Преобразование по уплотнению кода в LLVM

Илья Скапенко Денис Дубров

Южный Федеральный Университет
ФММиКН
г. Ростов-на-Дону

5 апреля 2017 г.

Инфраструктура LLVM

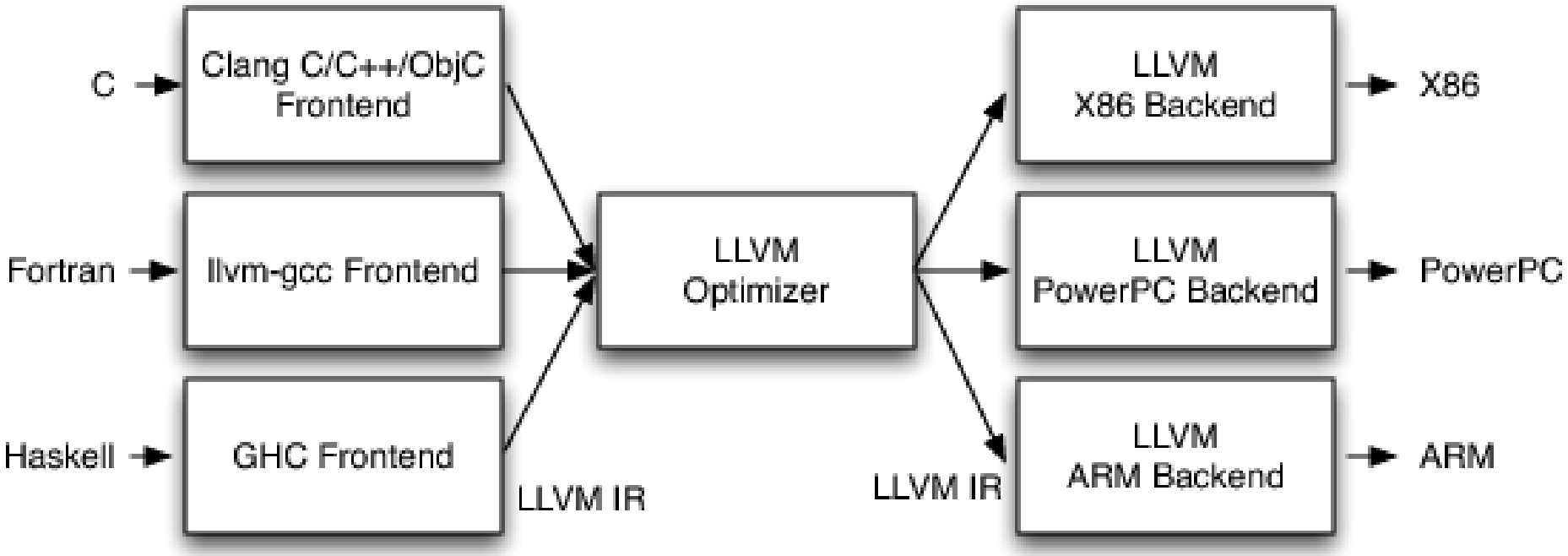

Рис. 1: Инфраструктура LLVM

Основные цели

- Написать проход для LLVM 4.0 по вынесению общих базовых блоков в отдельную функцию на уровне промежуточного представления
- Произвести замеры размера кода до и после работы оптимизации
- Сделать выводы об отпимизациях размера кода на уровне промежуточного представления LLVM

Особенности промежуточного представления LLVM

- Высокоуровневая структура (модули, функции, базовые блоки)
- Strong type system (строгая типизация)
- SSA (Static Single Assignment), бесконечное количество регистров. Phi-инструкции
- Terminator-инструкции

Особенности промежуточного представления LLVM

- Высокоуровневая структура (модули, функции, базовые блоки)
- Strong type system (строгая типизация)
- SSA (Static Single Assignment), бесконечное количество регистров. Phi-инструкции
- Terminator-инструкции

```
int i = 0;
while((i+=2) < 10);
```

```
b0:   i0 = 0;
      jmp b1
b1:
      φ = [b0: i0],[b1: n]
      n = φ + 2
      jmp b2 if n>=10 else b1
```

Вынесение общих базовых блоков

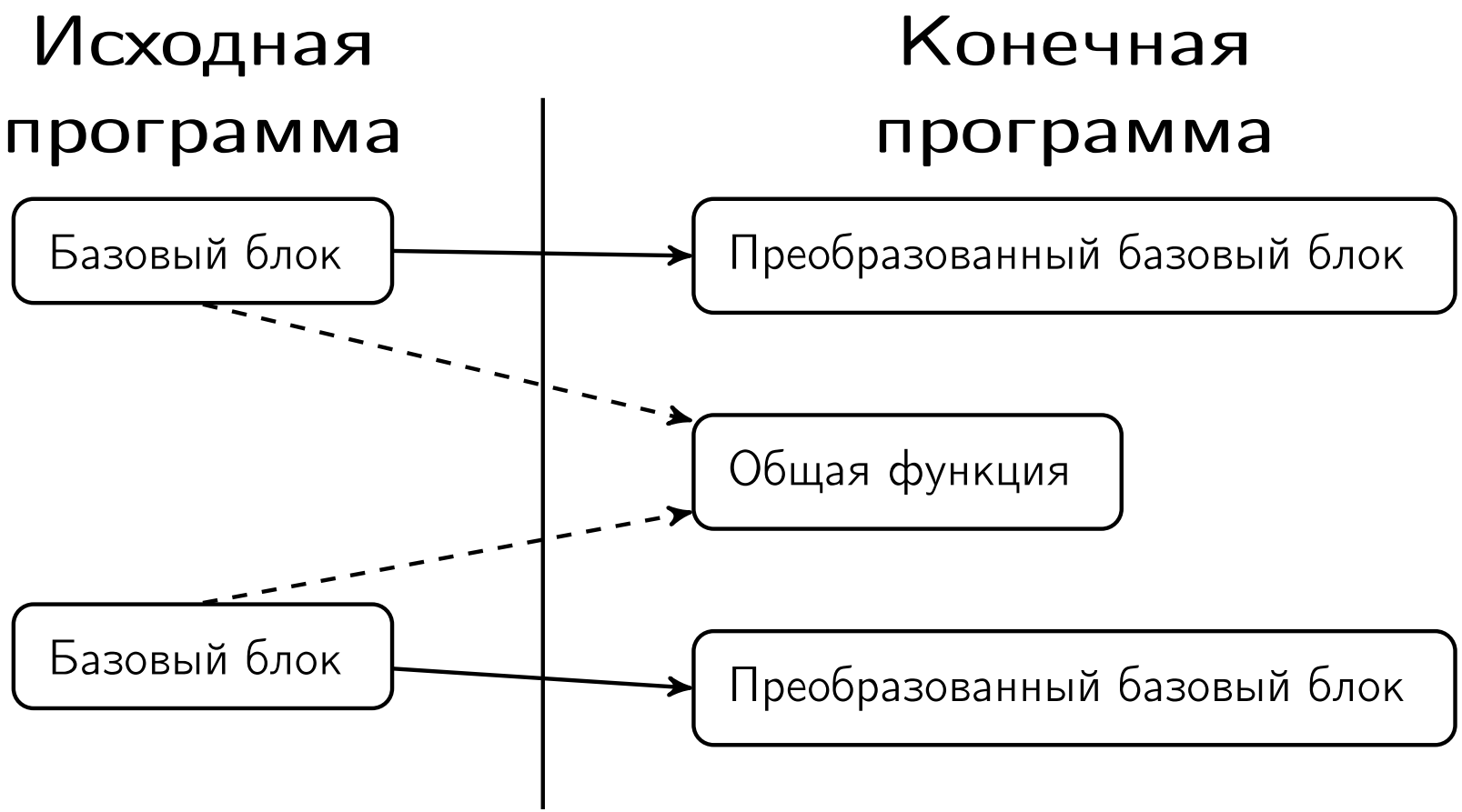

Рис. 2: Визуализация прохода процедурной абстракции базовых блоков

Алгоритм вынесения базовых блоков

1. Сравнение базовых блоков [MergeFunctions]
2. Слияние базовых блоков

Особенности сравнения, отличные от MergeFunctions

- Исключение из сравнения атрибутов, невлияющих на кодогенерацию

JumpTable, NoReturn, ReadOnly...

Особенности сравнения, отличные от MergeFunctions

- Исключение из сравнения атрибутов, невлияющих на кодогенерацию
- Пропуск Phi и Terminator-инструкций

```
1  block1:
2    %v = add i32 %d, %k
3    ret i32 %v
4  ...
5  block2:
6    %p = phi i32 [0, bl3],
          [1, bl4]
7    %r = add i32 %v, %p
8    br label block5
```

block1 $\leftrightarrow$ block2

Особенности сравнения, отличные от MergeFunctions

- Исключение из сравнения атрибутов, невлияющих на кодогенерацию
- Пропуск Phi и Terminator-инструкций
- Использование свойства коммутативности операторов

$$\%l1 = \text{add i32 } \%d, 1$$
$$\leftrightarrow$$
$$\%l1 = \text{add i32 } 1, \%d$$

```
1  blockN:
2    %d1 = add nsw %idx, 5
3    %d2 = mul nsw %d1, 6
4    %call = call i32 f(%d2)
5    br label %blockN_1
```

1. Находится список входных и выходных параметров

```
1  blockN:
2    %d1 = add nsw %idx, 5
3    %d2 = mul nsw %d1, 6
4    %call = call i32 f(%d2)
5    br label %blockN_1
```

Входные параметры: %idx

Выходные параметры определяются наличием использованиями переменных вне выносимого блока

1. Находится список входных и выходных параметров
2. $OUT = \bigcup_{i=1}^{N} OUT_i$

```
1  blockN:
2    %d1 = add nsw %idx, 5
3    %d2 = mul nsw %d1, 6
4    %call = call i32 f(%d2)
5    br label %blockN_1
```

Выходные параметры блока A: 4

Выходные параметры блока B: 3

Выходные параметры: 3, 4

1. Находится список входных и выходных параметров
2. $OUT = \bigcup_{i=1}^{N} OUT_i$
3. Ищется (создаётся, если не нашли) подходящая функция

```
1  define i32 @F0(i32 %In0, i32*
       %Out0){
2  entry:
3    %d1 = add nsw %In0, 5
4    %d2 = mul nsw %d1, 6
5    store i32 %d2, %Out0
6    %call = call i32 f(%d2)
7    br label %blockN_1
8    ret i32 %call
9  }
```

1. Находится список входных и выходных параметров
2. $OUT = \bigcup_{i=1}^{N} OUT_i$
3. Ищется (создаётся, если не нашли) подходящая функция
4. Выполняется замена базовых блоков

```
1  blockN:
2    %ptr = alloca i32
3    %call = tail call @F0(%idx,
         %ptr)
4    %val = load i32, i32* %ptr
5    br label %blockN_1
```

Первоначальные результаты

Проект	Архитек-тура	Размер до оптимизации (Кб)	Размер после оптимизации (Кб)	Уменьшение размера
Tinyxml	X86-64 ARM	28.6 26.5	29.6 27.5	-3.4% -4%
Libcurl	X86-64 ARM	292 303.8	301.8 326.3	-3.4% -7.4%
FatFS	X86-64 ARM	9.8 11.4	10.9 12.9	-11.4% -13.7%

Методы уменьшения конечного размера кода

Фильтрация базовых блоков по количеству инструкций

```
block1:
  store i32 5, %stor
  br label block_n

-----------------------------------------

block2:
  %v0 = load i8, i8 * %done
  %tobool0 = icmp eq i8 %v0, 0
  br i1 %tobool0, label block_k, label block_n
```

Уменьшение количества выходных параметров

Перемещение или дублирование «бесплатных» инструкций: bitcast, getelementptr, lifetime.start

```
block0:
  %c = call i8* malloc(4)
  %b0 = bitcast i8* %c to i32*

-------------------------------------

blockN:
  %i = alloca i32, align 4
  %i_cast = bitcast i32* %i to i8*
   call void llvm.lifetime.start(4, %i_cast)
```

Методы определения размера кода

1. Размер начального базового блока
2. Размер созданной функции
3. Размер вызова новой функции

- Доступ к кодогенератору llvm::TargetTransformInfo
- Определение размера кода для конкретной архитектуры

Конечные результаты

Проект	Архитек-тура	Размер до оптимизации (Кб)	Размер после оптимизации (Кб)	Уменьшение размера
Tinyxml	X86-64 ARM	28.6 26.5	28.4 26.3	0.5% 0.7%
Libcurl	X86-64 ARM	292 303.8	292 303.7	0.01% 0.03%
FatFS	X86-64 ARM	9.8 11.4	9.8 11.4	0% 0%

Итоги работы

- Написан подключаемый модуль для LLVM 4.0
- Модуль протестирован на корректность (lit, FileChecker, lli)
- Проверена оптимальность работы прохода
- https://github.com/skapix/codeCompaction