ISP RAS

Ускорение оптимизации программ во время связывания

Долгорукова К.Ю.
Аришин С.В.
ИСП РАН

Содержание доклада

- Введение: LTO (Link-time optimization)
- Горизонтальное масштабирование системы LTO на основе LLVM
- Легковесные оптимизации времени связывания для системы LTO на основе LLVM

Схема сборки программы из исходного кода без LTO

ISP RAS

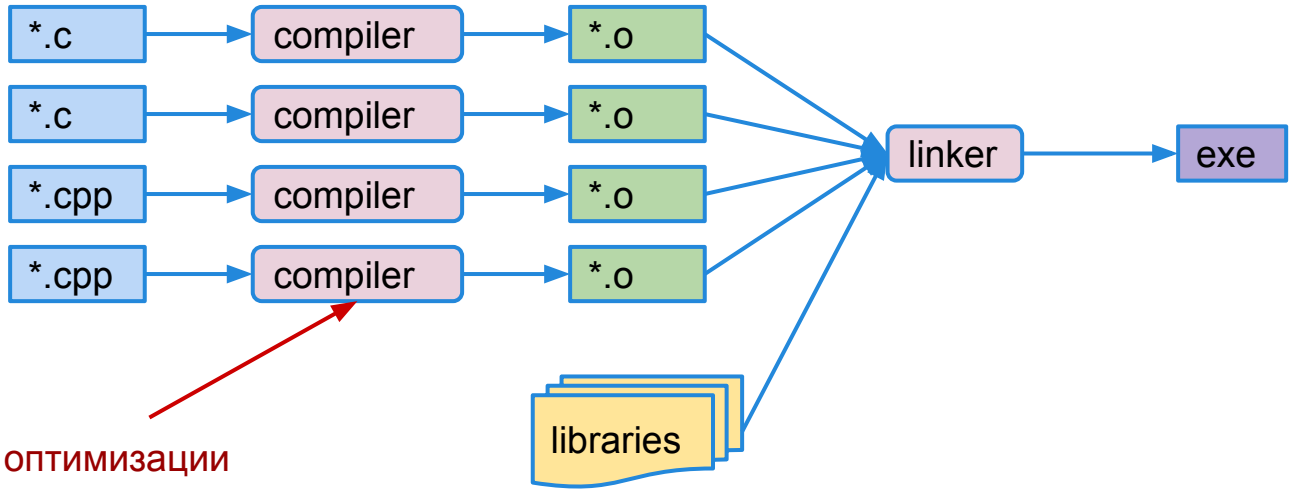
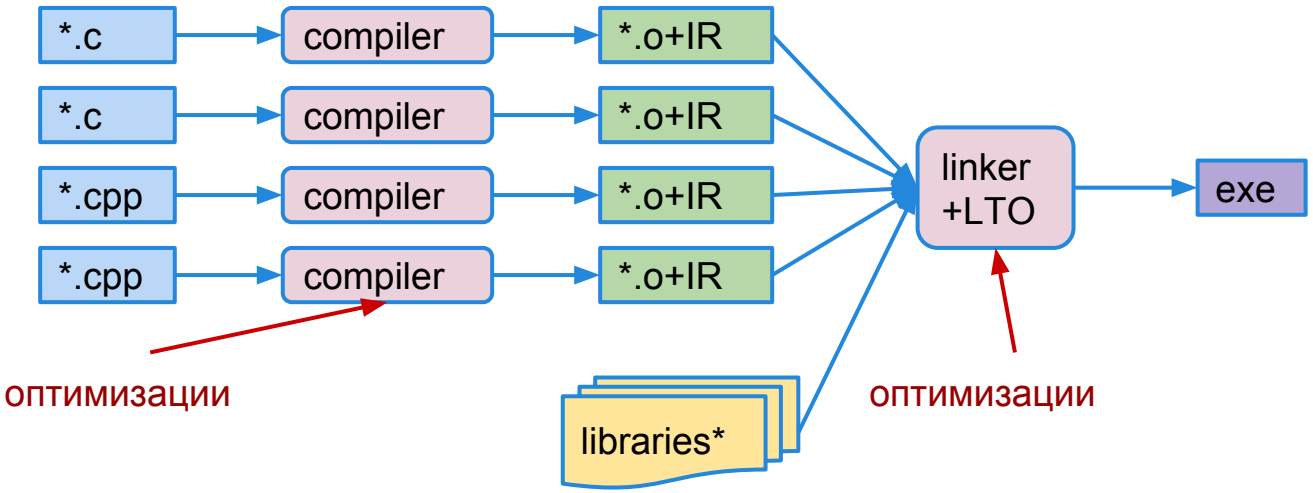


Схема сборки программы из исходного кода с LTO

ISP RAS



Оптимизации времени связывания (Link-Time Optimization)

ISP RAS

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

Главная проблема LTO

А что, если пользователь хочет собирать с LTO

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

Главная проблема LTO

А что, если пользователь хочет собирать с LTO

- 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
- ...

Главная проблема LTO

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

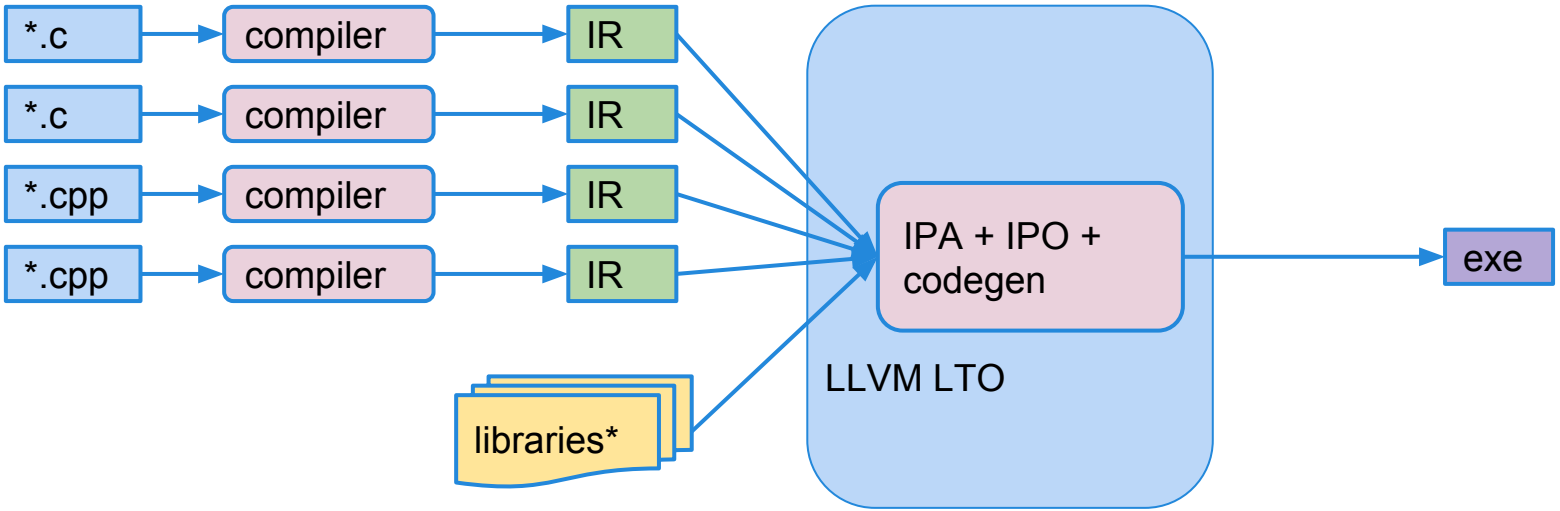
- Firefox: 11-26 минут, 6-34 Гб ОЗУ [1]
- LibreOffice: 61-68 минут, 8-14 Гбайт ОЗУ [2]

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

2 http://hubicka.blogspot.ru/2014/09/linktime-optimization-in-gcc-part-3.html

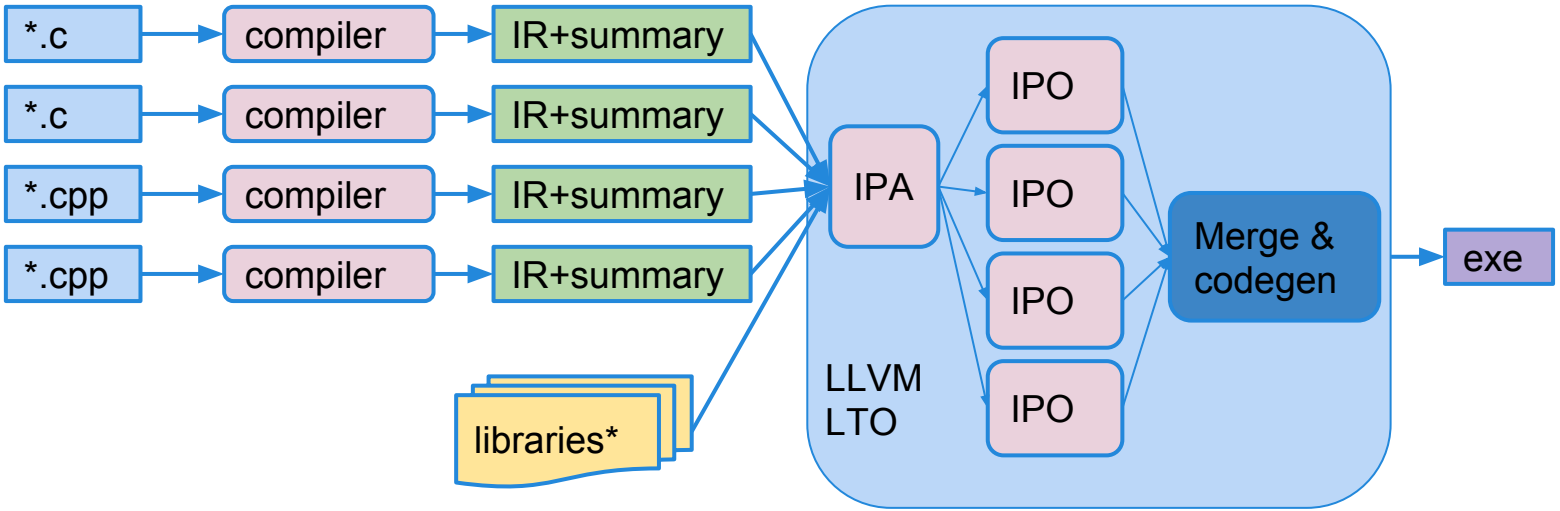
Горизонтальное масштабирование LLVM

ISP RAS



Горизонтальное масштабирование LLVM

ISP RAS



Горизонтальное масштабирование

Разбиение промежуточного кода (IR)

Горизонтальное масштабирование

Разбиение промежуточного кода (IR)

=>

Разбиение графа вызовов

Разбиение графов вызовов

- NP-полная задача
- Выбор алгоритма зависит от свойств графов

Проблема разбиения графа вызовов

ISP RAS

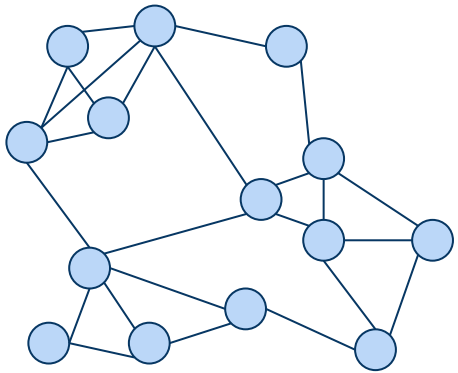
- Каковы критерии разбиения графов?
- Что представляет собой граф вызовов?
- Какими характеристиками должен обладать алгоритм разбиения?

Проблема разбиения графа вызовов

ISP RAS

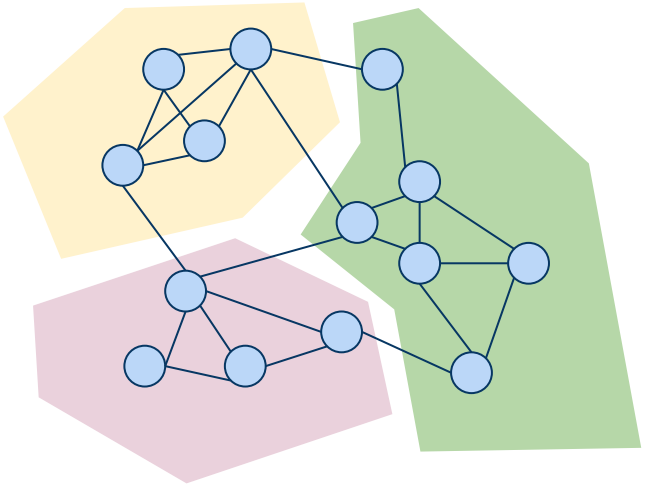
- Каковы критерии разбиения графов?
- Что представляет собой граф вызовов?
- Какими характеристиками должен обладать алгоритм разбиения?

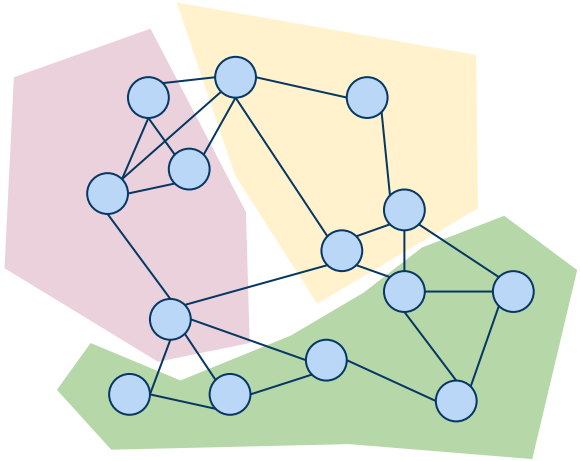
Критерии разбиения графа



Пример:
Попробуем разбить граф на 3 подграфа

Критерии разбиения графа





О задаче разбиения (кластеризации) графа

ISP RAS

- 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 подмножества C, что внутренняя плотность каждого

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

будет превосходить внешнюю

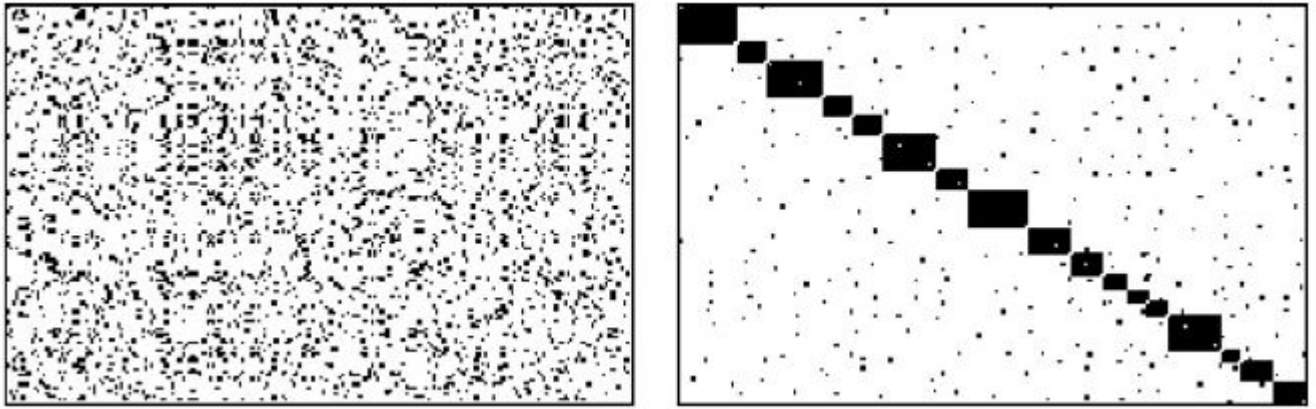
$$\delta_{\text{ext}}(G \mid \mathcal{C}_1,\ldots,\mathcal{C}_k) = \frac{\left|\left\{\{v,u\} \mid v \in \mathcal{C}_i, u \in \mathcal{C}_j, i \neq j\right\}\right|}{n(n-1) - \sum_{\ell=1}^{k} (|\mathcal{C}_\ell|\,(|\mathcal{C}_\ell|-1))}.$$

О задаче разбиения (кластеризации) графа

ISP RAS

- 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

$Q = \Sigma_i(e_{ii} - a_i^2)$

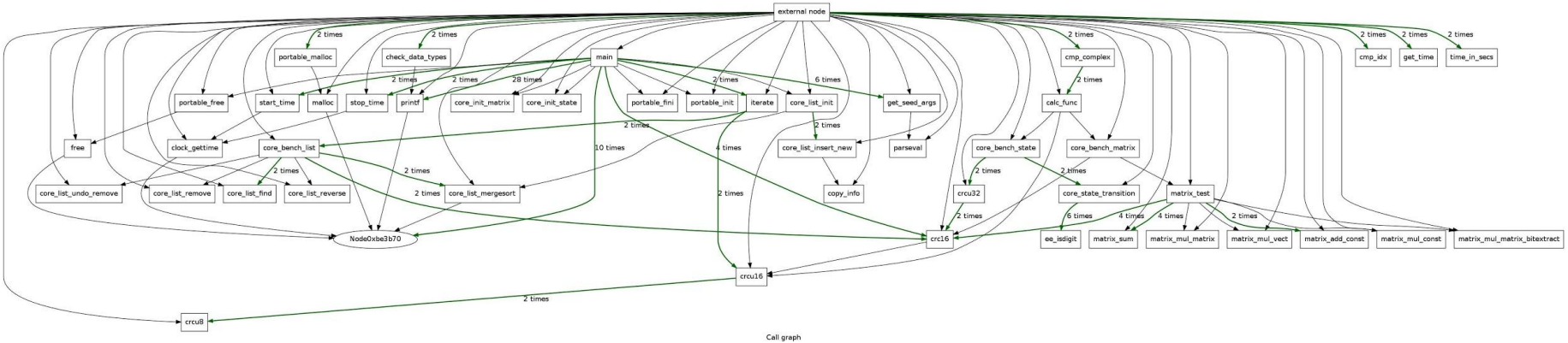
где $a_i = \Sigma_j e_{ij}$ для матрицы относительных степеней кластеров, E = {e_{ij}}, где для кластеров C_i и C_j e_{ij} = $|(u,v)|/m$, $u \in C_i$, $v \in C_j$, а $e_{ii} = deg(C_i)/m$

Проблема разбиения графа вызовов

ISP RAS

- Каковы критерии разбиения графов?
- Что представляет собой граф вызовов?
- Какими характеристиками должен обладать алгоритм разбиения?

Пример графа вызовов



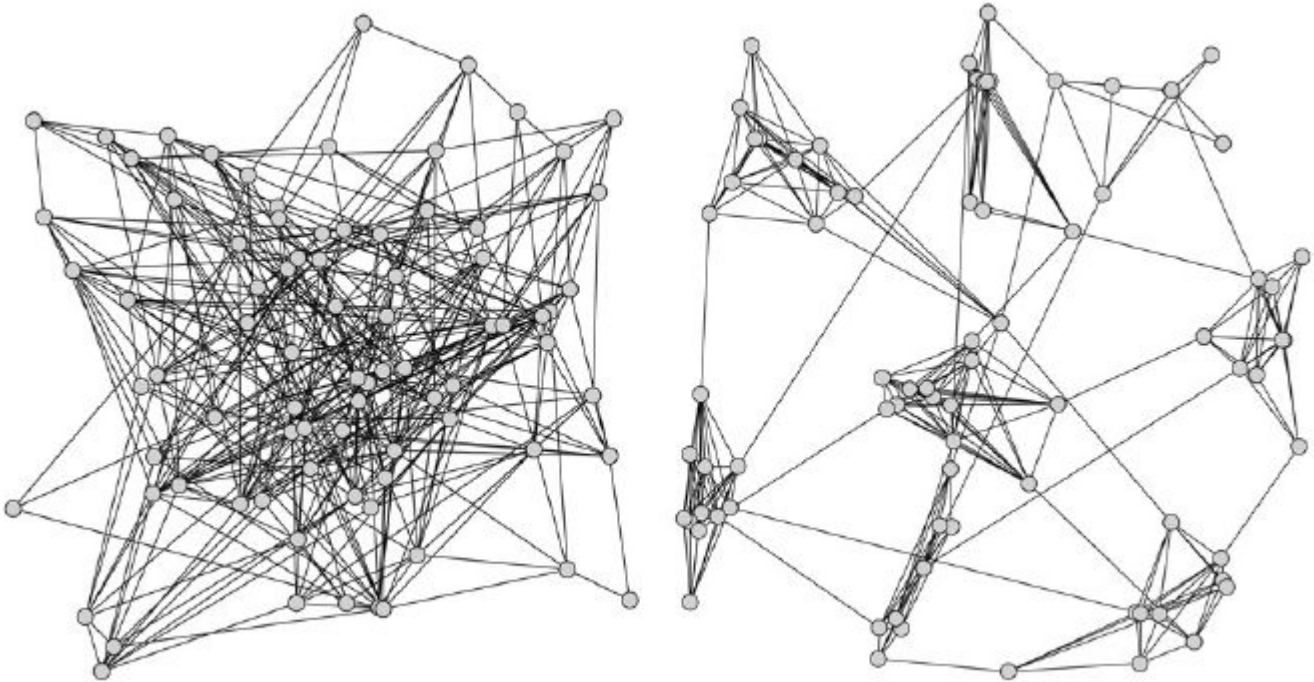
Свойства графов вызовов программ

ISP RAS

- 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
  - . . .

Анализ кластерных свойств графов вызовов программ

ISP RAS



Анализ кластерных свойств графов вызовов программ

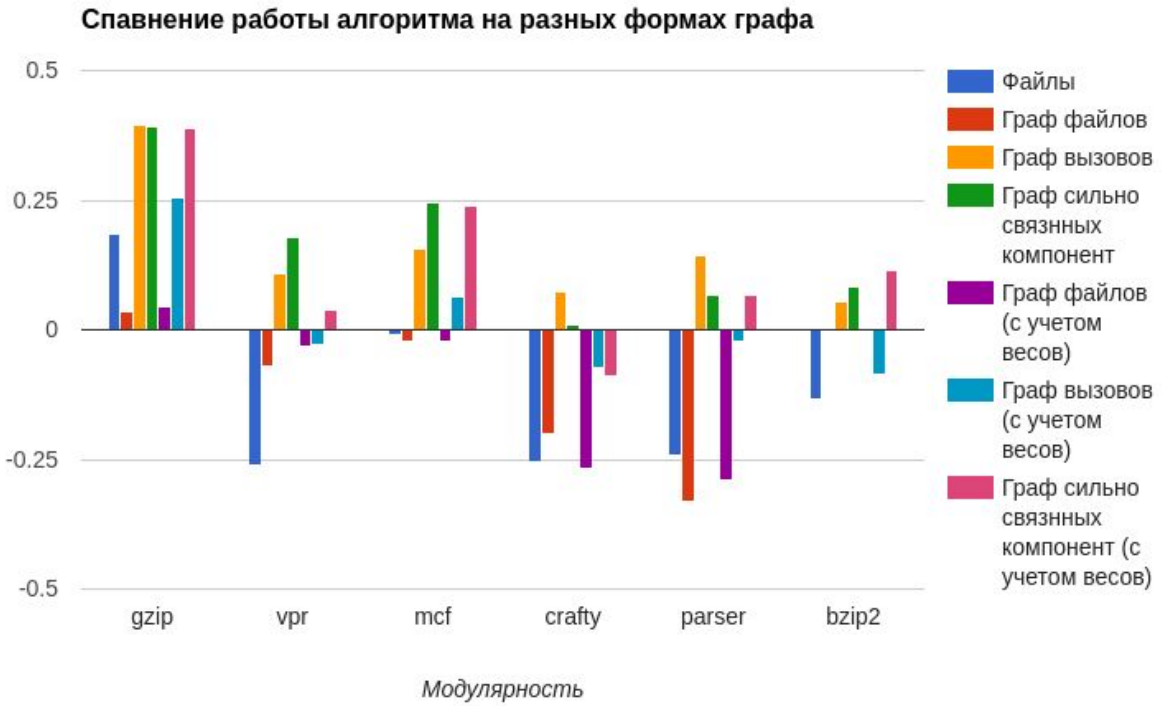
ISP RAS

Разбиение графов можно проводить, руководствуясь априори существующими “кластерами”

- Компоненты сильной связности
- Файлы

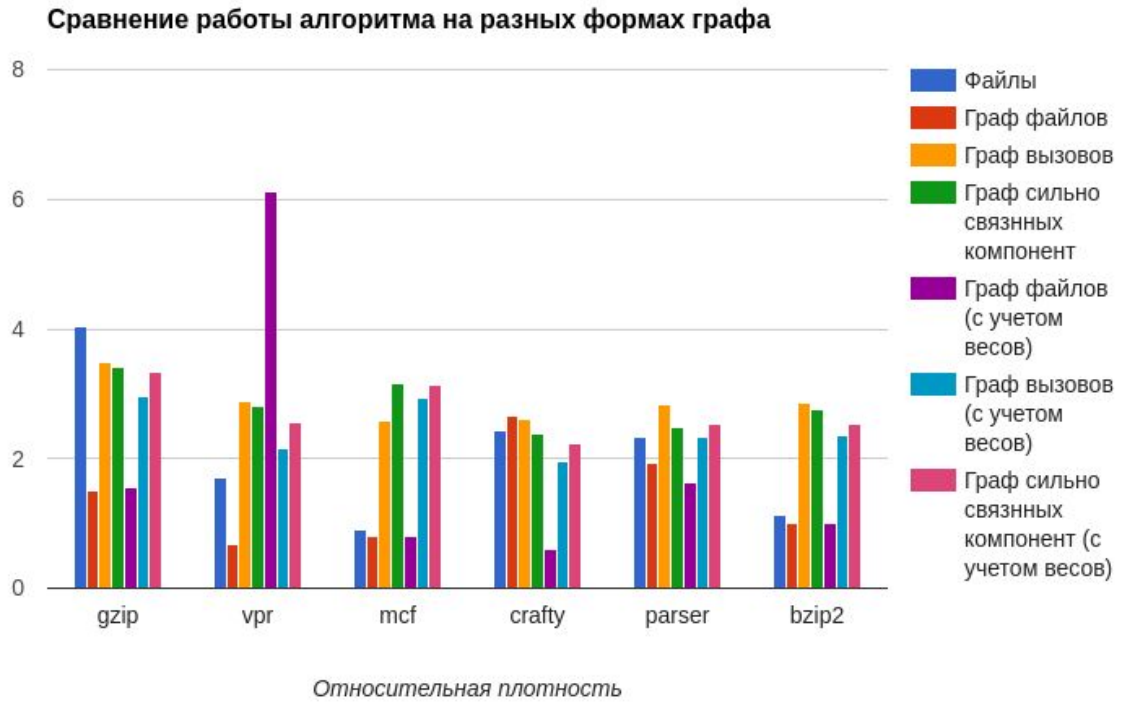
Сравнение модулярности разбитых итеративным алгоритмом графов

ISP RAS



Сравнение относительной плотности

ISP RAS



Проблема разбиения графа вызовов

ISP RAS

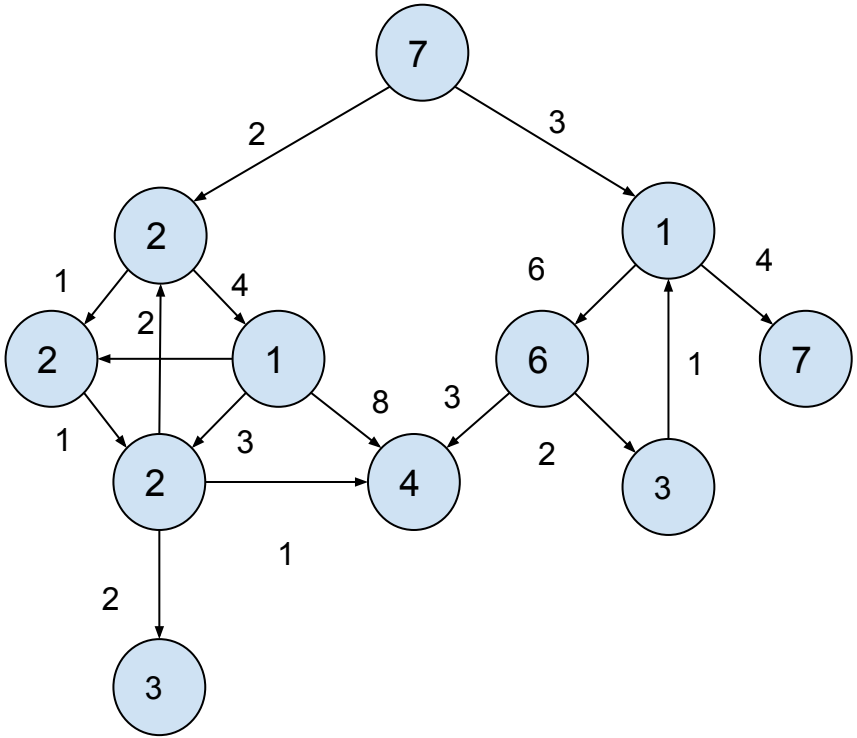
- Каковы критерии разбиения графов?
- Что представляет собой граф вызовов?
- Какими характеристиками должен обладать алгоритм разбиения?

Проблема разбиения графа вызовов

ISP RAS

- Каковы критерии разбиения графов?
- Что представляет собой граф вызовов?
- Какими характеристиками должен обладать алгоритм разбиения?
  - Быстрый
  - На выходе - кластеры приблизительно одинакового размера
  - Находит относительно “хорошее” разбиение

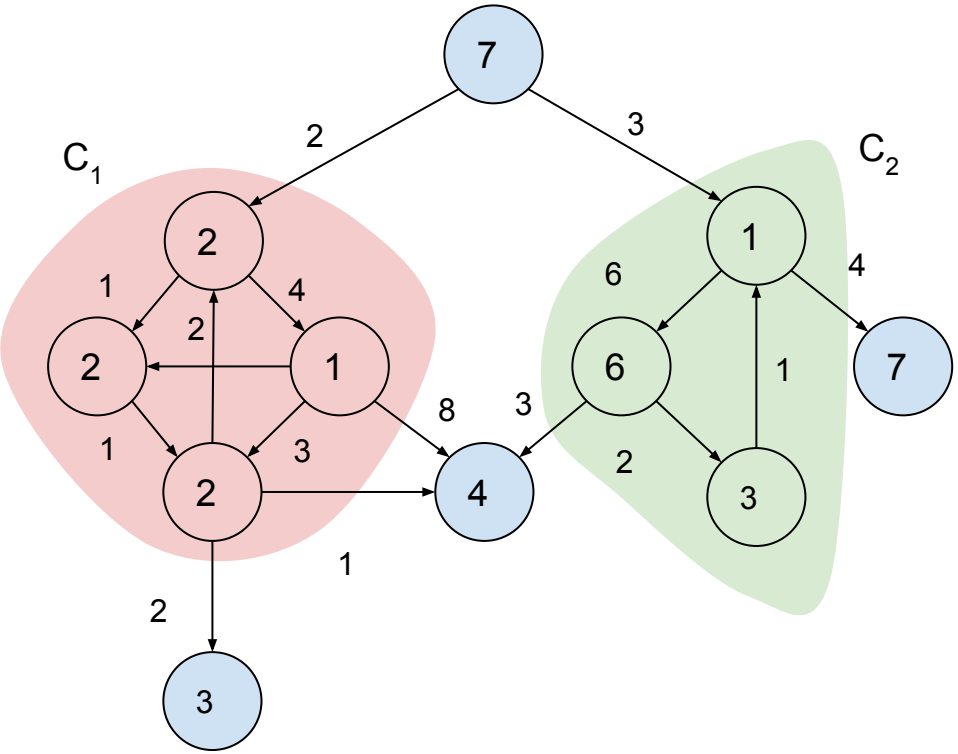
Описание алгоритма



Постановка задачи

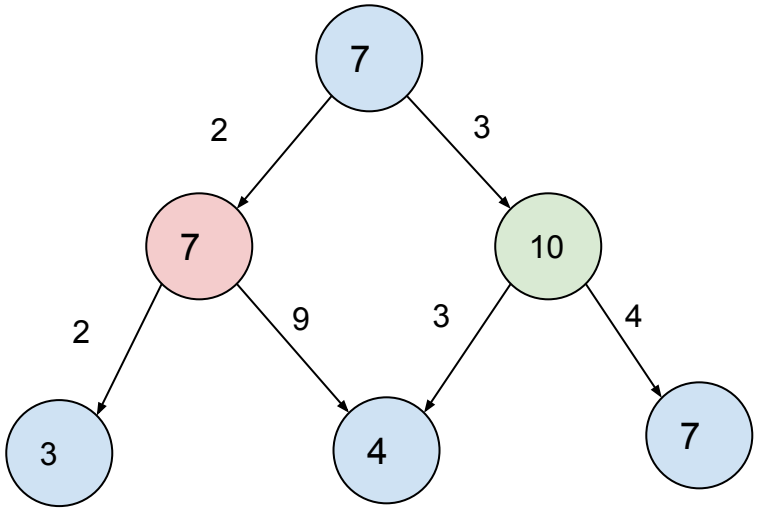
- Взвешенный ориентированный граф, |V| = n, |E| =m, вершины также могут иметь вес
- Нужно разбить на k кластеров

Описание алгоритма



Находим компоненты сильной связности (алгоритм Косарайю или Тарьяна)

Описание алгоритма



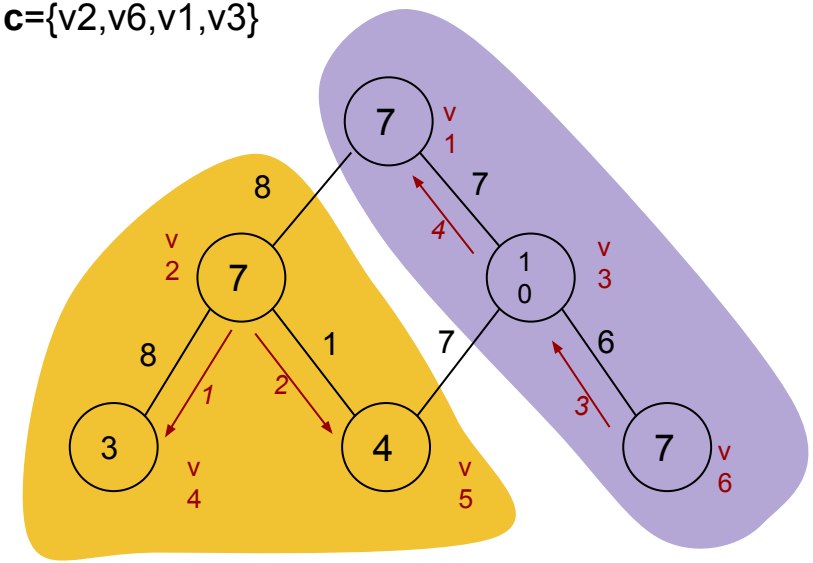
Схлопываем компоненты сильной связности в 1 вершину

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}

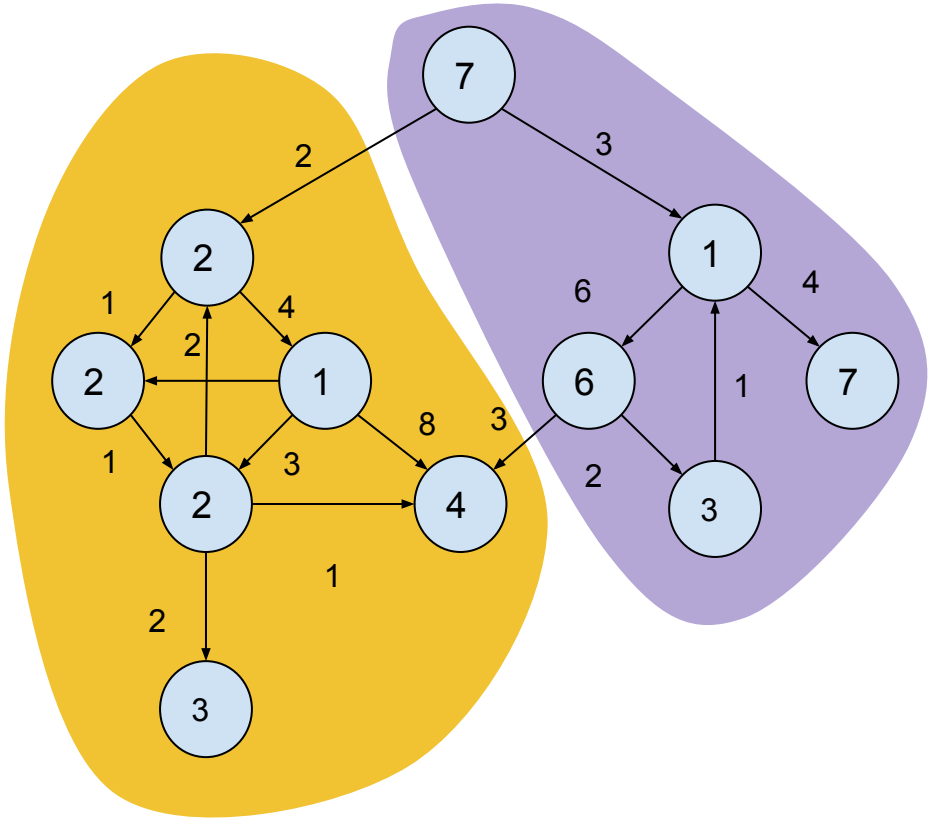


- Приводим к неориентированному виду
- Ищем 2k потенциальных “центров” с максимальным удельным весом

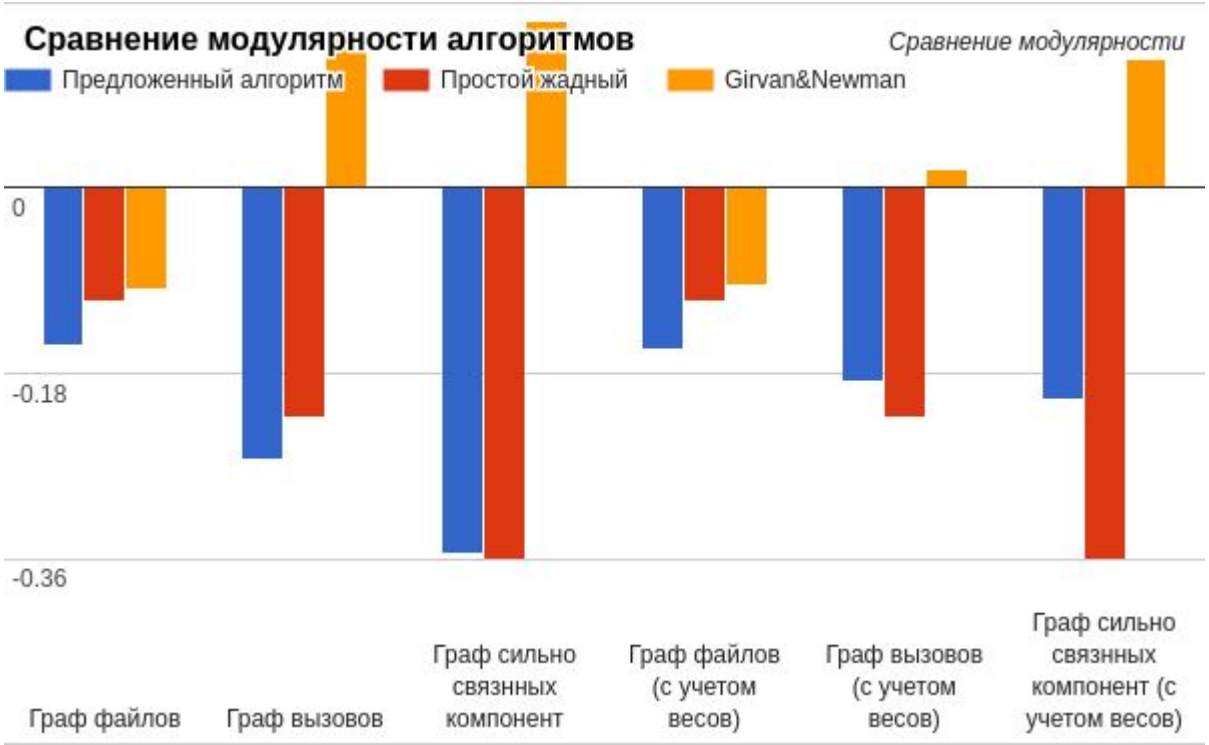
$$\omega(v)=w(v)\cdot\sum_{(v,u)\in E'} w(v,u)$$

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

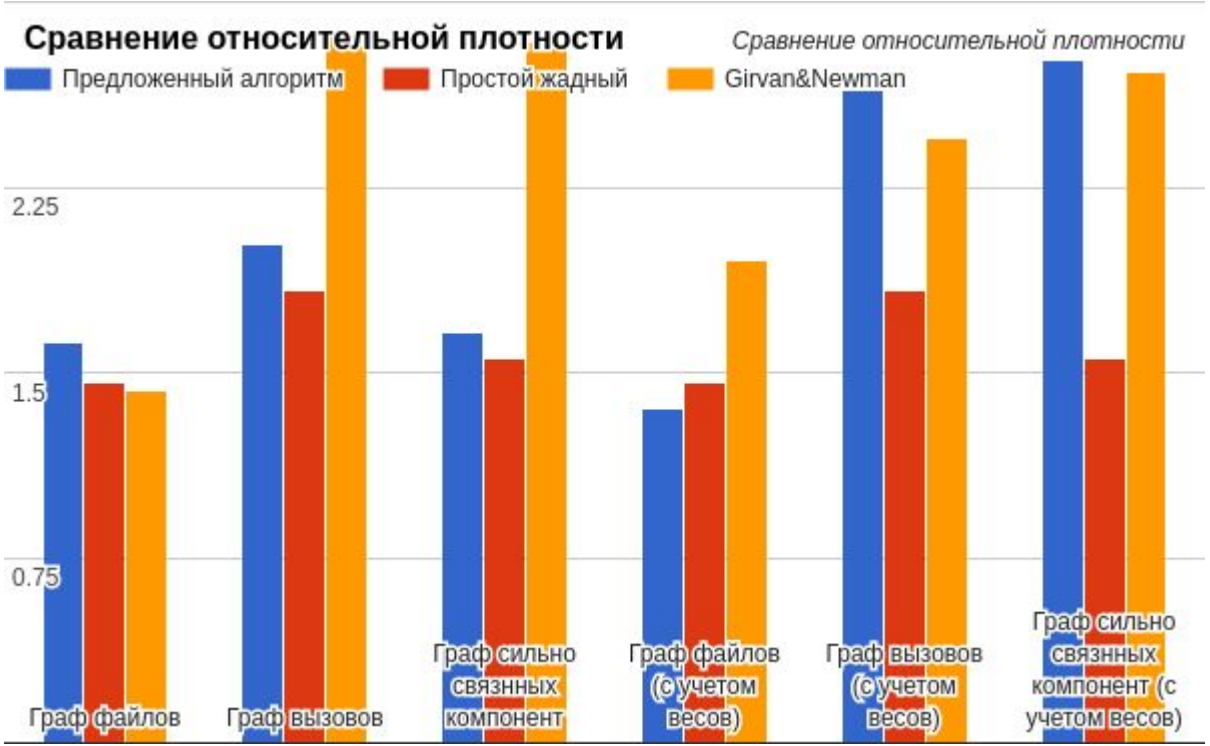
Описание алгоритма



Измерение параметра модулярности алгоритма



Измерение параметра отн. плотности алгоритма



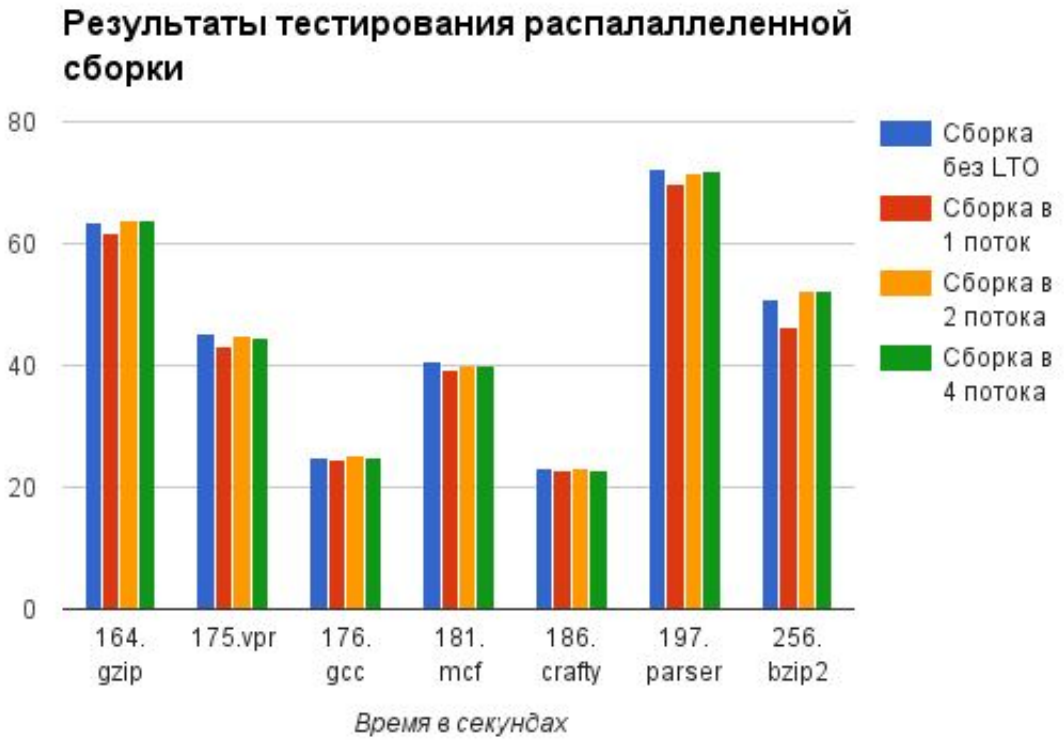
Результаты сборки в несколько потоков: время работы

ISP RAS

Тест	Сборка без распараллеливания, с	Сборка с распараллеливанием в 4 потока, с	Ускорение, %
164.gzip	0.28	0.26	7.28
175.vpr	1.06	0.71	32.9
176.gcc	13.88	7.97	42.60
181.mcf	0.13	0.09	31.44
186.crafty	1.5	0.85	43.14
197.parser	1.07	0.78	27.66
256.bzip2	0.4	0.24	38.67
Суммарное время	**18.31**	**10.89**	**31.9**

Результаты сборки в несколько потоков: производительность

ISP RAS



Легковесные оптимизации времени связывания

ISP RAS

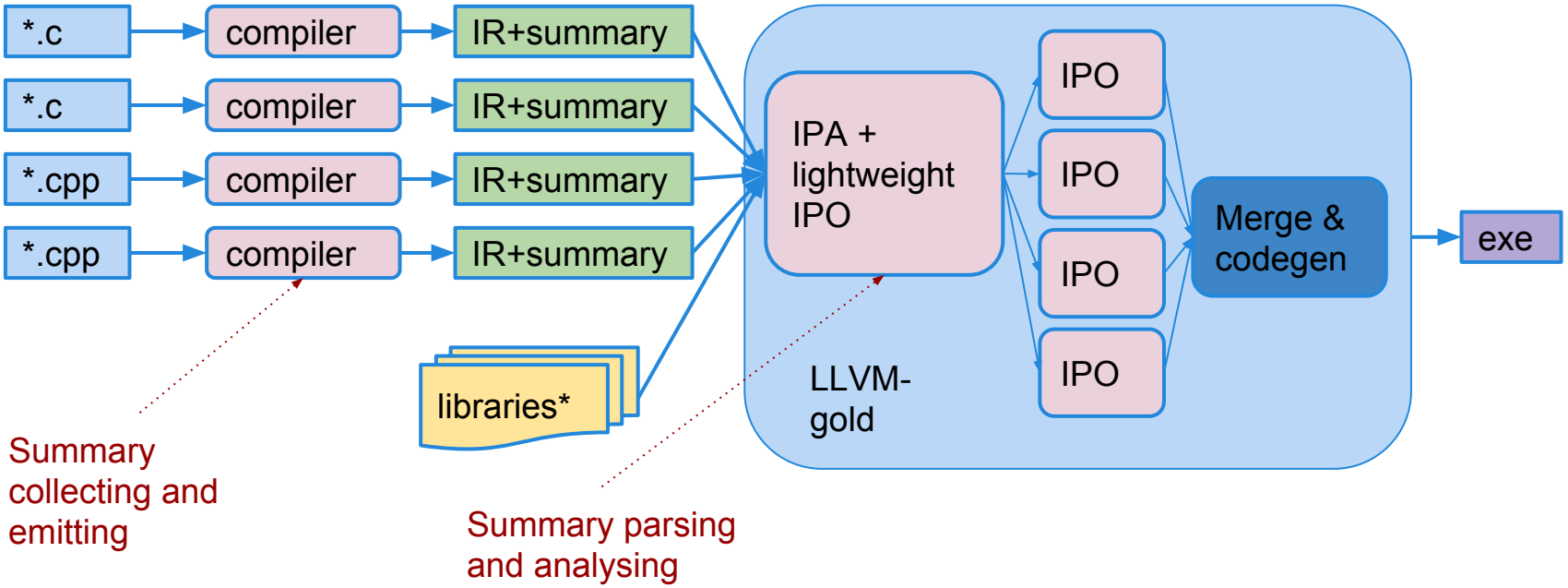
Легковесные оптимизации времени связывания

ISP RAS

Необходимость

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

Оптимизации по аннотациям



Легковесная оптимизация удаления глобальных переменных

ISP RAS

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

Результаты: оптимизация удаления глобальных переменных

ISP RAS

Название теста	Без оптимизации	С оптимизацией	Прирост
164.gzip	78,8	77,9	1,14%
175.vpr	58,4	58,3	0,17%
176.gcc	33,4	32,8	1,79%
181.mcf	86,9	86,6	0,34%
197.parser	94,4	94,2	0,21%
256.bzip2	69,3	69,7	-0,57%
	421,2	**419,5**	**0,51%**

Спасибо за внимание!