Презентация к предзащите дипломной работы  
Тема: "Аннотирование промежуточного представления MLIR на основе профиля исполнения"

Слайд 1: Заголовок и мотивация  
Здравствуйте, меня зовут [Ваше Имя], и моя дипломная работа посвящена созданию системы аннотирования промежуточного представления (IR) компилятора MLIR на основе профиля исполнения.

Цель работы — использовать информацию о времени выполнения и структуре вызовов операций, полученную во время профилирования, для улучшения оптимизаций в компиляторе. Такая обратная связь всё активнее используется в современных компиляторах — например, в LLVM или TensorFlow.

Слайд 2: Что такое MLIR и зачем он нужен  
MLIR — это инфраструктура для построения компиляторов с поддержкой многоуровневого промежуточного представления. Он используется в таких системах, как LLVM, TensorFlow и XLA.

Главная особенность MLIR — возможность создавать собственные диалекты. Это позволяет точно описывать операции конкретных вычислительных моделей и управлять процессом оптимизаций и поэтапного понижения IR (lowering).

Слайд 3: Проблема — как использовать профиль в компиляции  
Сегодня компиляторы часто применяют эвристики без знания реального поведения программы. Однако современные фреймворки, такие как TensorFlow, позволяют собирать точные данные профиля: какие функции самые "горячие", какие операции дольше всего выполняются и т.д.

Вопрос — как передать эти данные обратно в компилятор и использовать их для оптимизации? MLIR по умолчанию не предоставляет общего механизма для такого аннотирования. Моя работа направлена на решение этой задачи.

Слайд 4: Мое решение — аннотирование IR через пользовательский диалект  
Я создаю легковесный диалект annot, который оборачивает или дополняет другие операции метаданными из профиля.

**Сохраняет стандартные диалекты** при понижении (lowering)

**Добавляет структурированный синтаксис** для аннотаций с валидацией

**Позволяет очистить или понизить аннотации** в отдельном проходе

Пример:

annot.for attributes {metadata = {unroll = 4}} {

affine.for %i = 0 to 16 {

...

}

}

Эта аннотация указывает компилятору, что этот цикл стоит развернуть 4 раза. Такие метаданные могут использоваться при понижении IR или в кастомных оптимизационных проходах.

Слайд 5: Архитектура системы  
Вот как работает вся система:

* Сначала мы собираем профиль с помощью внешнего инструмента, например, TensorFlow Profiler
* Данные сохраняются в YAML-формате
* Мой инструмент считывает YAML и вставляет аннотации в MLIR
* Затем проходят компиляторные преобразования, использующие эти аннотации

Такую систему можно легко расширять: добавлять новые метаданные, новые диалекты и правила трансформаций.

Слайд 6: Пример сквозного применения  
Пример:

* Профиль сообщает, что функция matmul занимает 24мс и выполняется на GPU
* Мы аннотируем annot.func @matmul соответствующими атрибутами
* Оптимизационный проход принимает решение, например, о тиле или перераспределении вычислений

Таким образом, мы строим универсальный механизм передачи информации из исполнения в компиляцию.

Слайд 7: Выводы и дальнейшие шаги  
Итак:

* Разрабатывается расширяемый фреймворк для аннотирования MLIR на основе профиля исполнения
* Используется диалект annot и YAML-схема для метаданных
* Создаются проходы, применяющие эти данные для оптимизации

Следующие шаги:

* Тестирование на реальных трассах TensorFlow
* Разработка CLI-инструмента
* Добавление поддержки анализа вызовов и графов зависимостей

Спасибо за внимание! Готов ответить на вопросы.