ISP RAS

Разработка и реализация метода масштабирования по памяти для систем межмодульных оптимизаций и статического анализа на основе LLVM

Долгорукова К.Ю.
ИСП РАН

Схема сборки программы из исходного кода

ISP RAS

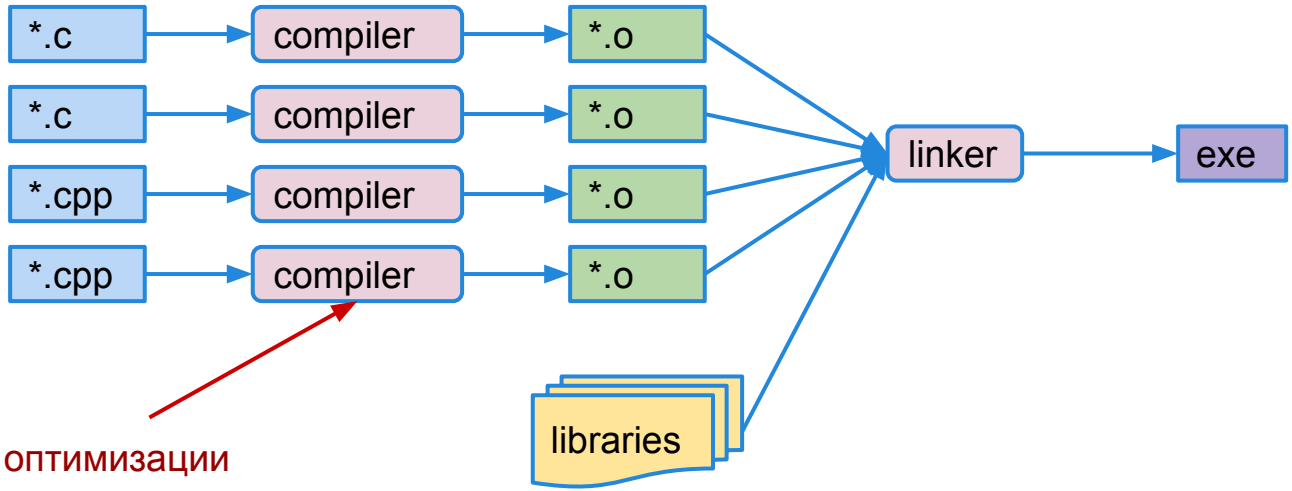
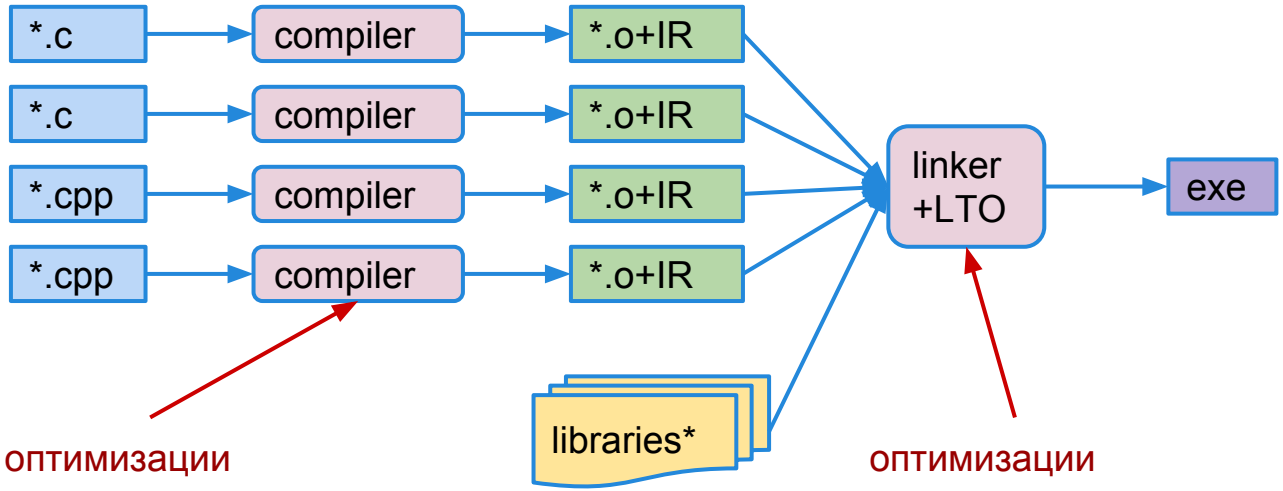


Схема сборки программы из исходного кода с LTO

ISP RAS



* Библиотеки тоже могут содержать промежуточное представление

Оптимизации времени связывания (Link-Time Optimization)

ISP RAS

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

Оптимизации времени связывания (Link-Time Optimizations)

ISP RAS

Увеличивается эффективность межпроцедурных оптимизаций

- Межпроцедурное распространение констант
- Межпроцедурное удаление мертвого кода
- Межпроцедурное продвижение аргументов
- Встраивание (особенно эффективно с профилем)
- и т.д.

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

Статический анализ

- Проводится на всей программе
- Контекстно-чувствителен, либо манипулирует большим объёмом мета-информации

Статический анализ

- Проводится на всей программе
- Контекстно-чувствителен, либо манипулирует большим объёмом мета-информации

Требует много памяти

=> Проблемы те же, что и у LTO, но в бóльших масштабах

Масштабируемость

Масшта́бируемость (англ. *scalability*) — в электронике и информатике означает способность системы, сети или процесса справляться с увеличением рабочей нагрузки (увеличивать свою производительность) при добавлении ресурсов (обычно аппаратных).[1]

An algorithm, design, networking protocol, program, or other system is said to ***scale*** if it is suitably efficient and practical when applied to large situations.[2]

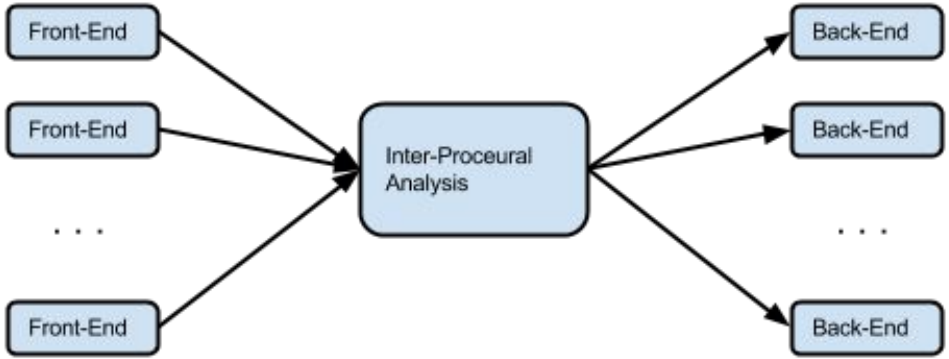
[1] Русская википедия
[2] Английская википедия

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

Масштабируемость

- Разбиение этапа LTO на независимые задачи, которые можно выполнять параллельно
- Схема масштабирования систем LTO, традиционный подход



Масштабируемость по памяти

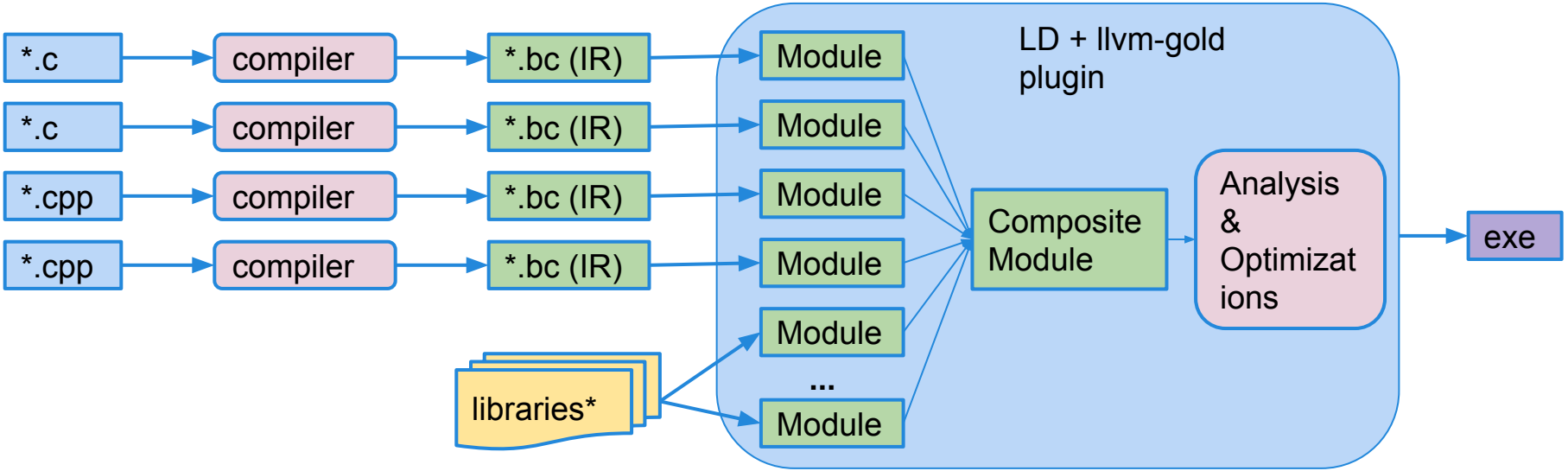
- Подходы к масштабированию в существующих системах
  - Диспетчер с отгрузкой неиспользуемых участков кода на диск (HLO)
  - Работа анализа только на аннотациях (GCC)
  - Совмещение run-time компоненты с анализом (Google LIPO)

Как масштабируется LTO в LLVM

- Промежуточное представление LLVM весьма легковесно
- И...

… это всё...

Как происходит LTO в LLVM



Предлагаемое решение

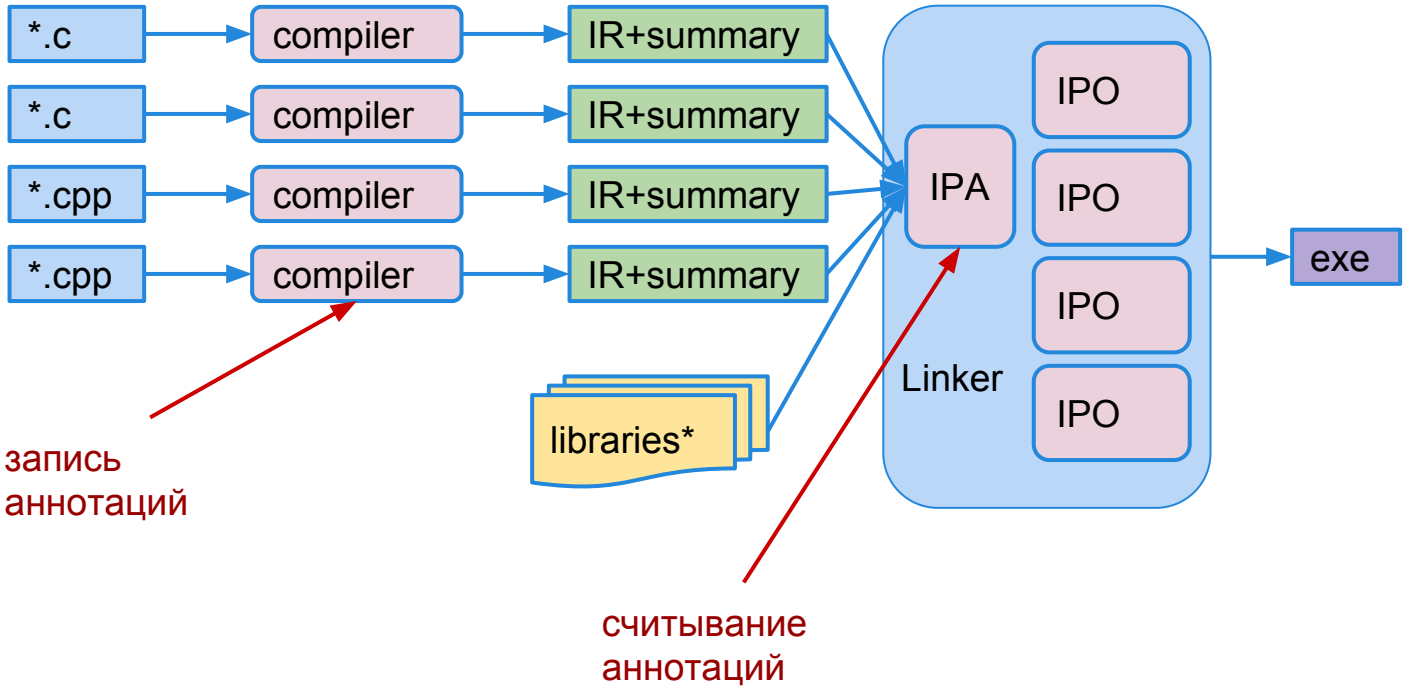
- Разделить межпроцедурные анализ и оптимизации
- Ленивая загрузка участков кода (процедур)
- Выгрузка участков кода (процедур)

Разделение анализа и оптимизаций

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

Разделение анализа и оптимизаций

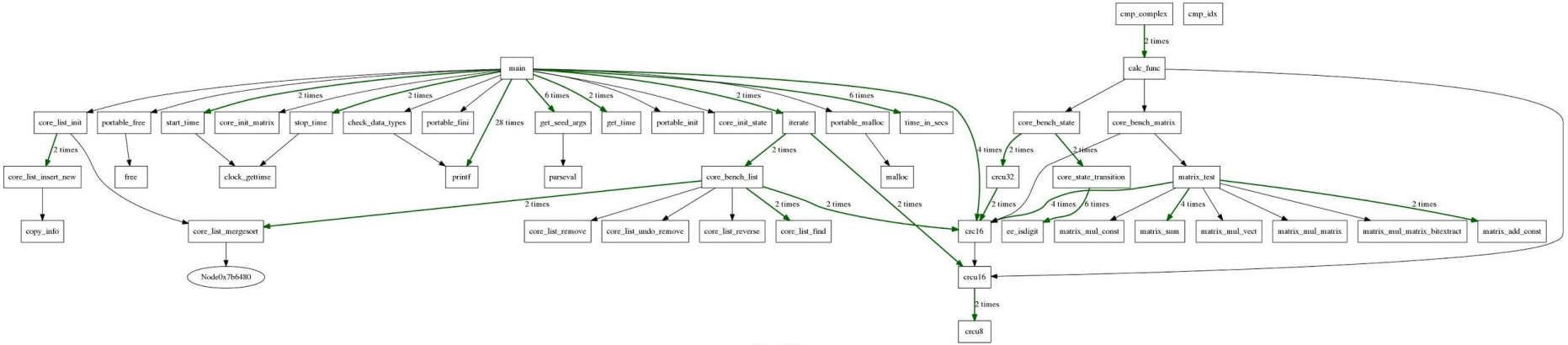


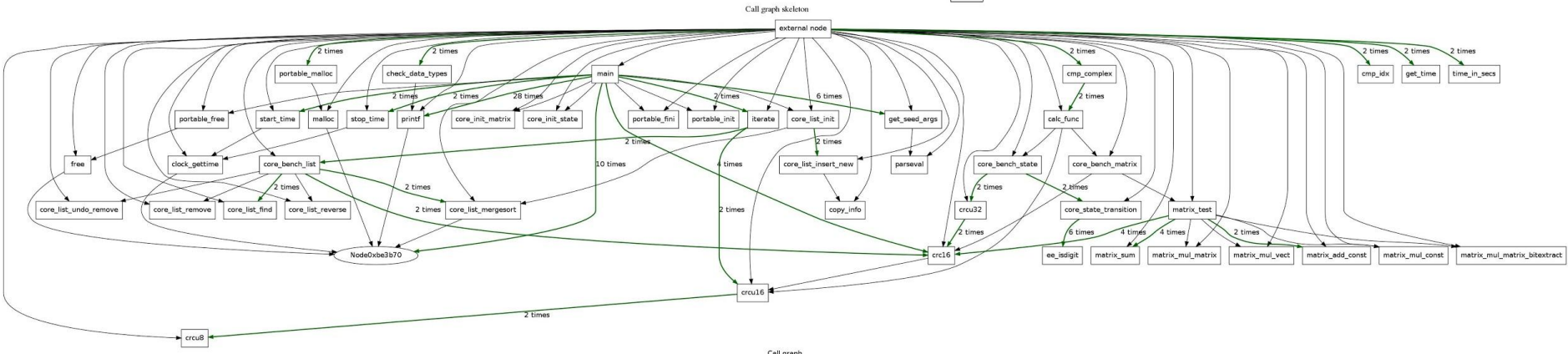
Разделение анализа и оптимизаций

Пример разделения анализа: построение графа вызовов

- Аннотируем процедуры информацией о вызовах в теле во время генерации IR
- Во время компоновки считываем только заголовки процедур, информацию о типах и глобальных переменных и аннотации, пропуская тела процедур
- Строим граф вызовов на основе аннотаций и объявлений

Граф вызовов





Ленивая загрузка участков кода

Ленивая загрузка участков кода

Разделение загрузки кода из файлов с IR на этапы

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

Ленивая загрузка участков кода

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

Ленивая загрузка участков кода

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

Ленивая загрузка участков кода

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
- ...

Ленивая загрузка участков кода

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

Ленивая загрузка участков кода

Предварительная компоновка

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

Ленивая загрузка участков кода

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

Ленивая загрузка участков кода

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

Предварительные результаты

Увеличение объема файлов биткода

Название теста	Размер файлов с биткодом в байтах при обычной сборке	Размер файлов с биткодом в байтах при сборке с аннотациями	Увеличение размера кода в процентах
gcc	2797940	2970212	6.16
gzip	89156	96808	8.58
vpr	303208	318008	4.88
mcf	43863	45656	4.09
crafty	571744	583700	2.09
parser	312748	377888	20.83
bzip2	86516	92292	6.68
Суммарное значение	**4205175**	**4484564**	**6.64**

Предварительные результаты

Временные издержки на ленивую загрузку

Название теста	Время сборки в секундах	Время ленивой сборки в секундах	Увеличение времени в процентах
gzip	0.2993	0.3001	0.27
vpr	0.9942	0.996	0.18
gcc	13.0642	13.0932	0.22
mcf	0.1164	0.1174	0.86
crafty	1.4256	1.4185	-0.50
parser	1.0047	1.0176	1.28
bzip2	0.3827	0.385	0.60
Суммарное значение	**17.29**	**17.33**	**0.24**

Предварительные результаты

Накладные расходы по памяти на ленивую загрузку

Название теста	Пик потребления при обычной сборке, Кб	Пик потребления при ленивой сборке, Кб	Накладные расходы, %	Среднее потребление при обычной сборке, Кб	Среднее потребление при ленивой сборке, Кб	Накладные расходы, %
gzip	324	404	24.69	177	269	51.98
vpr	18720	21652	15.66	4484	4765	6.27
gcc	168992	212468	25.73	94811	132811	40.08
mcf	n/a	n/a	n/a	n/a	n/a	n/a
crafty	23328	28700	23.03	6287	7319	16.41
parser	20296	25264	24.48	4924	5742	16.61
bzip2	n/a	n/a	n/a	n/a	n/a	n/a
Суммарное значение	**231660**	**288488**	**24.53**	**110683**	**150906**	**36.34**

Дальнейшие планы

- Уменьшение издержек по памяти
- Менеджер памяти для умной выгрузки тел функций

Спасибо за внимание!