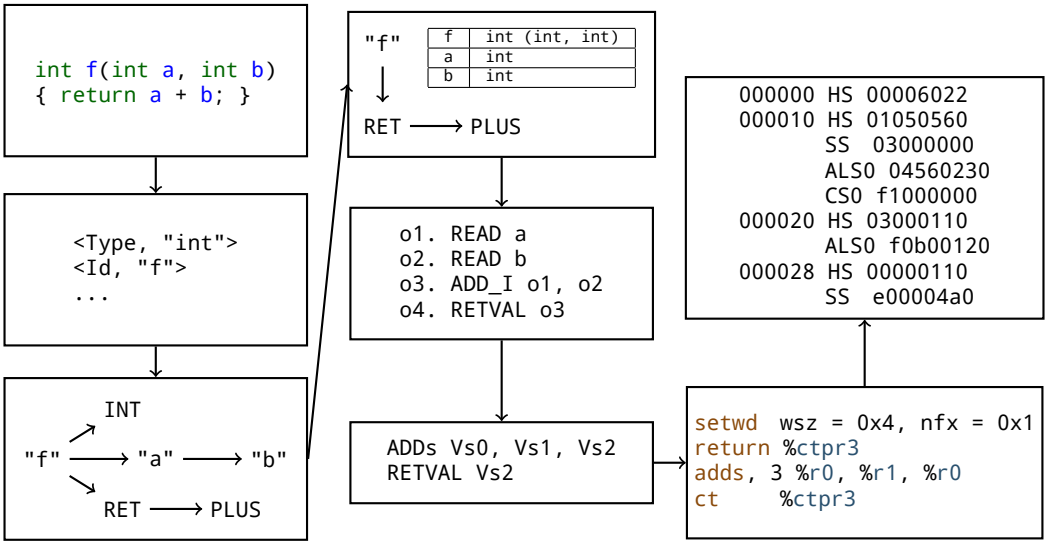
Проектирование компиляторов

Лекция 8.
Введение в оптимизации.

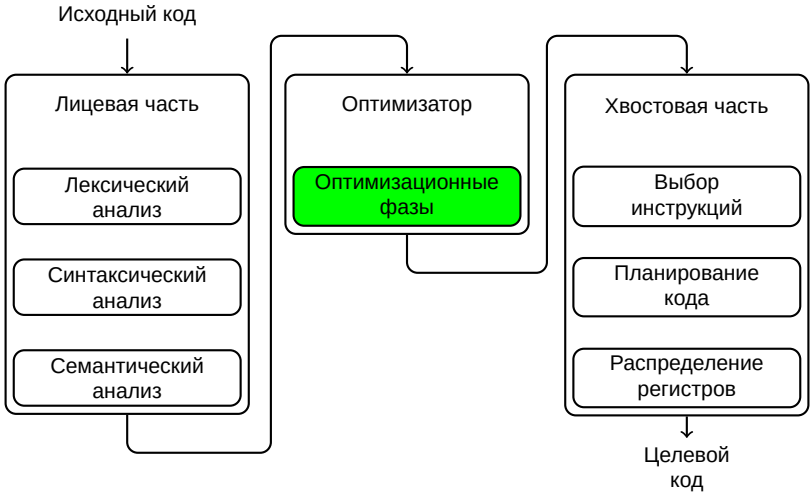
Маркин А. Л.
alexanius@gmail.com

2021

Компиляция программы



Компиляция программы



Оптимизации

Первые оптимизации в компиляторах стали появляться ещё в 50-х годах, вскоре после появления самих компиляторов.

В 80-х годах стало выделяться два вида компиляторов:

1. **Неоптимизирующие (отладочные)** — нацелены на максимально быструю компиляцию и простую отладку. Почти не перемешивают код, что позволяет быстро находить ошибки в исходных программах.
2. **Оптимизирующие** — ценой увеличения времени компиляции и крайне затруднительной отладки уменьшают время исполнения программы и используют возможности целевой платформы.

Оптимизации

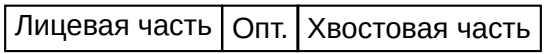
Оптимизация — преобразование программы, нацеленное на улучшение её характеристик.

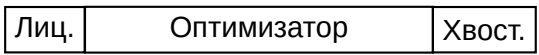
Первые компиляторы создавали более медленный код чем человек и не применялись для ускорения.

На сегодняшний момент человек не способен на столько же эффективно писать программы на ассемблере. Исключение составляют отдельные небольшие участки кода, которые компилятор не всегда может транслировать максимально эффективно.

Оптимизации

Пропорции времени работы различных стадий компилятора на ранних этапах развития и сейчас:





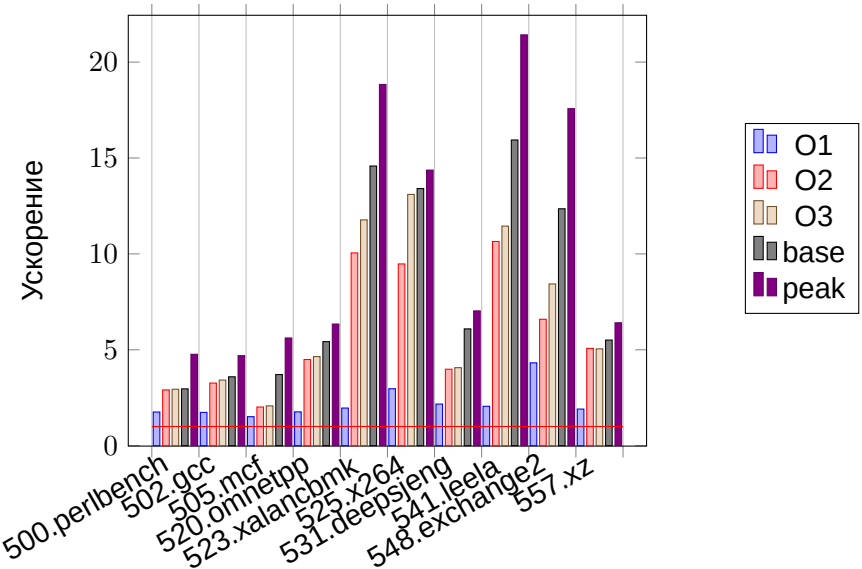
Оптимизации

Для чего применяются оптимизации:

- Ускорение программы
  - Исправление неоптимальности кода программиста
  - Использование возможностей целевой платформы
- Оптимизация потребляемых ресурсов
  - Размер исполняемого файла
  - Потребление памяти
  - Энергозатраты
- Поиск ошибок
  - Статический (анализ программы)
  - Динамический (анализ исполнения)

Оптимизации

Пример ускорения тестов spec 2017 intrate на процессоре Эльбрус:



Оптимизации

Помимо непосредственно оптимизаций, в компиляторе различают несколько возможных операций:

Трансформация — преобразование, программы носящее технический характер.

Анализ — сбор информации о программа для дальнейшего применения трансформации или оптимизации.

Фаза (pass) — применение одной оптимизации, трансформации или анализа.

Оптимизации

Оптимизации могут классифицироваться по зависимости от аппаратной платформы:

- **Архитектурно-независимые** — универсальные оптимизации, не опирающиеся на возможности целевой платформы.
- **Архитектурно-зависимые** — оптимизации, использующие возможности целевой платформы для ускорения итоговой программы.

Оптимизации

Оптимизации применяются к различным областям программы:

- **Локальные** — оптимизации применяются внутри одного линейного участка. Обычно это простые оптимизации, нацеленные на устранение избыточности кода и использование аппаратных возможностей платформы.
- **Регионные** — оптимизации применяются к набору линейных участков внутри графа потока управления. Такие оптимизации могут быть направлены на ускорение работы циклов, использование аппаратных возможностей, устранение избыточности, применение архитектурно-независимых оптимизаций.
- **Глобальные (внутрипроцедурные)** — оптимизации применяются ко всей процедуре, могут оценивать её характеристики и принимать решение об эффективности тех или иных оптимизаций.

Оптимизации

Оптимизации применяются к различным областям программы:

- **Межпроцедурные** — оптимизации используют граф вызовов и оптимизируют взаимодействие процедур друг с другом. Такие оптимизации являются архитектурно-независимыми.
- **Межмодульные** — расширение области видимости межпроцедурных оптимизаций на всю программу. Для их применения необходимо применять режим **вся программа**, объединяющий все модули компилируемой программы в один.

Оптимизации

При проведении оптимизации необходимо учитывать несколько факторов:

- **Безопасность** — оптимизация не должна менять поведение программы на любых входных данных.
- **Целесообразность** — оптимизация должна приносить определённую выгоду (ускорение программы, возможность применения других оптимизаций), при этом выгода должна быть адекватна времени, потраченному на компиляцию.

Оптимизации

Правило неизменности (As-If rule): оптимизации, проводимые компилятором не должны менять обозримого поведения программы.

В некоторых языках возможны ситуации, вызывающие **неопределённое поведение**, т.е. ошибки в программе, которые сложно или невозможно проверить на этапе компиляции. Для таких ошибок компилятор имеет право выполнять **любые** преобразования и вызывать **любые** эффекты.

Линейки оптимизаций

За время своей работы, компилятор может применять несколько сотен оптимизаций. Оптимизации могут преследовать разные цели, поэтому существуют готовые наборы оптимизаций, называемые **линейками**:

- `-O1` — линейка оптимизаций, уменьшающих размер исполняемого файла и ускоряющий исполнение. Немного увеличивает время компиляции.
- `-O2` — набор оптимизаций `-O1` плюс дополнительные оптимизации производительности, увеличивающие время компиляции.
- `-O3` — набор оптимизаций `-O2` плюс дополнительные агрессивные оптимизации производительности.

Линейки оптимизаций

Бывают и более агрессивные линейки оптимизаций:

- `-ffast-math` — включает алгебраические оптимизации над операциями с плавающей точкой. Может приводить к изменению точности вещественных вычислений. Требует обязательного ознакомления с документацией по опции.
- `-fstack-arrays` — оптимизация для языка Фортран, при которой компилятор кладёт все массивы известного размера и времени жизни на стек. Для её использования может потребоваться модификация предельных значений стека в окружении (команда `ulimit`).
- `-Ofast` — в **gcc** набор оптимизаций `-O3` плюс, оптимизации отменяющие строгое следование стандарту, например `-ffast-math` и `-fstack-arrays`

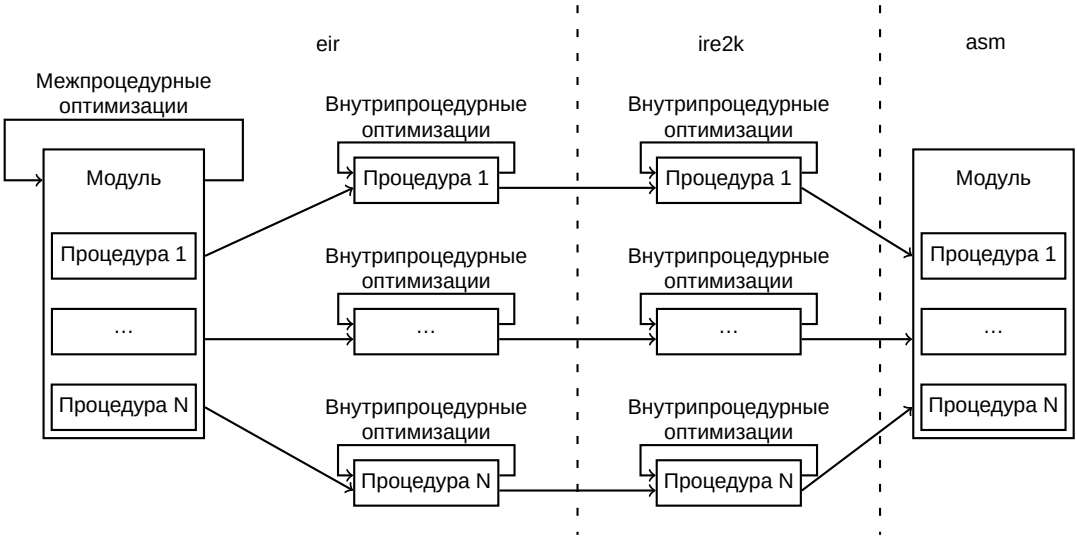
Линейки оптимизаций

Слишком агрессивные оптимизации могут вызывать рост кода, увеличение времени исполнения, и, как следствие общее ухудшение производительности целевой программы.

Для контроля работы оптимизаций применяются **эвристики** — некоторые полученные экспериментальным путём значения, регулирующие поведение оптимизаций.

Например, для ограничения времени работы в **Icc** применяется эвристика: $3\%(T_{\text{comp}}) \sim 1\%(T_{\text{exec}})$, т.е. при увеличении времени компиляции на три процента производительность должна вырастать на один процент.

Схема работы оптимизатора



Литература

- *А. В. Ахо, М. С. Лам, Р. Сети, Д. Д. Ульман* Компиляторы: принципы, технологии и инструменты.
- *S. S. Muchnick* Advanced Compiler Design and Implementation.
- *K. D. Cooper, L. Torczon* Engineering a compiler.
- *R. Allen, K. Kennedy* Optimizing Compilers for Modern Architectures: A Dependence-based Approach.
- *A. Aiken* CS143 Compilers. Stanford lectures.