых 2% это значимый выигрыш:
Пример: моделирование финансовых рынков

Требования к оптимизации. Понятность программы

- Текст программы должен оставаться понятным «нормальному» программисту

Основная причина – сопровождение

Требования к оптимизации. Возможность развития

- Необходимо соблюдать проектные решения (в целом)

Основная причина – модификации неизбежны

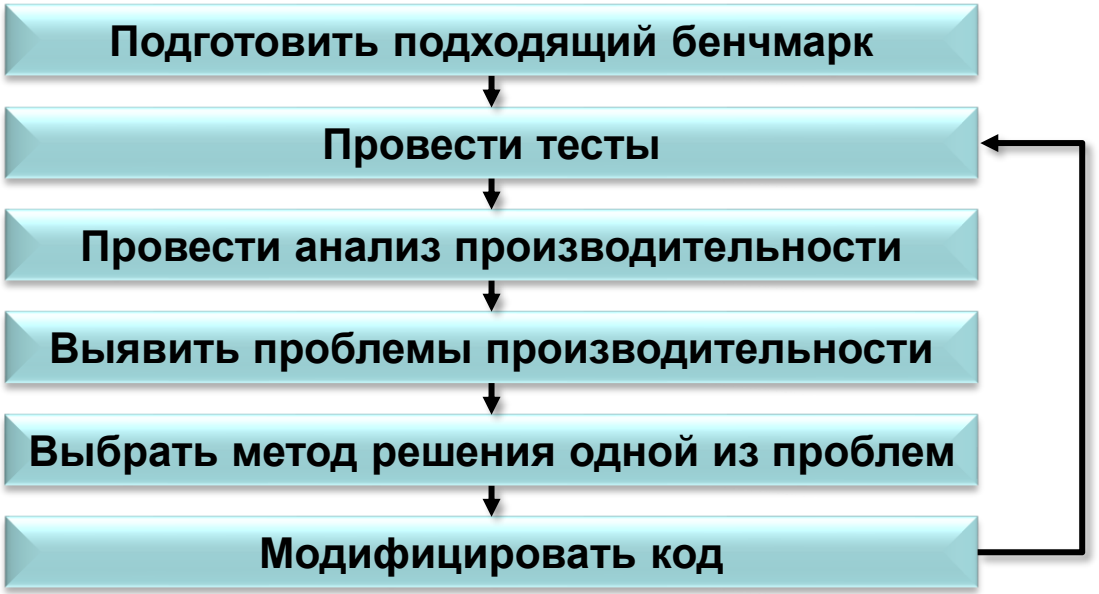
Правила оптимизации*

- Прежде чем оптимизировать программу, ***убеждаемся в ее работоспособности***
- Основной прирост производительности приходит от алгоритмической оптимизации и выбора структур данных, а не от «трюков». ***Ищем эффективные алгоритмы и подходящие СД***
- Оптимизация кода ≠ ассемблерная реализация. ***Улучшаем код для увеличения быстродействия в рамках ЯПВУ***. ***Используем возможности компилятора***
- ***Используем оптимизированные библиотеки***

* По материалам *К. Касперски. Техника оптимизации программ. Эффективное использование памяти. БХВ-Петербург, 2003.*

Жизненный цикл оптимизации

Предполагаем, что программа ***написана*** и ***работает***, но работает ***недостаточно быстро***

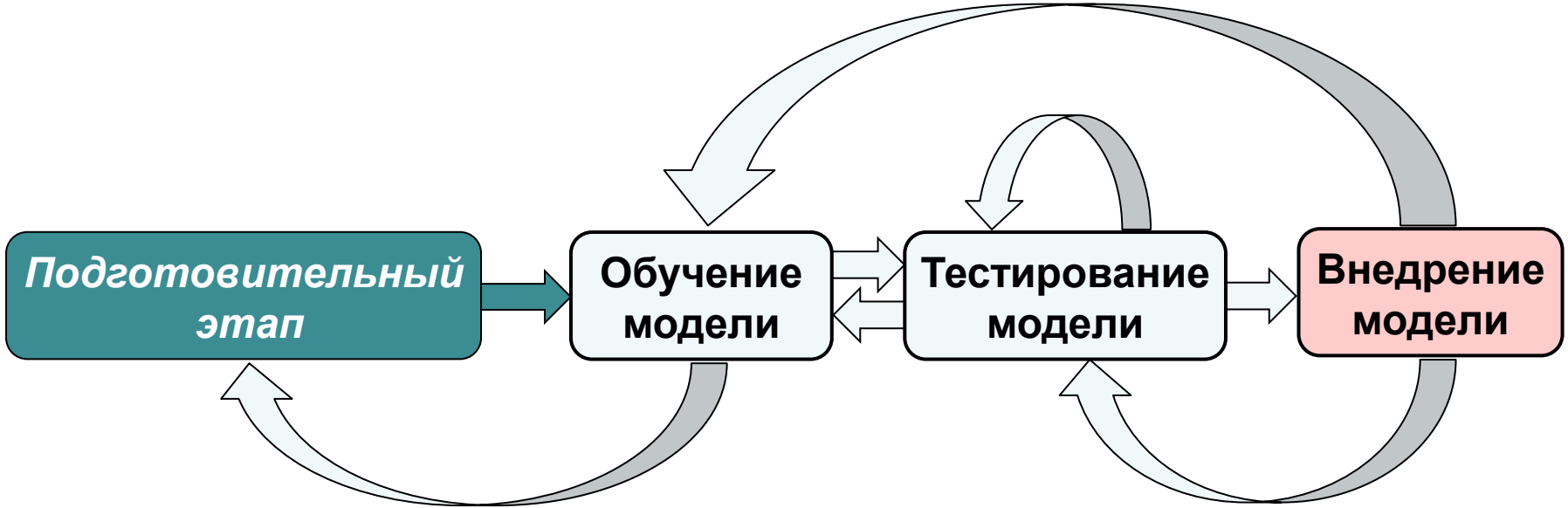


*Много **тонкостей***

*Тензорные компиляторы пытаются помочь программистам, решая часть задач оптимизации **автоматически / автоматизировано***

ПОНЯТИЕ ТЕНЗОРНОГО КОМПИЛЯТОРА

Решение задач средствами глубокого обучения



Обзор литературы

Подготовка данных

Подготовка тестовой инфраструктуры

Разработка топологии

Обучение модели

Верификация модели

Определение качества решения задачи

Внедрение

Многократный вывод

Оптимизация модели

Мотивация…

- Глубокое обучение широко применяется при решении практических задач
- Вывод предполагает многократное выполнение операций над 2-, 3- и 4-мерными матрицами – ***тензорами***
- Типовые преобразования:
  - Полносвязный слой
  - Сверточный слой

Мотивация

Внедрение глубоких моделей предполагает высокую скорость вывода

↓

Необходимо ускорять операции над тензорами

↓

Необходимо разрабатывать специализированные инструменты

↓

Тензорные компиляторы

Понятие тензорного компилятора

- ***Тензорный компилятор*** – отдельный инструмент или компонент какого-либо фреймворка, который обеспечивает оптимизацию и преобразование глубокой нейросетевой модели в исполняемый формат под конкретное устройство

ОБЩАЯ АРХИТЕКТУРА ТЕНЗОРНЫХ КОМПИЛЯТОРОВ

Общая архитектура тензорных компиляторов…

- Архитектура тензорных компиляторов отражает современный подход к проектированию традиционных компиляторов для языков программирования высокого уровня
- ***Основные компоненты:***
  - *Фронтенд (frontend)* – интерфейсная часть
  - *Бэкенд (backend)* – серверная часть

- ***Вход тензорного компилятора*** – модель глубокого обучения в формате какого-либо фреймворка
- ***Выход тензорного компилятора*** – исполняемый код под выбранную целевую платформу вычислительной системы (аналогично традиционным компиляторам)

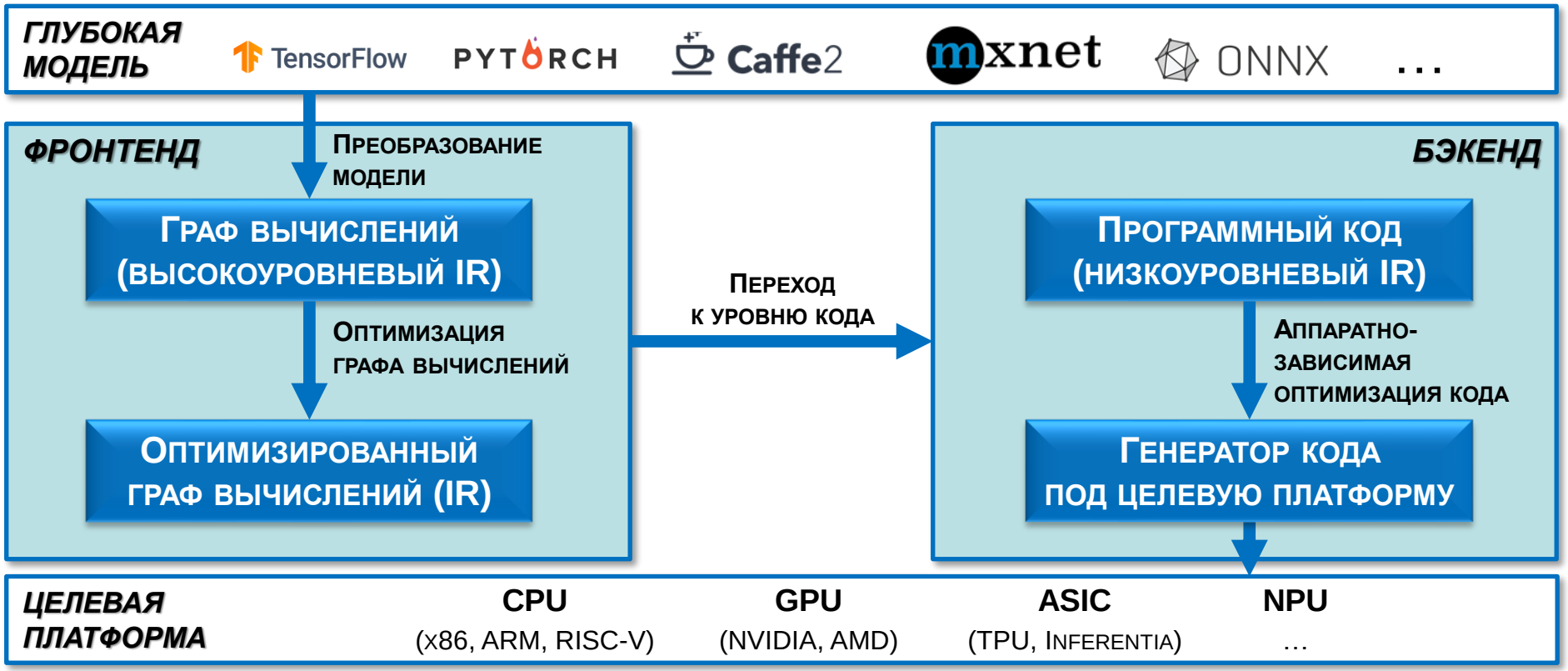
Общая архитектура тензорных компиляторов…

- ***Задача тензорного компилятора*** – преобразование модели глубокого обучения (входа) в исполняемый код под конкретную платформу (выход)

- Разнообразие вариантов представления моделей и множество целевых платформ ведет к тому, что работа тензорного компилятора разбивается на ***стадии***

- ***Результат всех стадий, кроме последней,*** – модель глубокого обучения, преобразованная в какой-либо вариант ***промежуточного представления (intermediate representation, IR)***

Общая архитектура тензорных компиляторов…



Общая архитектура тензорных компиляторов

- Схема работы тензорного компилятора:
  - ***Фронтенд*** принимает нейросетевую модель в формате какого-либо фреймворка глубокого обучения, преобразует в граф вычислений и обеспечивает оптимизацию на уровне графа вычислений
  - ***Бэкенд*** преобразует высокоуровневое представление в виде графа вычислений в низкоуровневое и выполняет аппаратно-зависимые оптимизации
  - Низкоуровневое представление используется для генерации кода (например, для генерации LLVM IR)

Еще раз о мотивации

ПРЕДСТАВЛЕНИЕ НЕЙРОСЕТЕВОЙ МОДЕЛИ В ВИДЕ ГРАФА ВЫЧИСЛЕНИЙ

Граф вычислений…

- Граф вычислений может определяться двумя способами

Вариант 1

- ***Граф вычислений*** состоит из следующих компонент:
  - Узлы – операторы (преобразования)
  - Ребра – тензоры (обрабатываемые данные)
- Тензоры могут представлять:
  - Входные данные
  - Тензоры весов
- Операторы имеют набор атрибутов

- ***Примечание 1:*** граф вычислений не содержит информацию о том, как реализованы операторы
- ***Примечание 2***: граф вычислений не содержит циклов

Граф вычислений

- Граф вычислений может определяться двумя способами

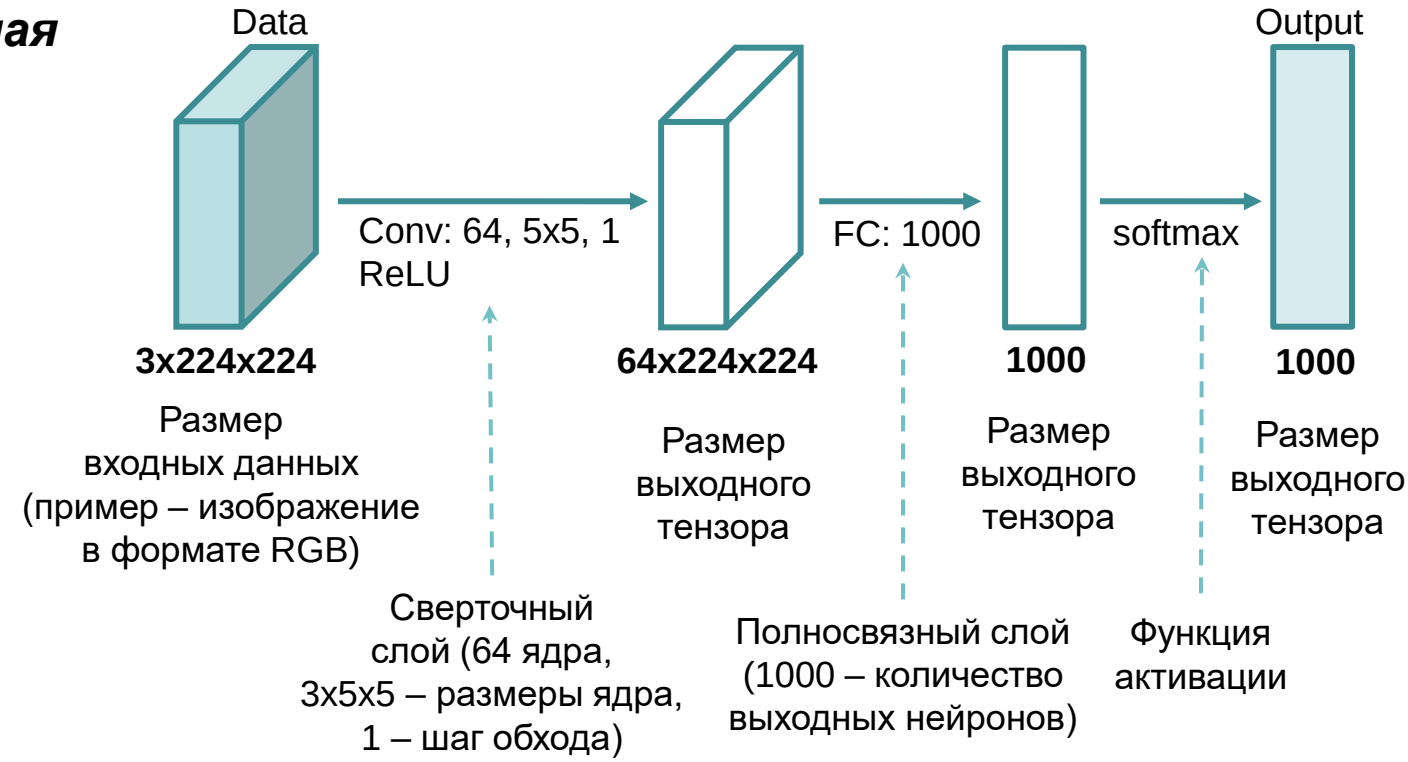
Вариант 2

- ***Граф вычислений*** состоит из следующих компонент:
  - Узлы – данные или операторы (преобразования)
  - Ребра – зависимость от потока данных
- Данные – тензоры:
  - Входные данные
  - Тензоры весов
- Операторы имеют набор атрибутов

- ***Примечание 1:*** граф вычислений не содержит информацию о том, как реализованы операторы
- ***Примечание 2***: граф вычислений не содержит циклов

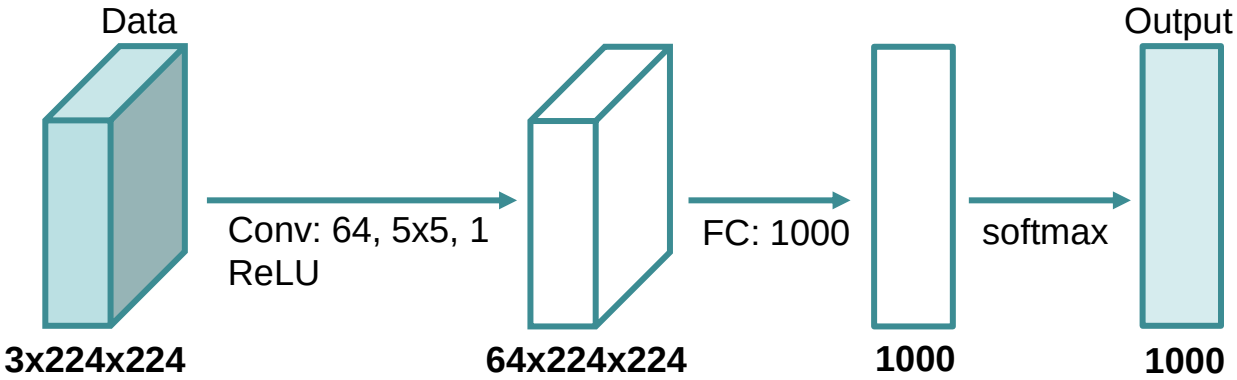
Граф вычислений – пример…

Нейронная сеть

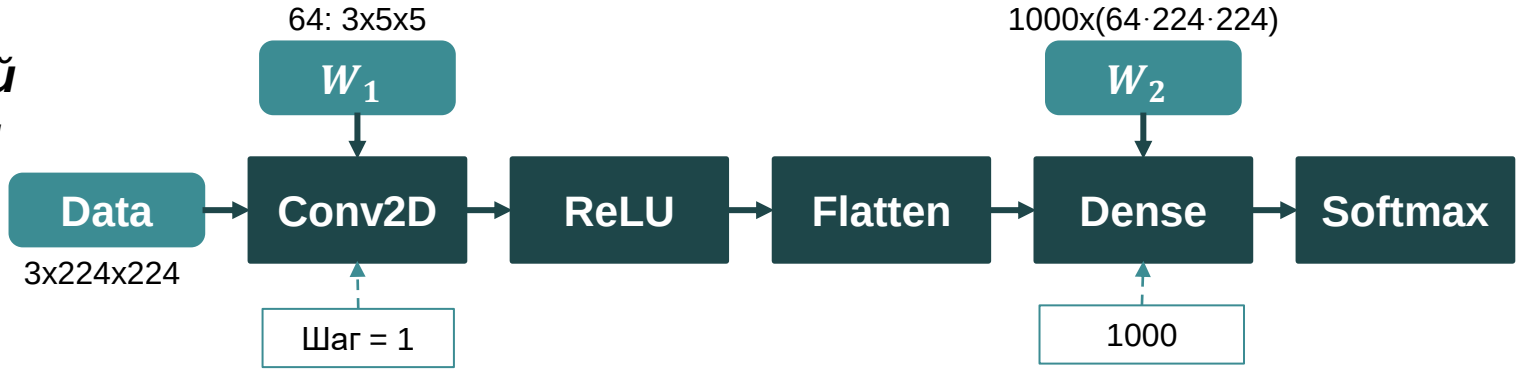


Граф вычислений – пример

Нейронная сеть

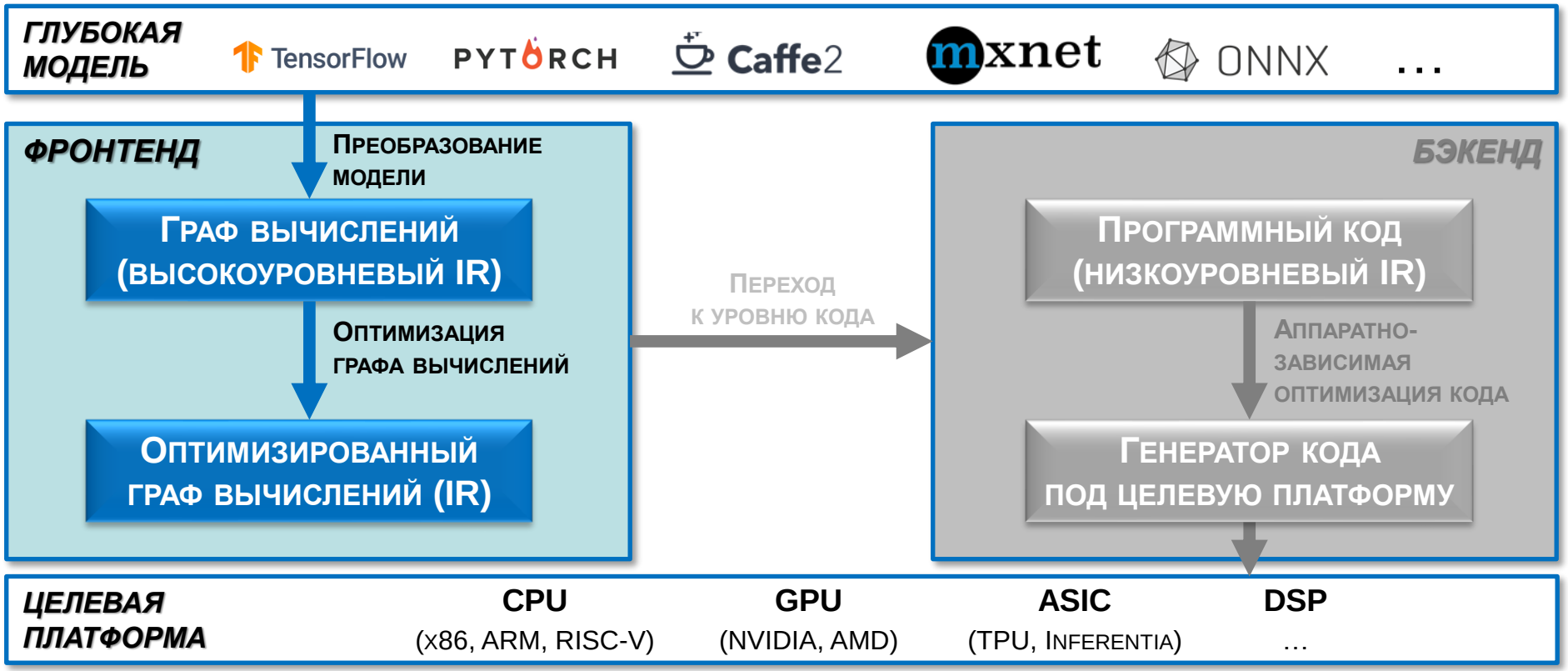


Граф вычислений нейросети



ТИПОВЫЕ ПОДХОДЫ К ОПТИМИЗАЦИИ ГРАФА ВЫЧИСЛЕНИЙ

Общая архитектура тензорных компиляторов



Оптимизация графа вычислений…
Уровни

- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - Блок
  - Поток данных

Оптимизация графа вычислений…
Уровень узла

- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - **Узел** – Блок – Поток данных
- Оптимизация на уровне узла включает:
  - Устранение узлов
    - Пусть A – нулевой тензор,
    - Тогда узел суммирования A и B можно удалить, поскольку результат известен заранее – это тензор B
  - Замену на узлы с более низкой вычислительной стоимостью
    - Пусть A – тензор размерности 0, B – тензор-константа,
    - Тогда узел суммирования A и B можно заменить на узел-константу с тензором B

Оптимизация графа вычислений…
Уровень блока…

- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - **Блок**
  - Поток данных
- Оптимизация на уровне блока включает:
  - Алгебраические упрощения (algebraic simplification)
  - Слияние операторов (operator fusion)

Оптимизация графа вычислений…
Уровень блока…

- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - **Блок**
  - Поток данных
- Оптимизация на уровне блока включает:
  - Алгебраические упрощения (algebraic simplification)
    - Эквивалентные преобразования
      - Пусть требуется вычислить $C = A^T B^T$, что требует два транспонирования и одно умножение матриц
      - Вместо этого можно вычислить $C = (BA)^T$, что требует одно транспонирование и одно умножение матриц
  - Слияние операторов (operator fusion)

Оптимизация графа вычислений…
Уровень блока…

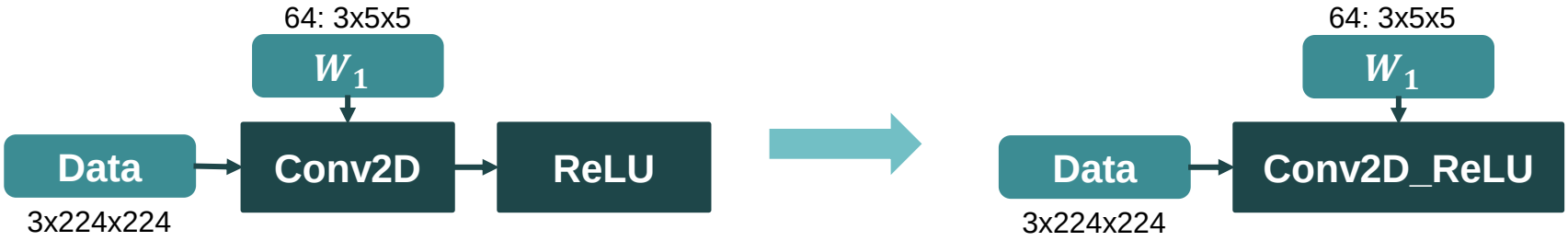
- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - **Блок**
  - Поток данных
- Оптимизация на уровне блока включает:
  - Алгебраические упрощения (algebraic simplification)
    - Свертка констант (constant-folding) – предварительное вычисление частей графа, которые могут быть определены статически

`int x = 14`
`int y = 7-x/2`
`return y*(28/x+2)`
	`int x = 14`
	`int y = 7-14/2`
	`return y*(28/14+2)`
		`int x = 14`
		`int y = 0`
		`return 0`

  - Слияние операторов (operator fusion)

Оптимизация графа вычислений…
Уровень блока

- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - **Блок**
  - Поток данных
- Оптимизация на уровне блока включает:
  - Алгебраические упрощения (algebraic simplification)
  - Слияние операторов (operator fusion) – слияние нескольких преобразований в одно без сохранения в памяти промежуточных результатов



Оптимизация графа вычислений…
Уровень потока данных…

- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - Блок
  - **Поток данных**
- Оптимизация на уровне потока данных включает:
  - Удаление общих подвыражений
  - Удаление «мертвого» кода
  - Статическое планирование памяти
  - Преобразование размещения структур данных

Оптимизация графа вычислений…
Уровень потока данных…

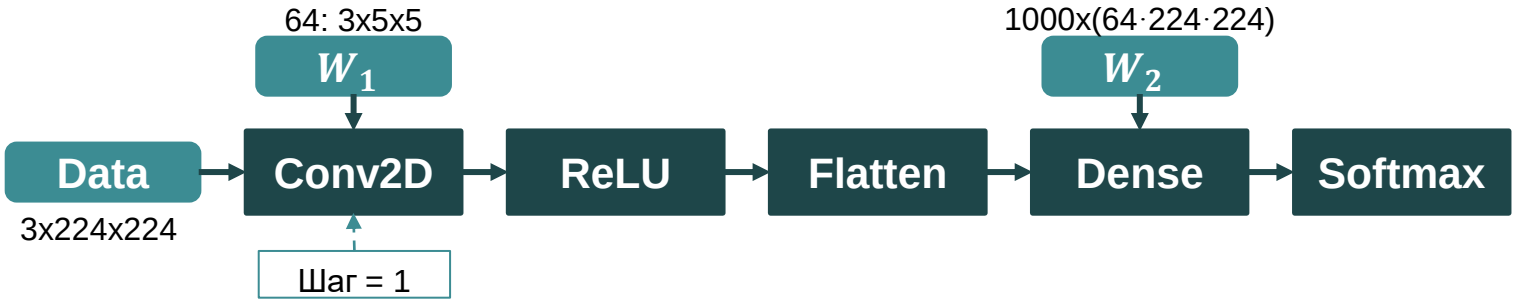
- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - Блок
  - **Поток данных**
- Оптимизация на уровне потока данных включает:
  - Удаление общих подвыражений
    - Пусть требуется вычислить $C = AB$, а затем далее $D = AB + E$, что требует два умножения матриц и одно сложение
    - Вместо этого можно во втором выражении заменить AB на уже вычисленное C, т.е. вычислять $D = C + E$, что требует одно умножение матриц и одно сложение

Оптимизация графа вычислений…
Уровень потока данных…

- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - Блок
  - **Поток данных**
- Оптимизация на уровне потока данных включает:
  - Удаление «мертвого» кода
    - Код называется «мертвым», если его результаты и/или побочные эффекты не используются
    - Обычно «мертвый» код возникает после выполнения других оптимизаций над графом
    - Удаление «мертвого» кода не влияет на конечный результат, но ускоряет вычисления

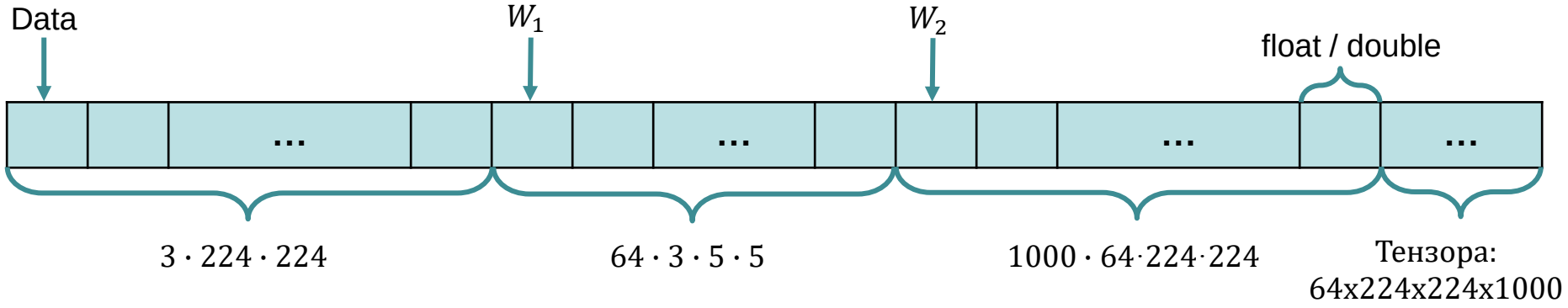
Оптимизация графа вычислений…
Уровень потока данных…

- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - Блок
  - **Поток данных**
- Оптимизация на уровне потока данных включает:
  - Статическое планирование памяти – предварительное выделение памяти



Оптимизация графа вычислений…
Уровень потока данных…

- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - Блок
  - **Поток данных**
- Оптимизация на уровне потока данных включает:
  - Статическое планирование памяти – предварительное выделение памяти

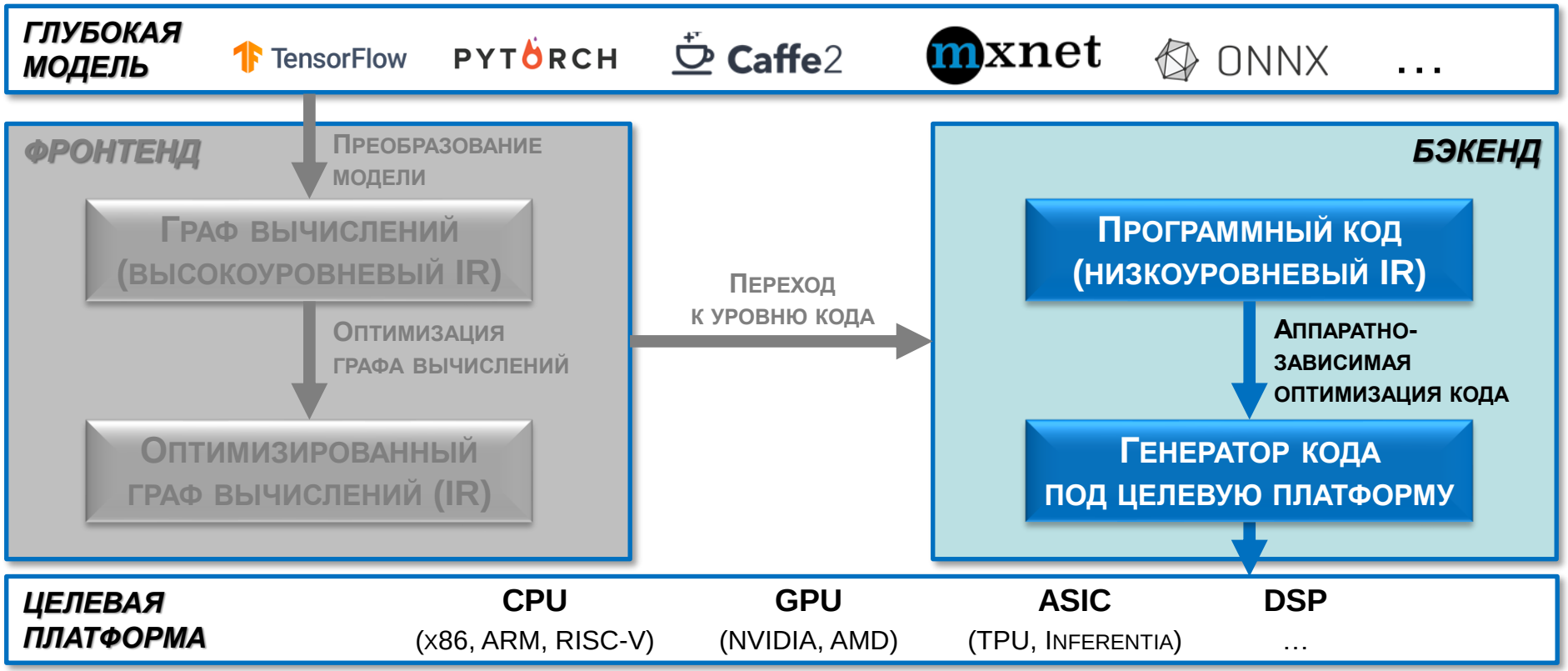


Оптимизация графа вычислений
Уровень потока данных

- Тензорные компиляторы могут выполнять оптимизацию графа вычислений на трех уровнях:
  - Узел
  - Блок
  - **Поток данных**
- Оптимизация на уровне потока данных включает:
  - Преобразование размещения структур данных (layout transformations)
    - Граф вычислений преобразуется так, чтобы использовать более эффективное внутреннее расположение данных для выполнения на целевой платформе
    - Для каждого оператора определяется предпочтительное размещение данных
    - Например, преобразование данных в строчный, столбцовый или блочный формат

ТИПОВЫЕ ПОДХОДЫ К ОПТИМИЗАЦИИ КОДА ОПЕРАТОРОВ

Общая архитектура тензорных компиляторов



Оптимизация кода операторов…

- ***Задача бэкенда тензорного компилятора*** – сгенерировать исполняемый код под целевую платформу
- Эффективное выполнение кода требует использования различных аппаратно-зависимых оптимизаций:
  - Реализация операторов через использование готовых оптимизированных ядер
  - Учет особенностей подсистемы памяти на целевой платформе
  - Оптимизация циклов
  - Распараллеливание

Оптимизация кода операторов…
Использование оптимизированных ядер

- ***Задача бэкенда тензорного компилятора*** – сгенерировать исполняемый код под целевую платформу
- Эффективное выполнение кода требует использования различных аппаратно-зависимых оптимизаций:
  - Реализация операторов через использование готовых оптимизированных ядер
    - Возможность использовать оптимизированные под конкретную целевую платформу библиотеки
      - предоставляют как реализации операторов в целом (например, матричного умножения)
      - так и оптимизированные ядра (например, для эффективного использования векторных инструкций)
    - Возможность «подставлять» собственные реализации операторов, например, написанные в интринсиках

Оптимизация кода операторов…
Учет особенностей подсистемы памяти

- ***Задача бэкенда тензорного компилятора*** – сгенерировать исполняемый код под целевую платформу
- Эффективное выполнение кода требует использования различных аппаратно-зависимых оптимизаций:
  - Учет особенностей подсистемы памяти на целевой платформе
    - Память на различных целевых платформах организована по разным принципам
      - Иерархия кэш-памяти, векторные регистры на CPU
      - Несколько видов памяти на GPU
    - Генерация эффективного кода требует учета организации памяти для улучшения локальности обрабатываемых данных

Оптимизация кода операторов…
Оптимизация циклов

- Задача бэкенда тензорного компилятора – сгенерировать исполняемый код под целевую платформу
- Эффективное выполнение кода требует использования различных аппаратно-зависимых оптимизаций:
  - Оптимизация циклов
    - Разделение цикла
    - Слияние циклов
    - Векторизация цикла
    - Разворачивание цикла
    - Переупорядочивание циклов

Оптимизация кода операторов
Распараллеливание

- Задача бэкенда тензорного компилятора – сгенерировать исполняемый код под целевую платформу
- Эффективное выполнение кода требует использования различных аппаратно-зависимых оптимизаций:
  - Распараллеливание
    - Современные целевые платформы являются не только SIMD-устройствами, но и многопоточными/многоядерными
    - Технологии разработки программ поддерживают многопоточное/параллельное программирование
    - В отдельных случаях (например, распараллеливание простых циклов) могут помочь бэкенды универсальных компиляторов (например, LLVM)

Заключение

- Тензорные компиляторы обеспечивает оптимизацию и преобразование глубокой нейросетевой модели в исполняемый формат под конкретное вычислительное устройство
- Тензорные компиляторы выполняют оптимизацию вывода нейросетевой модели на двух уровнях:
  - Граф вычислений (во фронтенде компилятора)
  - Код операторов (в бэкенде компилятора)

Литература

1. Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press. – 2016. – [http://www.deeplearningbook.org].
2. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. Погружение в мир нейронных сетей. – Изд-во «Питер». – 2018. – 476 с.
3. Учебно-образовательный курс «Современные методы и технологии глубокого обучения в компьютерном зрении (версия 2)» [http://hpc-education.unn.ru/ru/обучение/курсы/магистратура/deep_learning_in_computer_vision_2].
4. Li, Mingzhen & Liu, Yi & Liu, Xiaoyan & Sun, Qingxiao & You, Xin & Yang, Hailong & Luan, Zhongzhi & Gan, Lin & Yang, Guangwen & Qian, Depei. (2021). The Deep Learning Compiler: A Comprehensive Survey. IEEE Transactions on Parallel and Distributed Systems. 32. 708-727. 10.1109/TPDS.2020.3030548.

Авторский коллектив

- Мееров Иосиф Борисович, к.т.н., доцент, зав. каф. ВВиСП Института ИТММ ННГУ им. Н.И. Лобачевского
- Кустикова Валентина Дмитриевна, к.т.н., доцент каф. ВВиСП Института ИТММ ННГУ им. Н.И. Лобачевского
- Родимков Юрий Александрович, младший научный сотрудник каф. ВВиСП Института ИТММ ННГУ им. Н.И. Лобачевского
- Сысоев Александр Владимирович, к.т.н., доцент каф. ВВиСП Института ИТММ ННГУ им. Н.И. Лобачевского