Разработка и реализация алгоритмов сокращения размера Java-приложений в закрытой модели для встраиваемых систем

Артур Пилипенко

Диссертация на соискание учёной степени
кандидата технических наук

Научный руководитель:
к. ф.-м. н., доцент
Кияев Владимир Ильич

Предметная область

- Реализация Java-платформы для встраиваемых устройств
- Целевые устройства ограничены в ресурсах, особенно в размере памяти

Закрытая модель

- Набор приложений, исполняемых на устройстве, заранее определен
- Платформа может быть специализирована для заданного приложения
- Специализация сокращает аппаратные требования платформы

Специализации

- Понижение избыточности
  - Удаление возможностей платформы, не используемых приложением
- Специализация набора инструкций
  - Нестандартное, но более эффективное кодирование программы

Понижение избыточности

- Удаление неиспользуемых методов, полей и классов
- Консервативный анализ зависимостей начиная с точек входа в приложение
  - Анализ достижимости методов и удаление недостижимых
  - Удаление неиспользуемых полей
  - Удаление неиспользуемых классов

Раздельная инициализация

- Механизм сокращения размера и ускорения запуска приложений
  - CLDC, Squawk, Android, IBM J9, TakaTuka
- Инициализированное состояние ВМ сохраняется в виде загрузочного образа и используется при запуске приложения

Понижение избыточности

- Понижение избыточности может быть использовано для сокращения размера образа
- Приходится анализировать инициализированное состояние программы
  - Включая объекты в памяти
- Большинство существующих алгоритмов не применимо

Инициализация классов в Java

- Спецификация требует ленивой инициализации при первом обращении к классу

- Для некоторых классов ранняя инициализация не меняет поведения приложения

- Такие классы можно инициализировать при подготовке образа

Ранняя инициализация классов

- Ускоряет инициализацию на целевом устройстве
- Сокращает размера образа за счет удаления методов, используемых только для инициализации
- Увеличивает размер образа за счет создания объектов в памяти

Существующий подход

- ExoVM — все загруженные классы инициализируются перед анализом достижимости методов [1]

[1] The ExoVM System for Automatic VM and Application Reduction / B. L. Titzer [и др.] // SIGPLAN Not. — New York, NY, USA, 2007.

Недостатки

- Для некоторых классов ранняя инициализация может нарушить поведение
  - Например, инициализация класса имеет побочные эффекты
- Инициализируются неиспользуемые классы
  - Может увеличить размер образа за счет создания неиспользуемых, но достижимых объектов

Какие классы инициализировать?

- Класс нельзя инициализировать, если
  - он не может быть инициализирован в процессе исполнения программы
  - инициализатор класса обладает побочными эффектами

Когда инициализировать классы?

- До анализа достижимости методов?
  - Неизвестно, какие классы могут быть инициализированы достижимым кодом
- После анализа достижимости методов?
  - Методы, используемые только для инициализации, становятся недостижимы

Предлагаемое решение

- Выборочная инициализация классов в процессе анализа достижимости методов

- Вычисляем множество потенциально инициализируемых классов

- Для каждого класса есть выбор, инициализировать во время анализа или нет

  - Анализируем код инициализаторов чтобы определить безопасна ли инициализация

Удаление неиспользуемых полей

- Стандартный критерий удалимости — отсутствие операций чтения в достижимом коде [1]
  - Позволяет удалять инициализируемые, но не используемые поля
  - Не применим для объектных полей в Java поскольку нарушает семантику финализации

[1] A Study of Dead Data Members in C++ Applications / P. F. Sweeney, F. Tip // SIGPLAN Not. — New York, NY, USA, 1998.

Finalizer Guardian

Идиоматический Java код

```
class Base {
  private final Object finalizerGuardian =
      new Object() {
       protected void finalize() {
          /* Finalize Base class instance */
       }
    }
}
```

Удаление инициализируемых, но не используемых полей нарушает поведение

Неудалимые объекты

- Объекты, удаление которых приложение может отследить
- Финализируемые объекты
- Объекты, доступные посредством специальных ссылок
  - WeakReference, SoftReference, …

Предлагаемое решение

- Поле нельзя удалять, если через ссылку в поле прямо или косвенно может быть достижим объект неудалимого типа

- Для определения данного свойства используем анализ типов

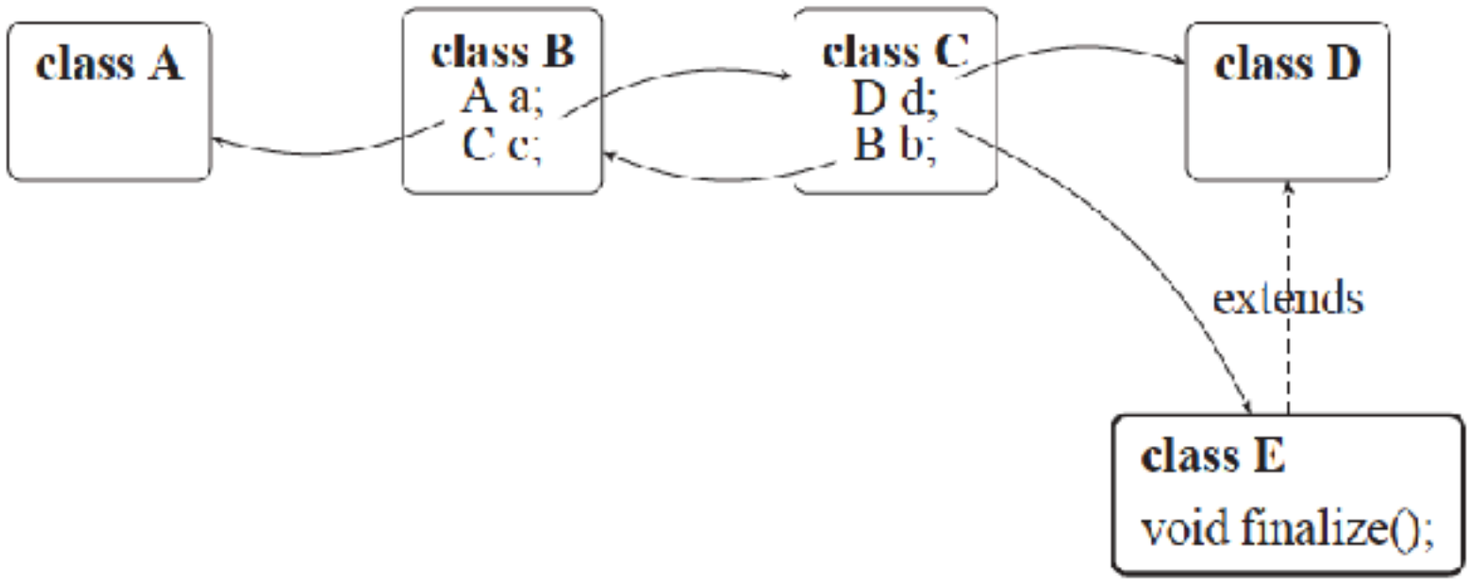
Анализ типов

- Объявленный тип поля определяет множество типов объектов, на которые может указывать поле
- Типы финализируемых объектов известны статически
- Типы объектов, на которые устанавливаются специальные ссылки, вычисляются с помощью межпроцедурного анализа указателей

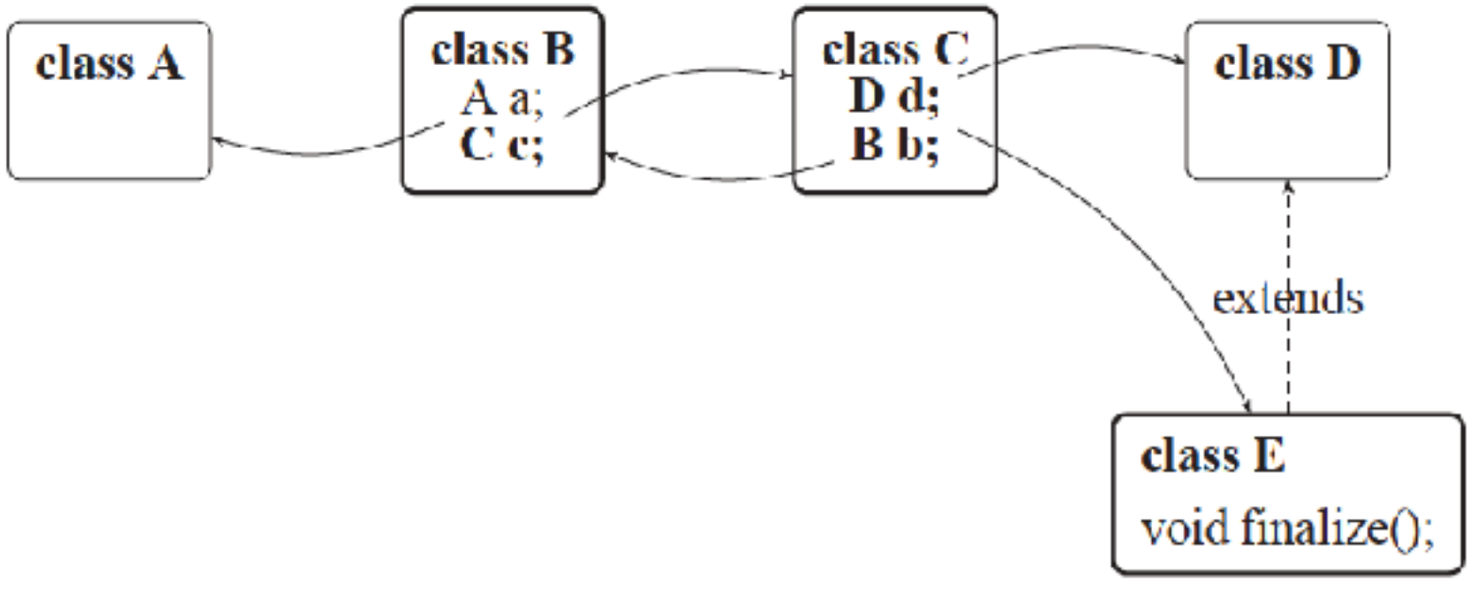
Анализ типов

- Для каждого класса вычисляем множество полей, которые могут на него ссылаться
- Включаем в множество неудалимых типов классы, которые транзитивно указывают на неудалимые объекты
- Поля, которые могут ссылаться на неудалимые типы нельзя удалять

Анализ типов



(а) Начало алгоритма



(б) Конец алгоритма

Специализация набора инструкций

- Java байт-код компактнее машинного кода за счет более высокоуровневого набора инструкций
- В закрытой модели можно специализировать набор инструкций для более компактного кодирования

Сокращение размера

- Свертка аргумента

  aload <0> => aload_0

- Укорачивание аргумента

  goto_w <2 byte offset> => goto <1 byte offset>

- Свертка последовательности инструкций

  isub, ifeq <offset> => if_icmpeq <offset>

Специализация Java байт-кода

- Существующие алгоритмы не используют свертку и укорачивание аргументов совместно со сверткой последовательностей инструкций

aload <5>, getfield <2 byte index> =>
aload_5_getfield <1 byte index>

Упрощение исходного набора инструкций

- Избыточные инструкции можно выразить в виде последовательностей базовых инструкций
- Удаление избыточных инструкций из исходного набора увеличивает эффективность сжатия [1]
- Существующие алгоритмы специализации набора инструкций для Java не упрощают исходный набор инструкций

[1] Code Compression / J. Ernst [и др.] // SIGPLAN Not. — New York, NY, USA, 1997.

Шаблоны

- Последовательности инструкций, параметризованные значениями некоторых из аргументов
- Значения остальных аргументов фиксируются шаблоном
- Размер параметра шаблона может быть меньше размера аргумента в оригинальной инструкции

Пример шаблона

aload 5, getfield <*1 byte>

Фиксированный аргумент

Параметризованный аргумент

Кодирование шаблонов

- Шаблоны кодируются с помощью специализированных инструкций
- Параметризованные аргументы — аргументы новых инструкций

Пример инструкции

aload_5_getfield <1 byte index>

Опкод,
1 байт

Аргумент,
1 байт

Сокращение размера

- Свертка аргументов — инструкции в шаблоне с фиксированным аргументом
- Укорачивание аргументов — параметр шаблона короче, чем аргумент оригинальной инструкции
- Сворачивание последовательностей инструкций — шаблоны последовательностей инструкций

Расширение набора инструкций

- Упрощение исходного набора инструкций
- Жадный итеративный алгоритм
- На каждом шаге выбирается шаблон с максимальным весом
    - Вес шаблона — оценка сокращения размера при кодировании шаблона новой инструкцией

Программная реализация

- Алгоритмы реализованы в виртуальной машине CLDC HotSpot Implementation (HI)
- CLDC HI используется в платформе Oracle Java ME Embedded

Эксперименты

Понижение избыточности

Эксперименты
Набор приложений

1	Hello World	CLDC	приложение, печатающее ”Hello World”
2	Chess	CLDC	Игра шахматы
3	Crypto	CLDC	Криптографический пакет
4	Parallel	CLDC	Тест на многопоточное исполнение
5	kXML	CLDC	XML парсер
6	PNG	CLDC	Декодер PNG изображений
7	RegExp	CLDC	Пакет вычисления регулярных выражений
8	AMS	MEEP	Application Management System

Эксперименты
Набор приложений

1	Hello World	CLDC	приложение, печатающее ”Hello World”
2	Chess	CLDC	Игра шахматы
3	Crypto	CLDC	Криптографический пакет
4	Parallel	CLDC	Тест на многопоточное исполнение
5	kXML	CLDC	XML парсер
6	PNG	CLDC	Декодер PNG изображений
7	RegExp	CLDC	Пакет вычисления регулярных выражений
8	AMS	MEEP	Application Management System

Минимальное приложение для платформы CLDC

Эксперименты
Набор приложений

1	Hello World	CLDC	приложение, печатающее ”Hello World”
2	Chess	CLDC	Игра шахматы
3	Crypto	CLDC	Криптографический пакет
4	Parallel	CLDC	Тест на многопоточное исполнение
5	kXML	CLDC	XML парсер
6	PNG	CLDC	Декодер PNG изображений
7	RegExp	CLDC	Пакет вычисления регулярных выражений
8	AMS	MEEP	Application Management System

Приложения из пакета EEMBC GinderBench

Эксперименты
Набор приложений

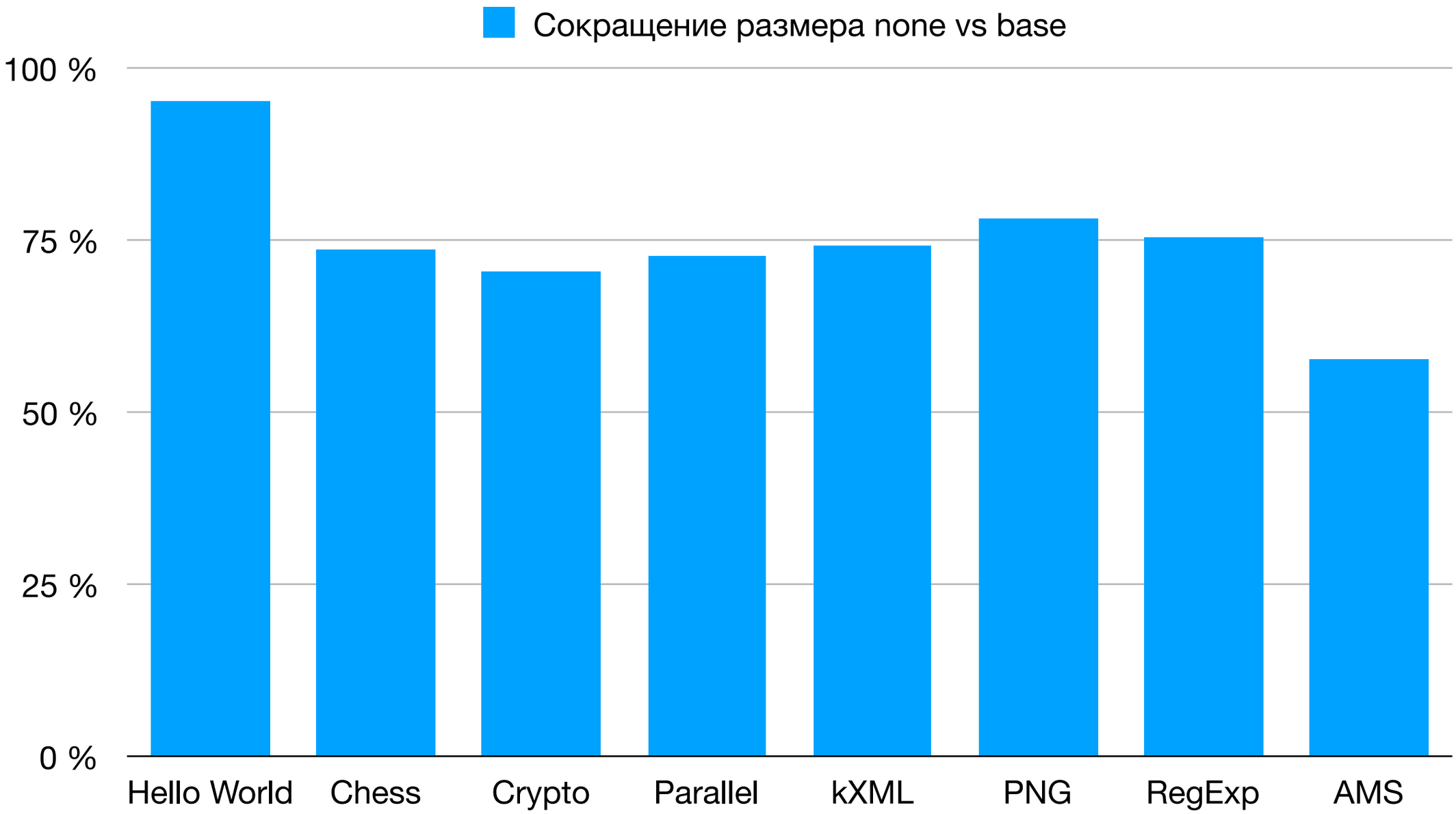
1	Hello World	CLDC	приложение, печатающее ”Hello World”
2	Chess	CLDC	Игра шахматы
3	Crypto	CLDC	Криптографический пакет
4	Parallel	CLDC	Тест на многопоточное исполнение
5	kXML	CLDC	XML парсер
6	PNG	CLDC	Декодер PNG изображений
7	RegExp	CLDC	Пакет вычисления регулярных выражений
8	AMS	MEEP	Application Management System

Приложение для платформы MEEP

Эксперимент 1

- Определить сокращение размера, обеспечиваемое реализованными алгоритмами
- Измерены размеры образов:
  - none — без понижения избыточности
  - base — с понижением избыточности с помощью реализованных алгоритмов

none vs base



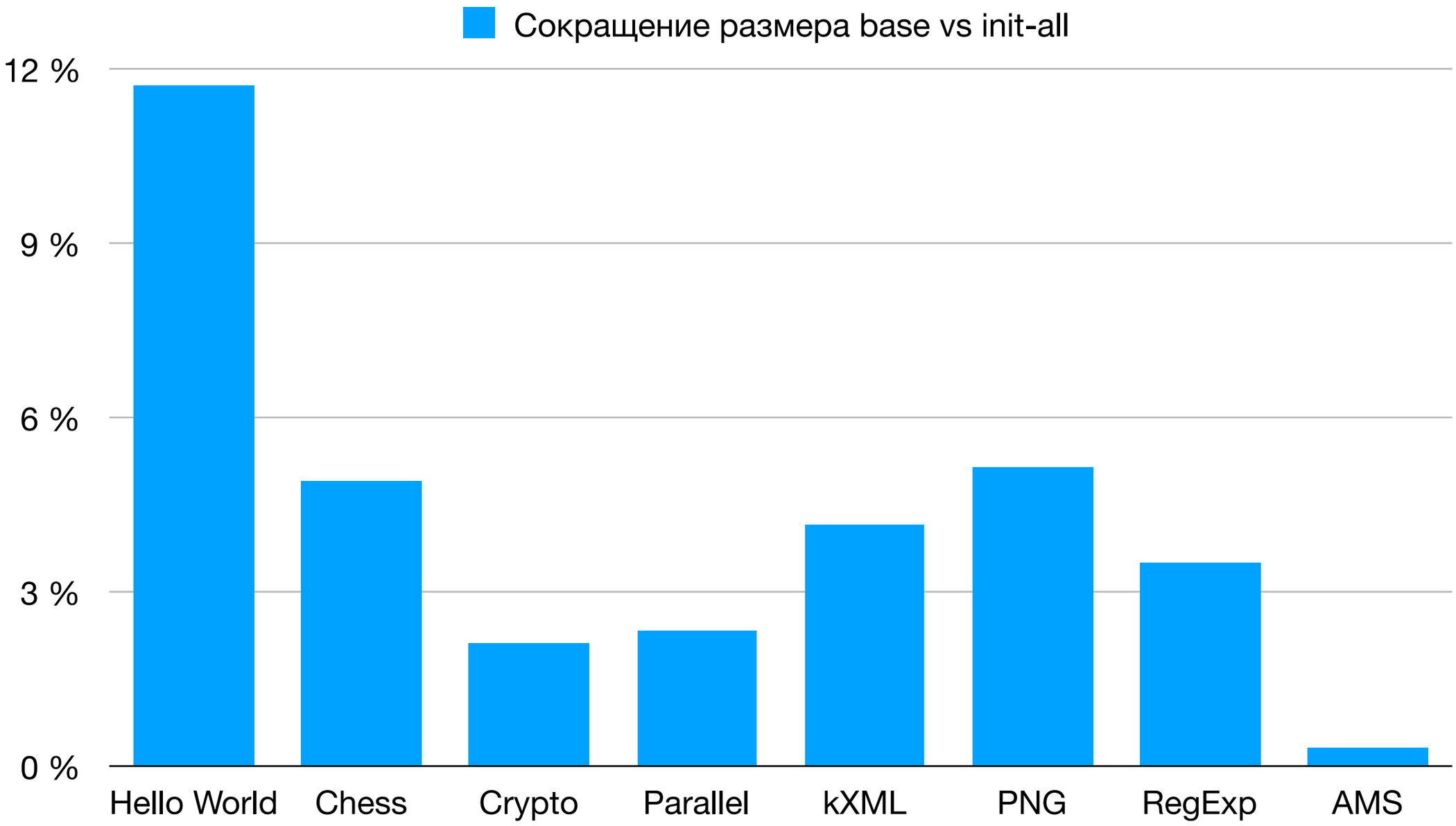
Сокращение размера 57,62–95,18 % (среднее значение 74,57 %)

Эксперимент 2

- Определить, на сколько выборочная инициализация классов сокращает размер образа по сравнению с безусловной инициализацией

- Реализована вариация алгоритма, инициализирующая все классы до анализа достижимости методов — init-all

base vs init-all



Сокращение размера 0,3–11,71 % (среднее значение 4,27 %)

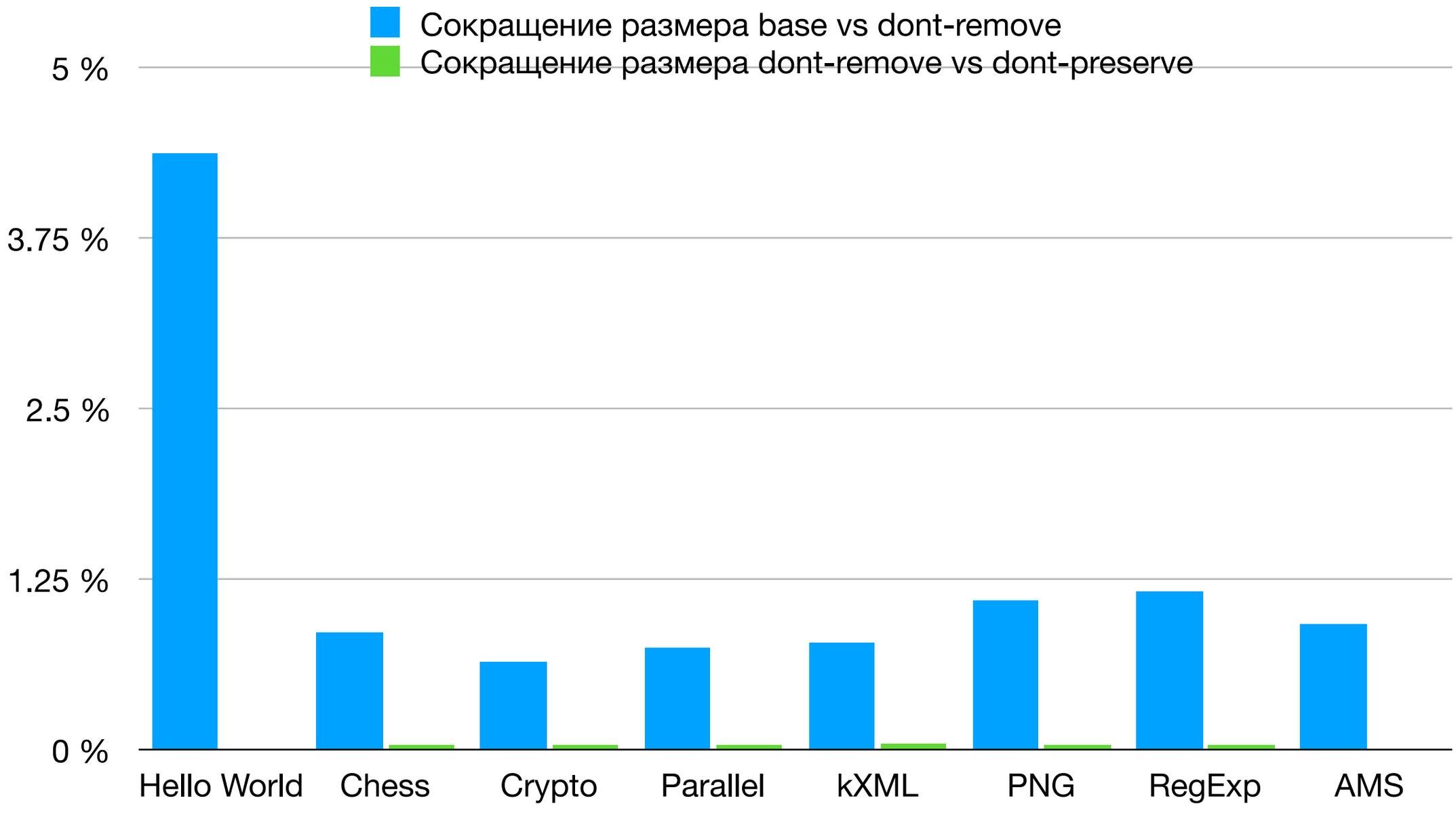
Эксперимент 3

- Определить, как удаление инициализируемых, но неиспользуемых полей влияет на размер образа.

- Реализованы две модификации алгоритма:
  - dont-remove – не удаляет инициализируемые, но неиспользуемые объектные поля;
  - dont-preserve – не сохраняет семантику финализации и удаляет все инициализируемые, но неиспользуемые объектные поля.

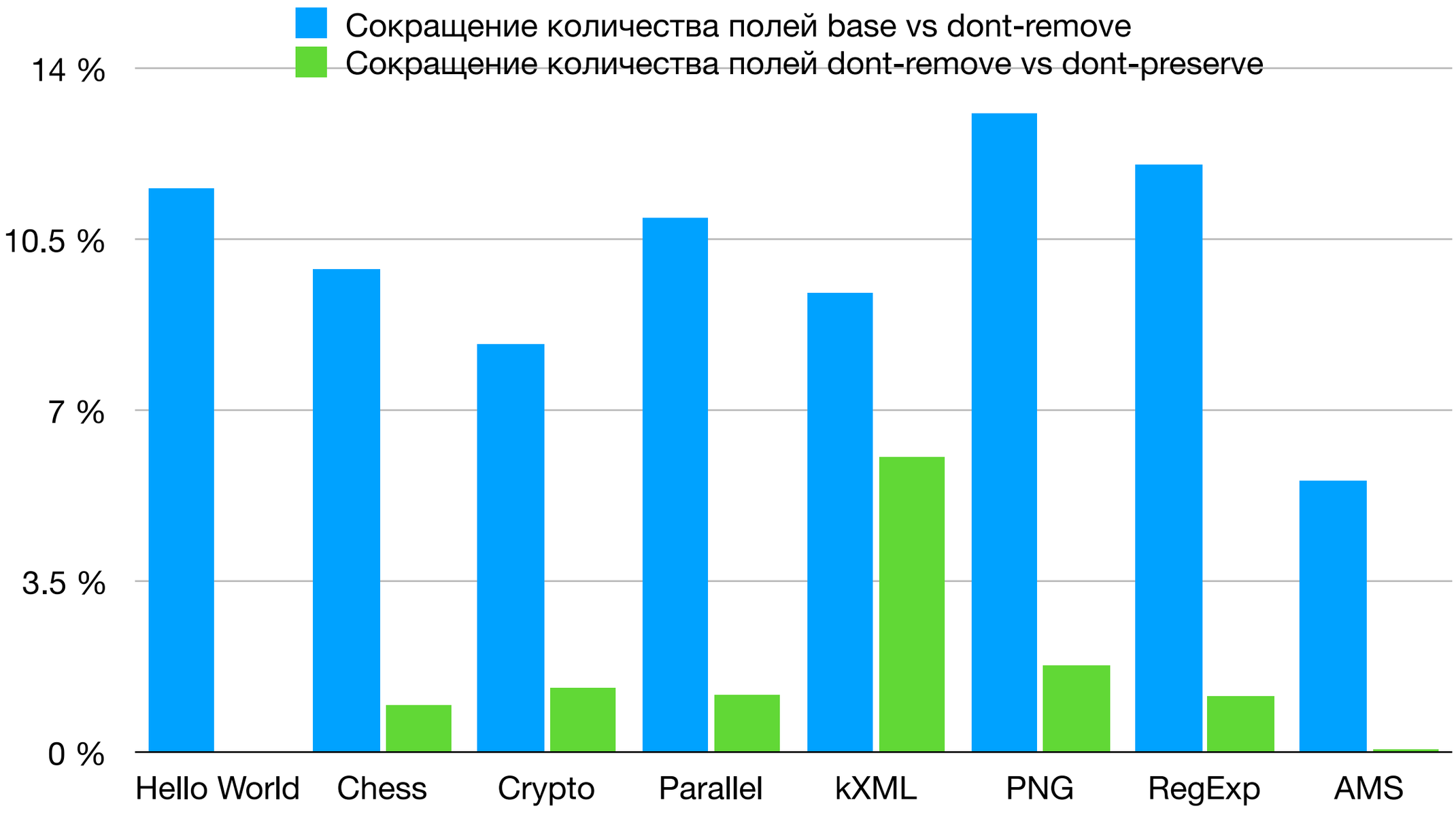
base vs dont-remove

Размер образа



Сокращение размера 0,64–4,37% (среднее значение 1,32%)

base vs dont-remove
Количество полей



Сокращение количества полей
5,55–13,08% (среднее значение 10,09%)

Эксперименты

Специализация набора инструкций

Степень сжатия

- Эффективность сжатия определяется степень сжатия

$$1 - \frac{compressed\_size}{original\_size}$$

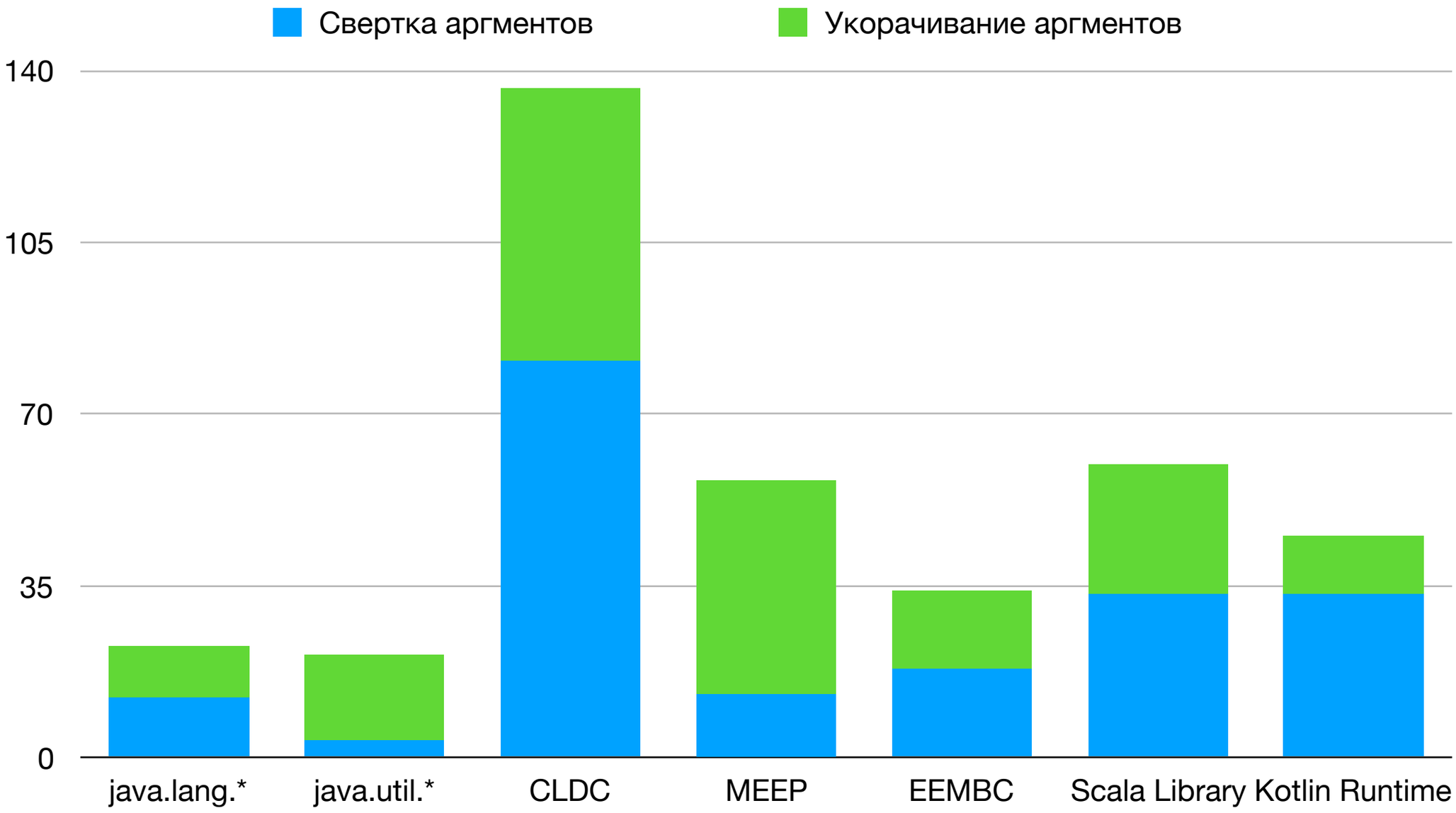
Набор классов

1	java.lang.*	реализации стандартных пакетов Java из OpenJDK версии 7u17
2	java.util.*	
3	CLDC	Java-классы реализации стандарта Connected Limited Device Configuration 8
4	MEEP	Java-классы реализации стандарта Java~ME Embedded Profile
5	EEMBC	EEMBC GrinderBench — набор тестов для измерения производительности JavaME платформ
6	Scala Library	стандартная библиотека языка Scala версии 2.10.3
7	Kotlin Library	подмножество стандартной библиотеки языка Kotlin версии 0.6.13, написанное на языке Kotlin

Эксперимент 4

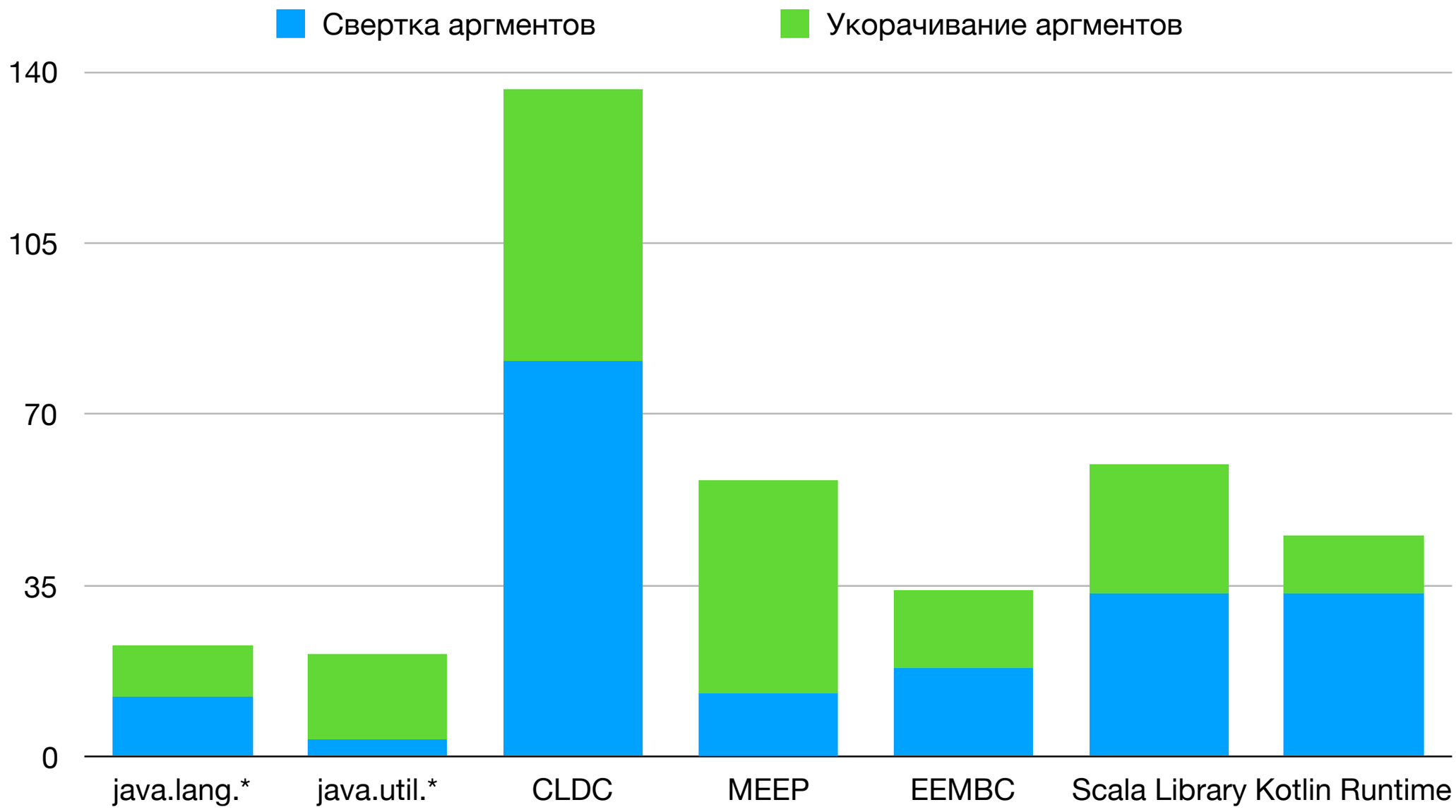
- Как контекстная свертка и укорачивание аргументов влияет на степень сжатия?

Контекстная свертка и укорачивание аргументов



Свертка аргументов увеличивает степень сжатия в среднем на 22%

Контекстная свертка и укорачивание аргументов

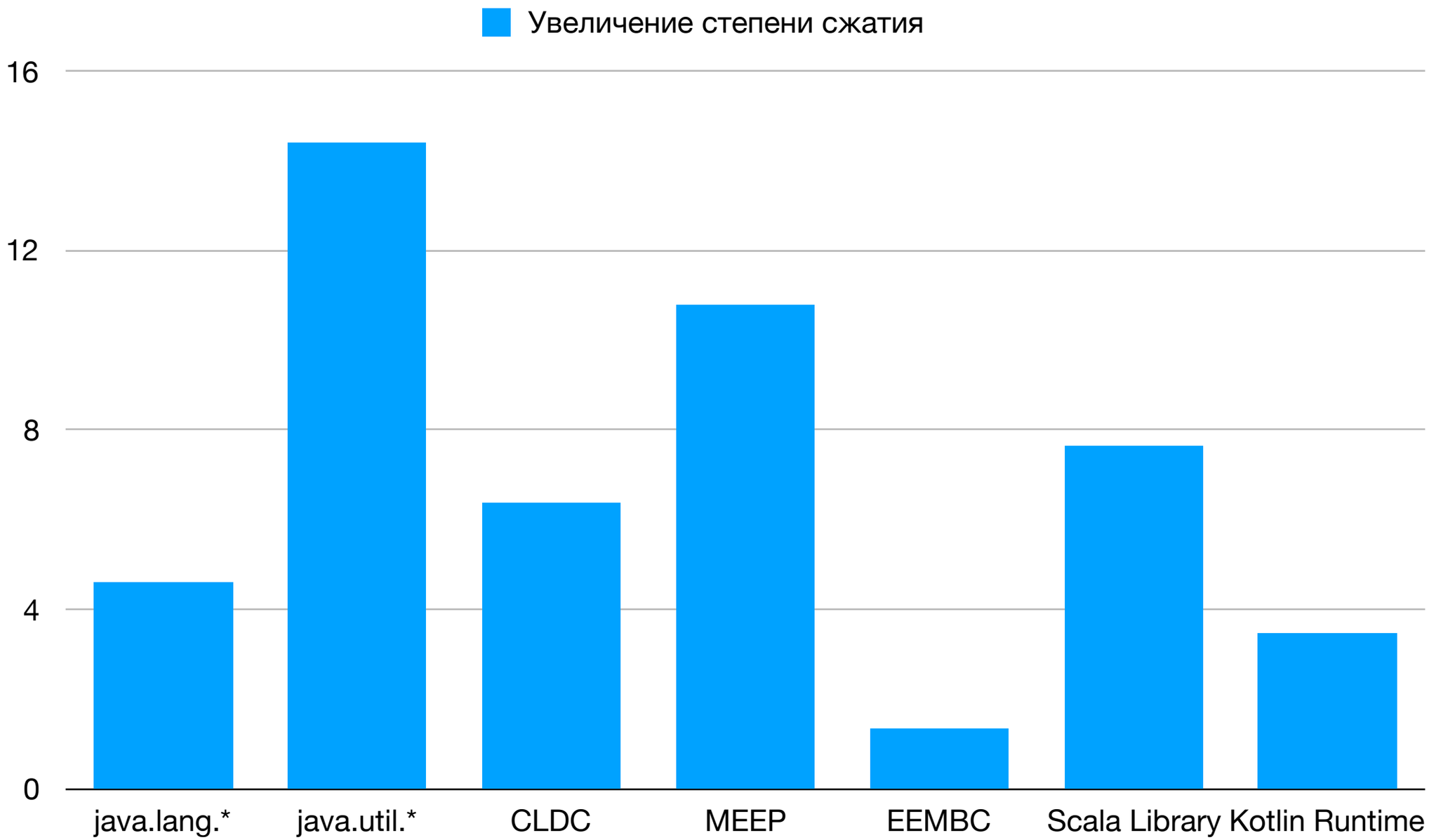


Укорачивание аргументов увеличивает степень сжатия в среднем еще на 22%

Эксперимент 5

- Как упрощение исходного набора инструкций влияет на степень сжатия?

Упрощение набора инструкций



Упрощение набора инструкций увеличивает степень сжатия в среднем на 6,46%

Заключение

- Разработаны и реализованы алгоритмы понижения избыточности для Java, применимые при использовании раздельной инициализации
- Разработаны и реализованы алгоритм специализации набора инструкций, минимизирующий суммарный размер Java-программы и интерпретатора, необходимого для ее исполнения