Масштабируемая система оптимизации времени связывания

Ксения Долгорукова, Семён Аришин
ИСП РАН

Содержание

- Введение
- Постановка задачи
- Обзор существующих решений
- Описание предлагаемой архитектуры
- Масштабирование по памяти
- Горизонтальное масштабирование
- Проблемы сборки больших приложений

Введение

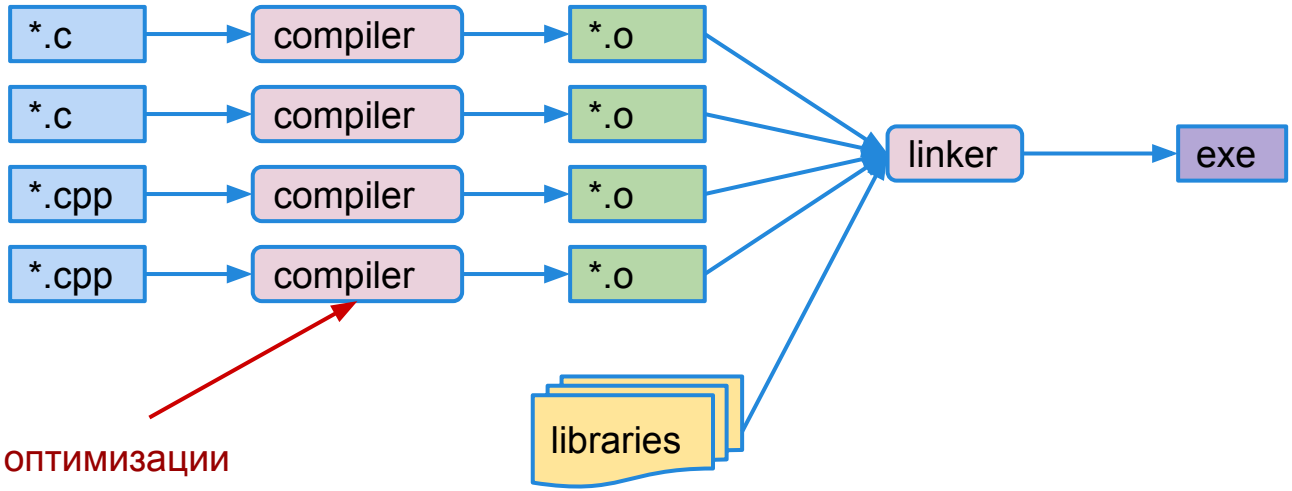
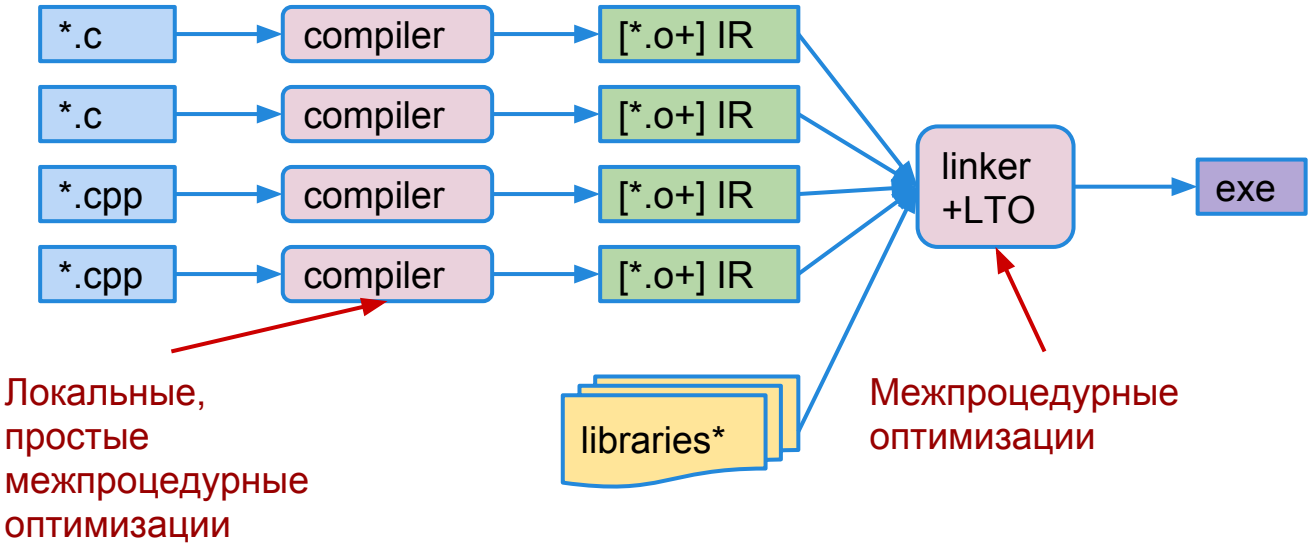

*.c
compiler
*.o
*.c
compiler
*.o
*.cpp
compiler
*.o
*.cpp
compiler
*.o
linker
exe
libraries
оптимизации

Схема сборки программы из исходного кода с LTO



Оптимизации времени связывания (Link-Time Optimization)

- Весь код в памяти
- Разрешены все коллизии имён
- Есть все определения методов статически связываемых компонент

Проблемы LTO

Сборка крупных систем:

- Операционные системы
- Браузеры
- Компиляторы
- СУБД
- Многофункциональные редакторы (текста, изображений, видео, аудио и т.д.)
- ...

Проблемы LTO

Сборка крупных систем:

- Android: 69005 C/C++ файлов, 582 Мбайта
- Firefox: 36555 C/C++ файлов, 241 Мбайт
- LLVM: 2846 C/C++ файлов, 46 Мбайт
- GCC: 46103 C/C++ файлов, 157 Мбайт
- PostgreSQL: 1762 C/C++ файла, 34 Мбайта
- LibreOffice: 19838 C/C++ файлов, 305 Мбайт
- ...

Проблемы LTO

Приблизительное время сборки и потребляемая память (GCC и LLVM с LTO) на x86-64

- Firefox: 11-26 минут, 6-34 Гб ОЗУ [1]
- LibreOffice: 61-68 минут, 8-14 Гбайт ОЗУ [2]

1 http://hubicka.blogspot.ru/2014/04/linktime-optimization-in-gcc-2-firefox.html

Масштабируемость

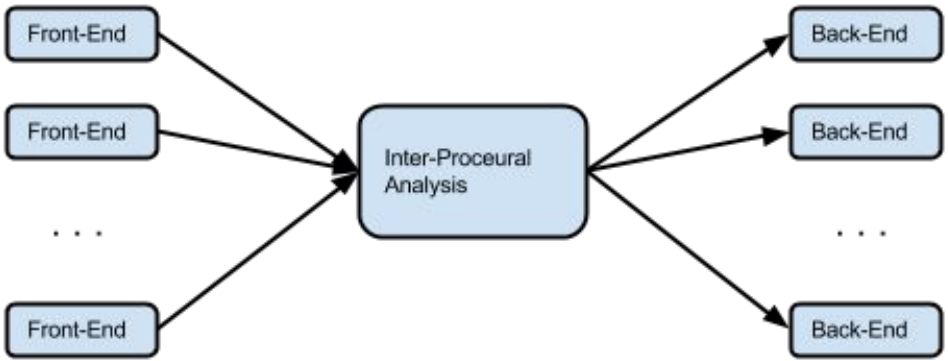
- В общем случае: способность ПО адаптироваться к большим нагрузкам в зависимости от возможностей аппаратуры
- Масштабируемость по памяти - способность сохранять работоспособность при обработке больших объемов данных в ограниченных возможностях по памяти

Постановка задачи

- Разработка архитектуры системы оптимизации времени связывания на основе LLVM, допускающей масштабирование по памяти и горизонтальное масштабирование
- Разработка методов оптимизации программ, работающих в рамках концепции масштабирования
- Реализация разработанных компонентов и методов в системе LLVM

Общий подход

- Разбиение этапа LTO на независимые задачи, которые можно выполнять параллельно



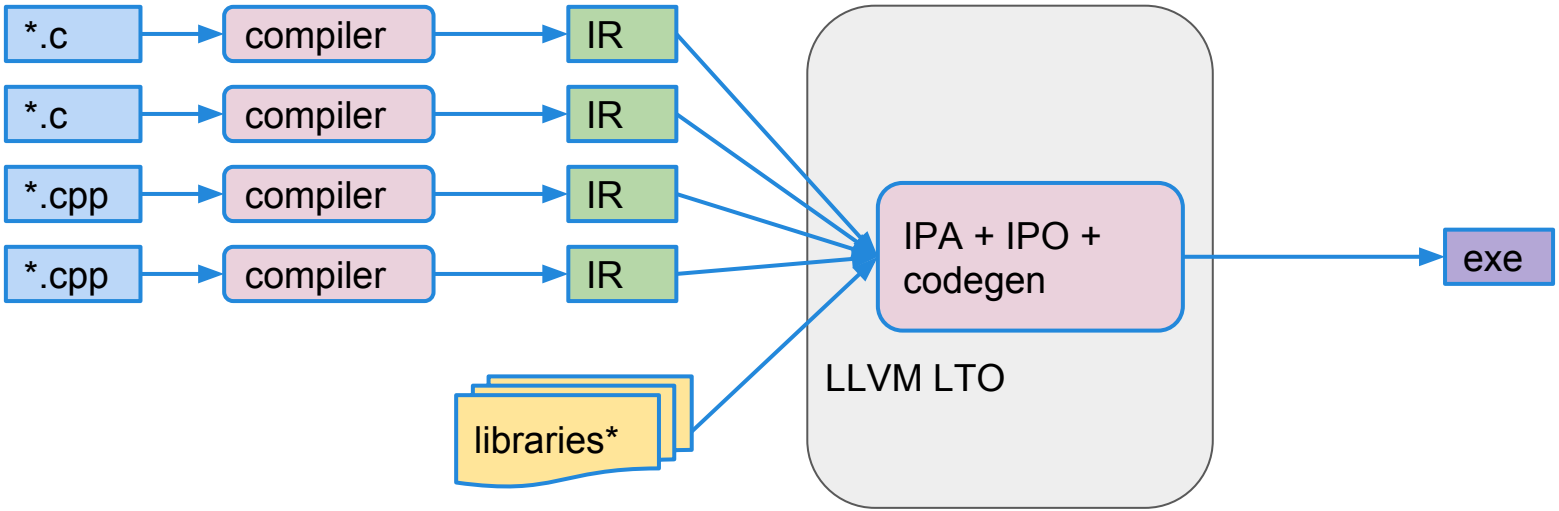
Существующие решения: масштабируемость по памяти

- Подходы к масштабированию в существующих системах
  - Диспетчер с отгрузкой неиспользуемых участков кода на диск (HLO)
  - Работа анализа только на аннотациях (GCC)
  - Совмещение run-time компоненты с анализом (Google LIPO)

Существующие решения: горизонтальное масштабирование

- Разбиение по модулям (HP SYZYGY)
- Разбиение модулей на несколько, возможно, пересекающихся подмножеств (GCC, Google LIPO)

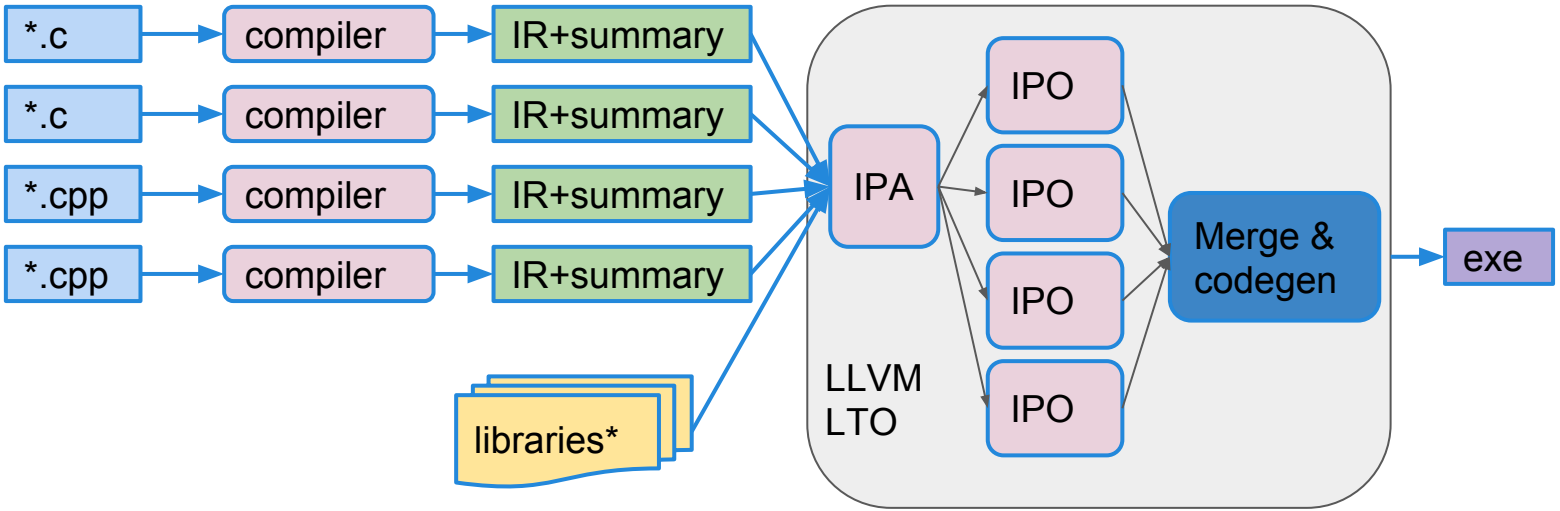
Схема работы LTO в LLVM



Предлагаемое решение

1. Разделить межпроцедурные анализ и оптимизации для проведения оптимизаций в несколько потоков
2. Ленивая загрузка участков кода (процедур) во время анализа, а также выгрузка участков кода во время анализа
3. Разбиение кода на участки непосредственно перед этапом оптимизации, проводить тяжеловесные оптимизации независимо и параллельно

Схема предлагаемой масштабируемой архитектуры



Разделение анализа и оптимизаций

- Аннотирование функций дополнительной информацией во время генерации IR
- Использование аннотаций для межпроцедурного анализа без использования непосредственно кода
- Проводить ресурсоёмкие оптимизации на условно независимых участках кода

Масштабирование по памяти

Масштабирование по памяти: общий метод

Разделение загрузки кода из файлов с IR на этапы

- Предварительная загрузка (типы, объявления процедур, глобальных переменных)
- Предварительная компоновка
- *Предварительный анализ аннотаций*
- Загрузка тел процедур
- Компоновка загруженной процедуры в контексте ранее скомпонованных типов и переменных
- Выгрузка тел при достижении порога потребления памяти

Масштабирование по памяти: предварительный анализ

Предварительный анализ аннотаций, функциональные требования:

- Получение информации, достаточной для проведения ленивой загрузки кода
- Сборка информации, необходимой для легковесных оптимизаций
- Анализ структуры программы в целях распараллеливания оптимизаций

Масштабирование по памяти: предварительный анализ

Предварительный анализ аннотаций, требования:

- Требует мало ресурсов
- Быстрый

Реализация масштабирования по памяти: аннотирование

Устройство IR-файлов LLVM (llvm-bitcode)

- Каждый bitcode файл соответствует одному модулю
- Каждый bitcode файл – это битовый поток
- Каждый bitcode файл состоит из блоков, выровненных по байтам
- Блоки бывают нескольких типов, соответствующих определенным структурам IR
- В блоке может содержаться сколько угодно подблоков
- Каждой процедуре соответствует блок, состоящий из подблоков заголовка и блока тела
- Каждому заголовку процедуры добавляется подблок с аннотациями

Реализация масштабирования по памяти: аннотирование

Содержимое блока с аннотациями

- Размер блока тела процедуры в байтах
- Информация о вызовах внутри процедуры для построения графа вызовов
- Количество инструкций в процедуре
- Информация об использовании глобальных переменных
- ...

Реализация масштабирования по памяти: ленивая загрузка

Предварительная загрузка

- Считывание и разбор блоков с описаниями типов, глобальных переменных, процедур, инициализаций глобальных переменных, аннотаций
- Пропуск тел процедур
- Сохранение информации, необходимой для быстрого поиска тел функций при повторном считывании

Реализация масштабирования по памяти: ленивая загрузка

Предварительная компоновка

- Разрешение коллизий имён
- Уточнение типов
- Уточнение ссылок
- Копирование сформированных сущностей в композитный модуль

Реализация масштабирования по памяти: ленивая загрузка

Загрузка и компоновка тел процедур

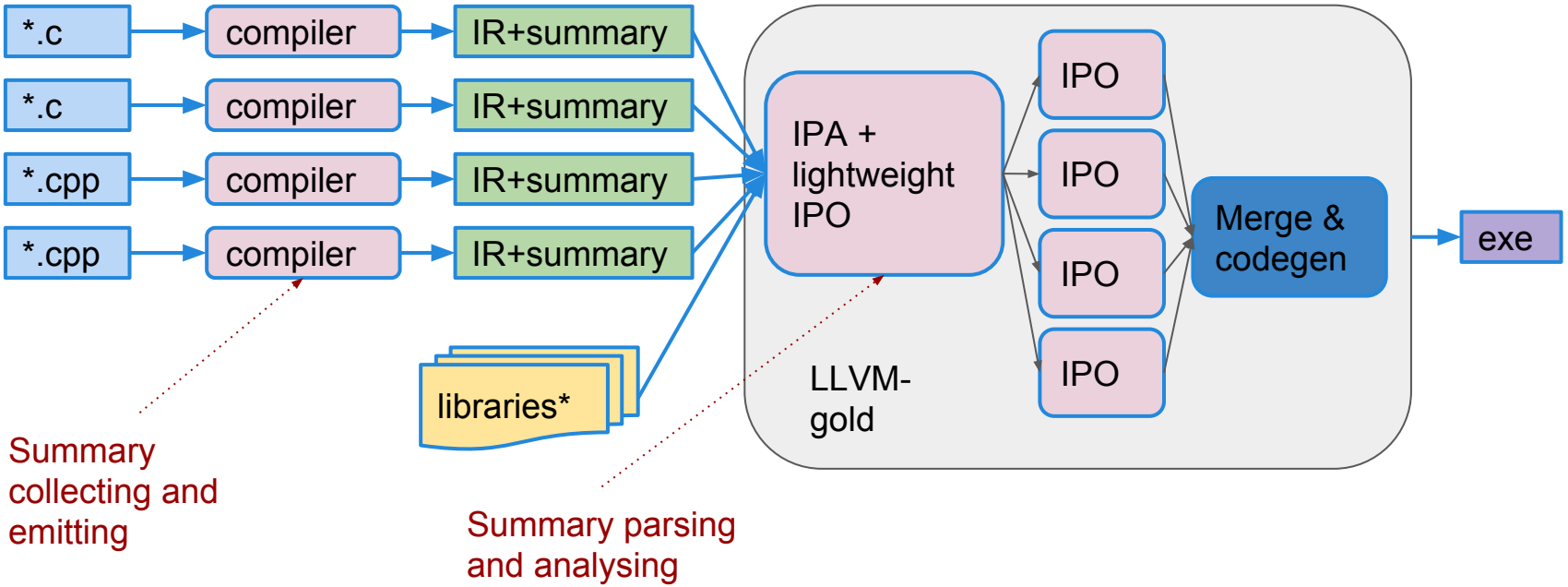
- Загрузка кода по сохранённым смещениям
- Компоновка загруженного кода в композитный модуль с учётом разрешенных ранее коллизий
- Удаление вспомогательных данных из памяти

Реализация масштабирования по памяти: выгрузка

Выгрузка тел процедур

- Менеджер памяти при каждой новой загрузке кода проверяет достижение заданного пользователем порога потребления памяти
- Менеджер выгружает обработанные участки кода на диск
- Если необходимо, менеджер снова подгружает необходимые участки кода

Реализация масштабирования по памяти: легковесные оптимизации



Легковесная оптимизация удаления глобальных

a.c

```
int a;
extern int b;
char *x;

foo1(){
  int d = b*b;
}
...
```

b.c

```
int b;
extern a;
extern char *x;

foo2(){
 printf(«%s», x);
}
...
```

***.bc**

```
int a;
int b;
char *x;

foo1()
foo2()
```

{i32* @b, i32 ()* @foo1, i32 2} {i8** @x, i32 ()* @foo2, i32 1}

метаданные

{i32* @b, i32 ()* @foo1, i32 2}
{i8** @x, i32 ()* @foo2, i32 1}

Вывод: Удаляем a

Горизонтальное масштабирование

Разбиение промежуточного кода (IR)

Горизонтальное масштабирование

Разбиение промежуточного кода (IR)

=>

Разбиение графа вызовов

Горизонтальное масштабирование: постановка задачи

- Исследование критериев разбиения графов
- Обзор существующих алгоритмов разбиения графов
- Исследование свойств графов вызовов
- Разработка алгоритма разбиения
- Реализация и тестирование алгоритма на реальных программах

Горизонтальное масштабирование: постановка задачи

- Исследование критериев разбиения графов
- Обзор существующих алгоритмов разбиения графов
- Исследование свойств графов вызовов
- Разработка алгоритма разбиения
- Реализация и тестирование алгоритма на реальных программах

Критерии разбиения (кластеризации) графа

- Что значит разбить граф на кластеры?
  - Это значит выделить в нем “наиболее связанные” множества вершин
- С математической точки зрения:

  Разбить граф $G=(V,E)$, $|V| = n$, $|E|=m$ на такие

  $$\delta_{\text{int}}(\mathcal{C}) = \frac{|\{\{v,u\} \mid v \in \mathcal{C}, u \in \mathcal{C}\}|}{|\mathcal{C}|\,(|\mathcal{C}|-1)}.$$

  внутренняя плотность каждого

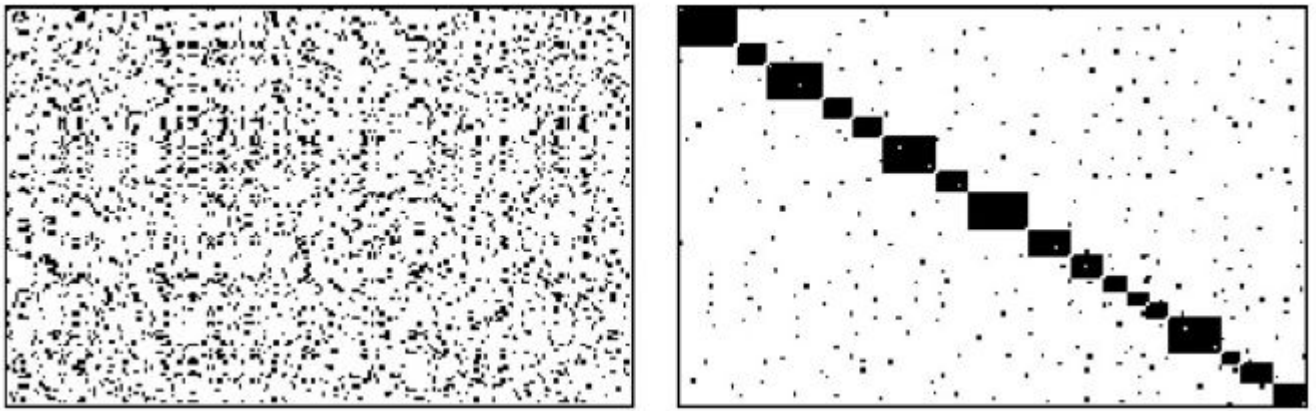
  будет превосходить внешнюю

  $$\delta_{\text{ext}}(G \mid \mathcal{C}_1, \ldots, \mathcal{C}_k) = \frac{\left|\left\{\{v,u\} \mid v \in \mathcal{C}_i, u \in \mathcal{C}_j, i \neq j\right\}\right|}{n(n-1) - \sum_{\ell=1}^{k} (|\mathcal{C}_\ell|\,(|\mathcal{C}_\ell|-1))}.$$

Критерии разбиения (кластеризации) графа

- Что значит разбить граф на кластеры?

  Или привести матрицу инцидентности к блочно-диагональному виду



Методы оценки разбиения графов

Относительная плотность кластера S

$$\frac{\delta(G(S))}{\delta(G)}$$

δ(G(S)) = |E(S)| / |S|,
δ(G(S)) = |E| / |V| = m/n

Мера модулярности

$Q = \boldsymbol{\Sigma}_i(\boldsymbol{e}_{ii} - \boldsymbol{a}_i^2)$

где $\boldsymbol{a}_i = \boldsymbol{\Sigma}_j\ \boldsymbol{e}_{ij}$ для матрицы относительных степеней кластеров, E = {$\mathbf{e}_{ij}$}, где для кластеров C_i и C_j $\boldsymbol{e}_{ij}$ = $|(u,v)|/m$, $u \in C_i$, $v \in C_j$, а $\boldsymbol{e}_{ii} = deg(C_i)/m$

Горизонтальное масштабирование: постановка задачи

- Исследование критериев разбиения графов
- Обзор существующих алгоритмов разбиения графов
- Исследование свойств графов вызовов
- Разработка алгоритма разбиения
- Реализация и тестирование алгоритма на реальных программах

Классификация методов кластеризации графов

- Глобальные
  - Итеративные
  - Иерархические
    - Агломеративные
    - Дивизивные
      - На основе разрезов
      - На основе максимального потока
      - Спектральные
      - На основе меры промежуточности
      - На основе разности потенциалов
      - На основе цепей Маркова
- Локальные

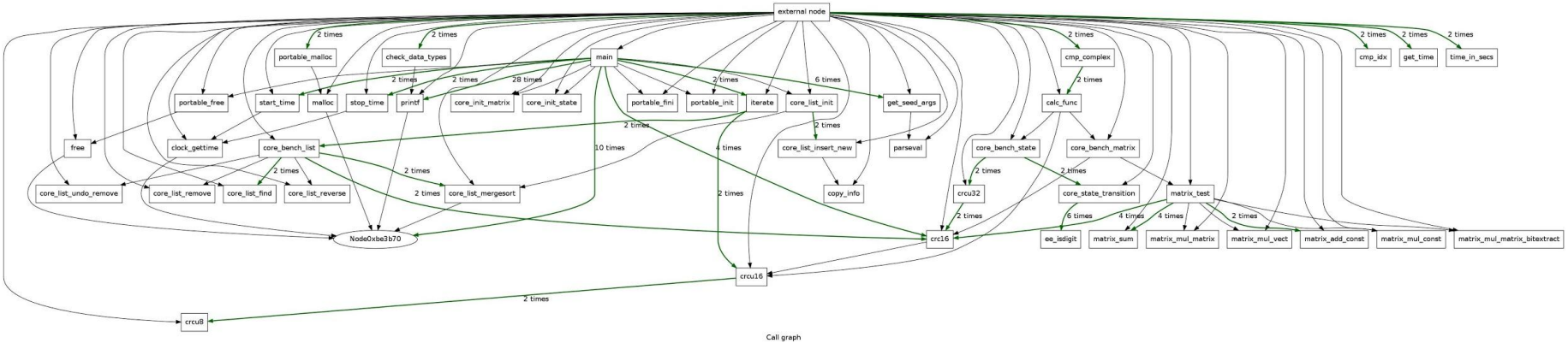
Горизонтальное масштабирование: постановка задачи

- Исследование критериев разбиения графов
- Обзор существующих алгоритмов разбиения графов
- Исследование свойств графов вызовов
- Разработка алгоритма разбиения
- Реализация и тестирование алгоритма на реальных программах

Свойства графов вызовов программ

- Разреженный
- Ориентированный
- Ребра и узлы обладают большим числом параметров, которые можно трактовать как веса:
  - Размер кода функции
  - Частота вызовов или профиль
  - Оценка возможности встраивания
  - Принадлежность компоненте одной сильной связности
  - Использование общих глобальных переменных
  - . . .

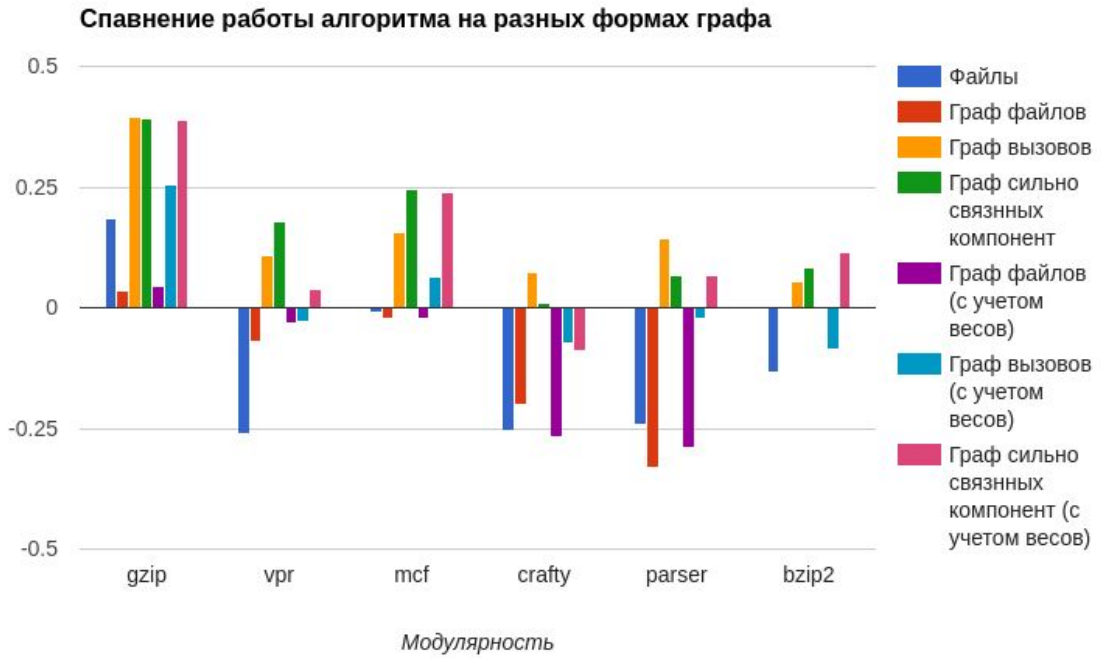
Пример графа вызовов



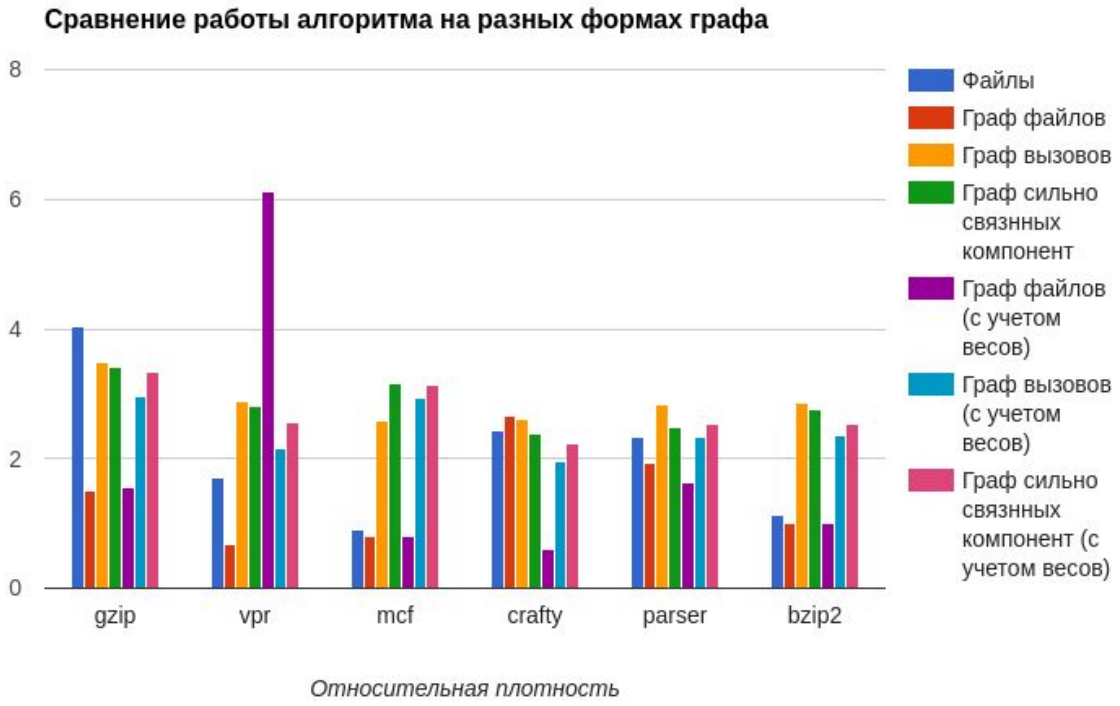
Что будем делить?

- Сам граф?
- Граф компонент сильной связности?
- Граф файлов?
- Может быть, сами файлы - это естественные “хорошие” кластеры?

Сравнение модулярности разбитых итеративным алгоритмом графов



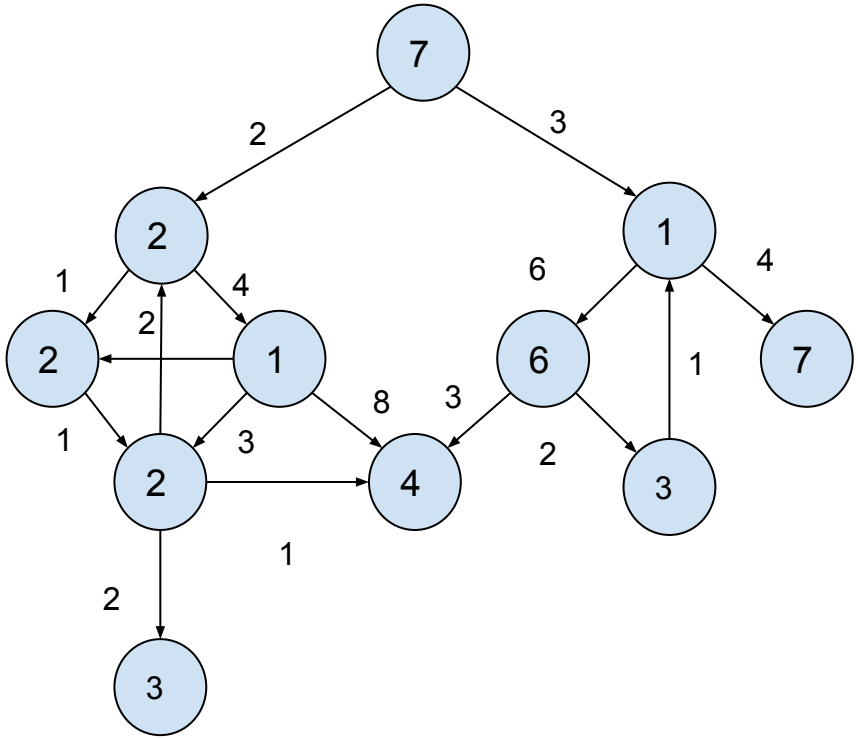
Сравнение относительной плотности



Горизонтальное масштабирование: постановка задачи

- Исследование критериев разбиения графов
- Обзор существующих алгоритмов разбиения графов
- Исследование свойств графов вызовов
- Разработка алгоритма разбиения
- Реализация и тестирование алгоритма на реальных программах

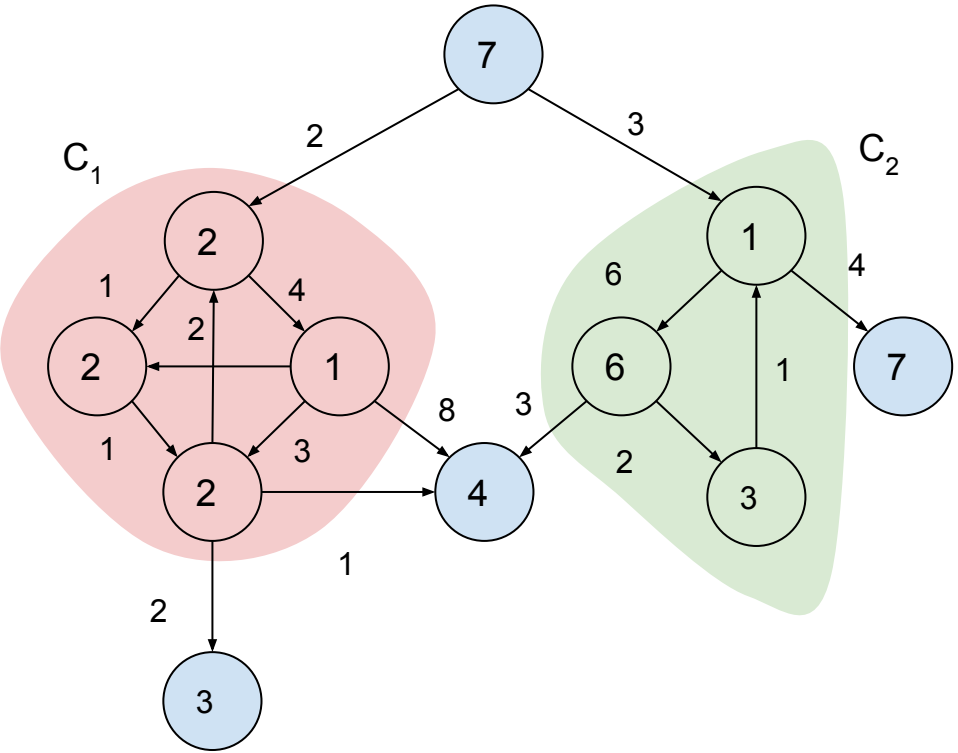
Описание алгоритма



Постановка задачи

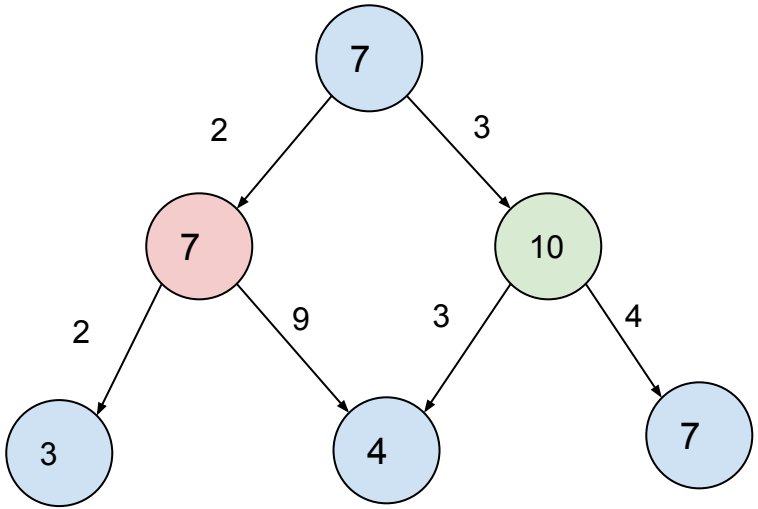
- Взвешенный ориентированный граф, |V| = n, |E| =m, вершины также могут иметь вес
- Нужно разбить на k кластеров

Описание алгоритма



Находим компоненты сильной связности (алгоритм Косарайю или Тарьяна)

Описание алгоритма



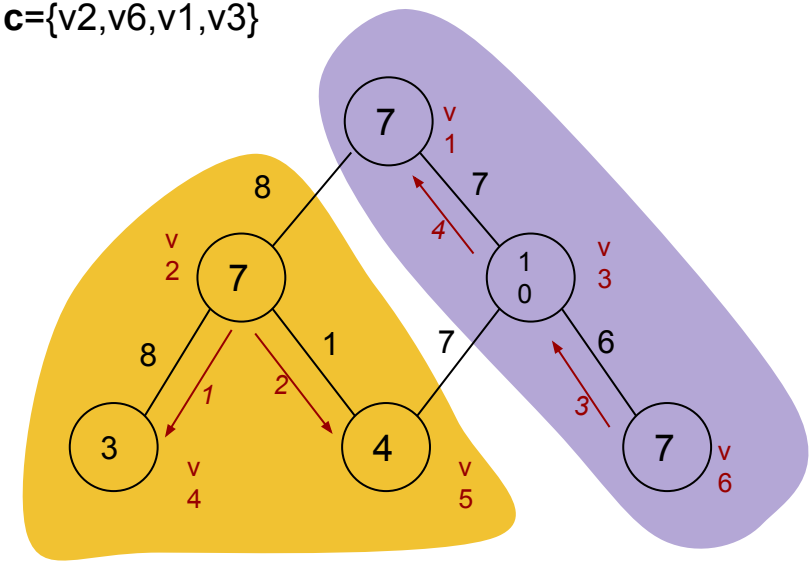
Схлопываем компоненты сильной связности в 1 вершину

Описание алгоритма

k=2
c_0={v1,v2,v3,v6}

d(v1,v2)=8, d(v1,v3)=7,
d(v1,v6)=11,
d(v2,v3)=8, d(v2,v6)=14,
d(v3,v6)=6

D(v1)=9, D(v2)=10, D(v3)=7, D(v6)=10
c={v2,v6,v1,v3}

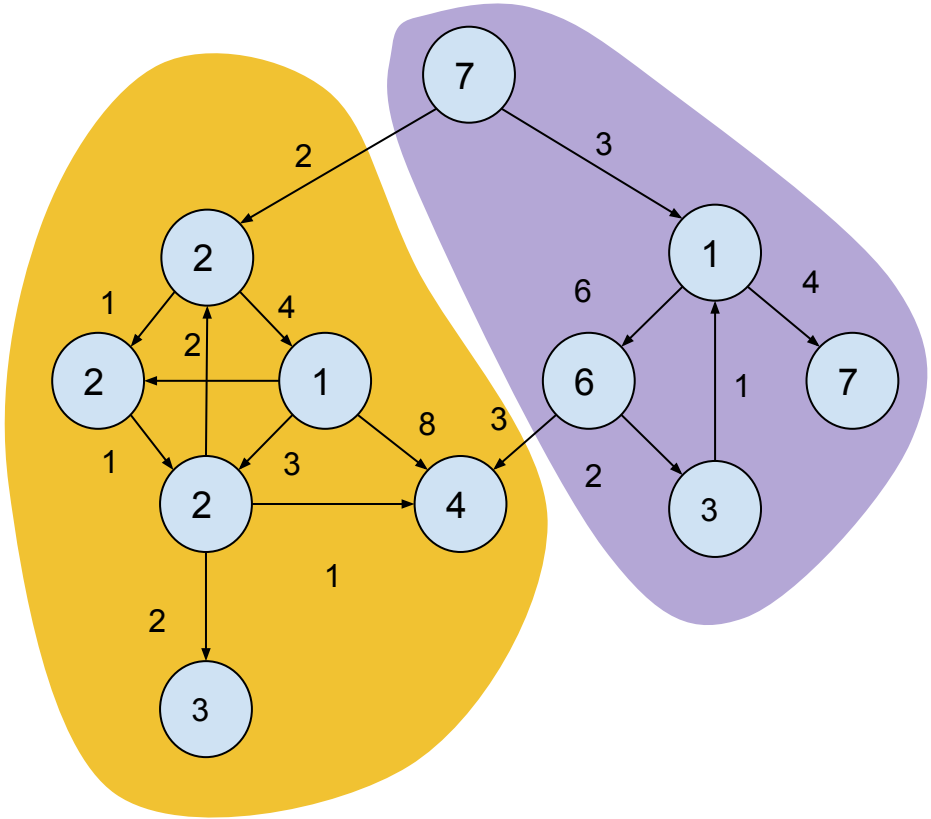


- Приводим к неориентированному виду
- Ищем 2k потенциальных “центров” с максимальным удельным весом

$$\omega(v)=w(v)\cdot\sum_{(v,u)\in E'} w(v,u)$$

- Вычисляем расстояния между центрами
- Для k максимально удаленных друг от друга центров набираем n/k ближайших соседей в кластер

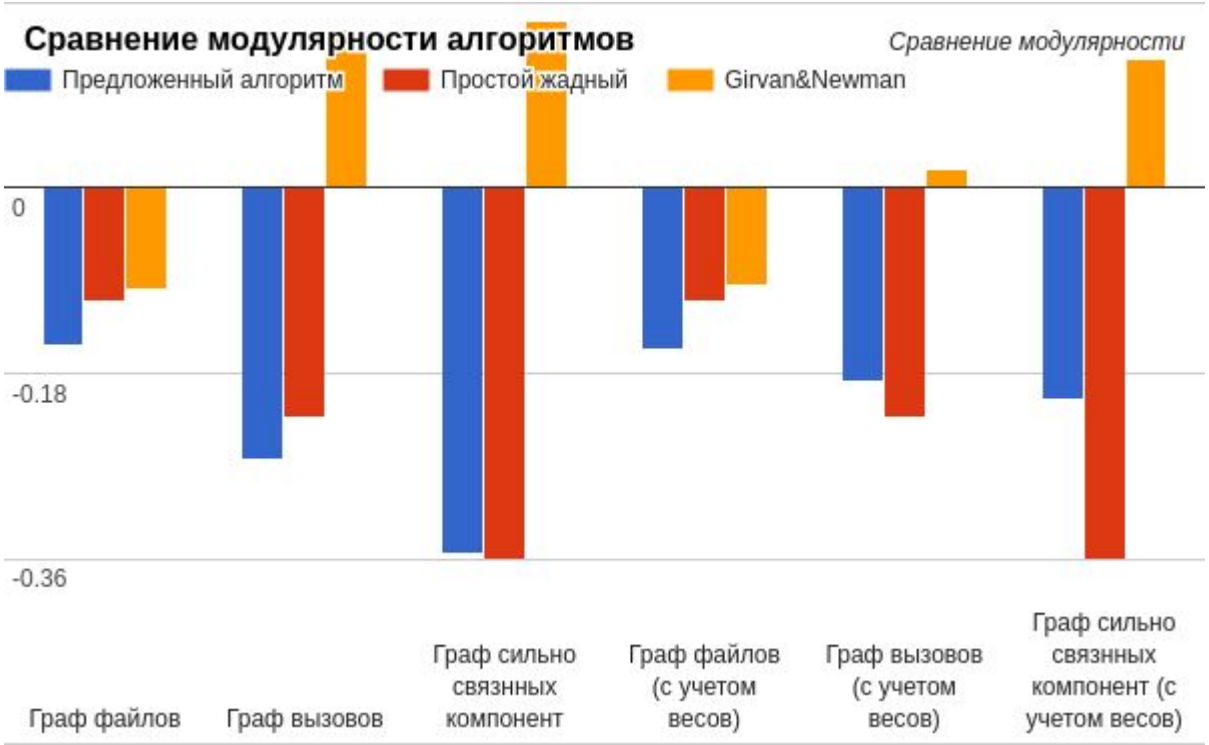
Описание алгоритма



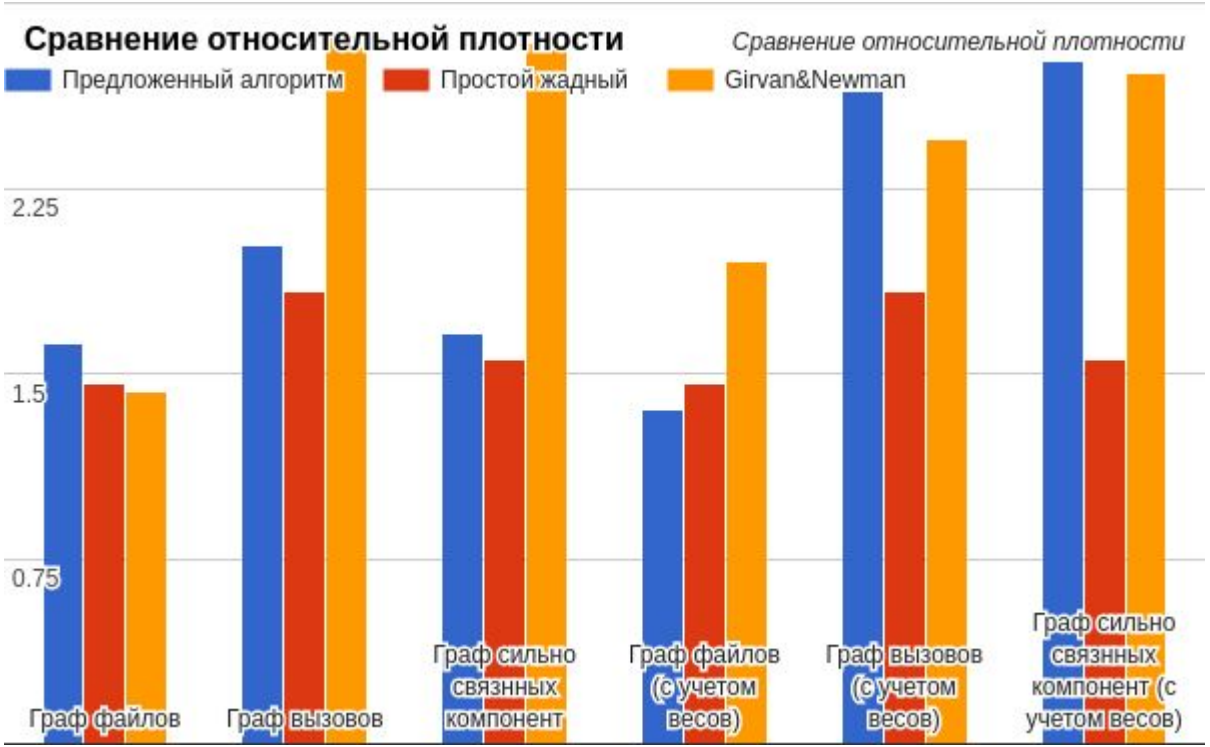
Горизонтальное масштабирование: постановка задачи

- Исследование критериев разбиения графов
- Обзор существующих алгоритмов разбиения графов
- Исследование свойств графов вызовов
- Разработка алгоритма разбиения
- Реализация и тестирование алгоритма на реальных программах

Измерение параметра модулярности алгоритма



Измерение параметра отн.плотности алгоритма



Результаты:

- Ускорение сборки на 4х потоках в режиме многопоточности -- 31.9%
- Замедление сборки программы в режиме ленивой загрузки -- в 3-5 раз в зависимости от ограничений
- Накладные расходы по времени на ленивую загрузку -- 0.2%
- Накладные расходы по памяти на ленивую загрузку -- 36.3%
- Увеличение объема биткода -- 6.6%
- Увеличение производительности программ с легковесной оптимизацией удаление мертвых глобальных переменных -- 0.5%

* Intel Core i7, 4 ядра, тесты SPEC2000 int, evas(expedite), coremark, clucene

Результаты

Работа на больших программах (Firefox, LibreOffice)

Проблемы работы LTO с большими приложениями

- Состоят из компонент, написанных на разных языках
- Содержат в том числе ассемблерный код, с которым возникают проблемы при компоновке с биткодом
- Зависят от стандартных библиотек, несовместимых с LLVM 3.4 (баги в Clang, связанные с поддержкой новых стандартов C++)
- Зависят от стандартных утилит, содержащих баги (Binutils)
- Содержат баги в скриптах конфигурации
- ...

Сравнение с ThinLTO

- Одна идея, разработанная параллельно и независимо
- Похожая архитектура
- Разные подходы к разбиению графа вызовов: использование линейного жадного алгоритма, принимающий к сведению только доступность по связыванию (linkage) функции

Спасибо за внимание!