Алгоритм создания случайных графов потока управления для анализа глобальных оптимизаций в компиляторе

Рыбаков Алексей Анатольевич

ЗАО «МЦСТ», Москва, Россия

rybakov.aax@gmail.com

Аннотация. *Оптимизирующие компиляторы в процессе своей работы могут использовать одно или несколько различных промежуточных представлений. Промежуточное представление, основанное на графе потока управления, является одним из них. Граф потока управления отражает параллелизм исполнения на уровне линейных участков. На графе потока управления выполняется ряд глобальных оптимизаций. В данной статье предлагается методика создания случайных графов потока управления для проверки работоспособности и анализа эффективности глобальных оптимизаций.*

Ключевые слова

Оптимизирующие компиляторы, промежуточное представление, граф потока управления, профиль исполнения, расположение в памяти линейных участков программы, переходы.

1 Введение

В настоящее время разработка оптимизирующих компиляторов не теряет своей актуальности, так как скорость исполнения приложений является критическим ресурсом [1]. Крайне востребованной и бурно развивающейся технологией является динамическая компиляция [2, 3], позволяющая повысить производительность вычислительной системы за счет объединения достоинств статической оптимизирующей компиляции программы и интерпретации кода.

Системы, использующие динамическую компиляцию, сначала транслируют текст программы в свое внутреннее представление (байт-код), который может быть исполнен с помощью виртуальной машины или переведен в код целевой архитектуры. В процессе выполнения байт-кода осуществляется его мониторинг, направленный на локализацию наиболее часто исполняемых участков кода. Далее, динамический компилятор может перекомпилировать обнаруженные часто исполняемые участки байт-кода, применяя нужный набор и агрессивность оптимизаций с учетом профиля кода и его отличительных характеристик [4, 5]. Таким образом, система динамической компиляции позволяет применять дорогую по вычислительным ресурсам оптимизацию только для того кода, который является критичным для данного приложения, оставляя редко выполняемый код неоптимизированным.

Важной частью динамических оптимизирующих компиляторов являются оптимизации переходов, так как плотность команд перехода в типовых задачах, работающих с целочисленной арифметикой, довольно высока.

2 Расположение в памяти линейных участков промежуточного представления

В процессе работы оптимизирующий компилятор может использовать одно или несколько различных промежуточных представлений [6]. В настоящей работе рассматривается промежуточное представление, основой которого является граф потока управления (control flow graph - CFG). Линейным участком будем называть такую последовательность команд промежуточного представления, в которой управление из других частей программы может

передаваться только на первую команду данной последовательности. Иными словами, внутри линейного участка нет точек входа. Граф потока управления – ориентированный граф, узлами которого являются линейные участки, а ребрами – переходы между ними, причем направление ребра соответствует направлению перехода.

Переходы между линейными участками можно разделить на два типа. К первому типу отнесем переходы, выполняемые с помощью специальных команд передачи управления. Ко второму типу отнесем такие переходы, когда после выполнения последней команды линейного участка управление должно передаться другому линейному участку, однако команда перехода отсутствует. Такой переход будем называть провалом. В случае множественных выходов из узла CFG только один выход может рассматриваться как провал, а все остальные реализуются с помощью команды перехода, и это определяется при построении CFG. Таким образом, из узла может исходить не более одного провала, но в него может заходить несколько провалов.

Граф потока управления является логической структурой, отражающей параллелизм программы на уровне линейных участков. Он активно используется компилятором для применения различных глобальных оптимизаций [7]. После завершения этапа оптимизации запускается механизм генерации кода. Во время генерации кода для каждого линейного участка промежуточного представления создается соответствующий блок результирующего кода, который записывается в память (далее будем говорить, что линейный участок записывается в память). Линейные участки записываются в память последовательно в определенном порядке.

Рассмотрим два узла CFG, связанные провалом, т.е. после обработки всех команд первого узла управление должно передаваться на второй узел. Рассмотрим, как линейные участки, соответствующие данным двум узлам, могут располагаться в памяти. Если два линейных участка, связанные провалом, располагаются в памяти строго последовательно (сначала расположен предшественник по провалу, далее – последователь), то после исполнения всех команд первого линейного участка управление естественным образом перейдет на второй линейный участок. Тогда будем считать, что два линейных участка склеились по провалу, или что провал удалось использовать. Если же линейные участки окажутся расположенными в памяти другим способом (в другом порядке или между ними встретятся другие команды), то в том месте, где ожидается переход по провалу, придется создавать команду передачи управления на последователя провала. Создание лишней команды передачи управления ухудшает результирующий код, поэтому для повышения производительности требуется эффективное использование провалов. Оптимизация расположения в памяти линейных участков промежуточного представления, описанная в [8] позволяет достичь эффективного использования провалов.

Для проверки работоспособности и оценки эффективности оптимизации расположения в памяти линейных участков промежуточного представления использовались случайные графы потока управления, алгоритм генерации которых описан в следующем разделе.

3 Моделирование графа потока управления

В данном разделе вводятся строгие определения графа потока управления и профиля исполнения и приводится механизм построения графа потока управления произвольной случайной структуры, обладающего корректным профилем исполнения. Более полное рассмотрение графа потока управления может быть найден в [9].

3.1 Формализация графа потока управления и профиля исполнения

Базовым блоком будем называть последовательность инструкций с одним входом и одним выходом. Наличие одного входа означает, что никакая инструкция базового блока кроме первой не может быть назначением команды перехода. Наличие одного выхода означает, что никакая инструкция базового блока кроме последней не может быть командой перехода. Линейным участком будем называть последовательность инструкций с одним входом. Очевидно, что любой базовый блок является линейным участком. Назовем регионом произвольный набор линейных участков, связанных между собой передачей управления.

Граф потока управления (control flow graph, CFG) региона – ориентированный граф, узлами которого являются линейные участки региона, а с помощью ребер указана передача управления между ними.

Рассмотрим некоторый CFG ($D = (V_D, E_D)$ – ориентированный граф). Обозначим через $E^I(v)$ множество всех ребер, входящих в узел v, а через $E^O(v)$ – множество всех ребер, выходящих из узла v. Степень захода и исхода вершины v будем обозначать через $d^I(v)$ и $d^O(v)$ соответственно. Узел v CFG будем называть входным узлом региона, если в него не входит ни одно ребро CFG ($d^I(v) = 0$). Узел v CFG будем называть выходным узлом региона, если из него не выходит ни одно ребро CFG ($d^O(v) = 0$).

Для удобства и технических нужд будем рассматривать два вспомогательных узла CFG: глобальный вход и глобальный выход. Глобальным входом назовем такой вспомогательный узел CFG, в который не входит ни одно

ребро CFG, и который соединен вспомогательными выходными ребрами со всеми входными узлами региона. Глобальный вход региона будем обозначать v_{root}. Глобальным выходом назовем такой вспомогательный узел CFG, из которого не выходит ни одно ребро CFG, и который соединен входными ребрами со всеми выходными узлами региона (то есть из каждого выходного узла выходит вспомогательное ребро, которое входит в глобальный выход). Глобальный выход региона будем обозначать v_{sink}.

Пусть на ребрах графа D задана неотрицательная функция $\omega(e) \geq 0$. Будем говорить, что функция $\omega(e) \geq 0$ является профильной функцией графа D (и соответствующего CFG), если для любого узла v, кроме глобального входа и глобального выхода, выполняется условие

$$\sum_{e \in E^I(v)} \omega(e) = \sum_{e \in E^O(v)} \omega(e). \tag{1}$$

При этом, значение $\omega(e)$ будем называть *счетчиком* ребра e. Счетчиком узла, отличного от глобального входа и глобального выхода, назовем сумму счетчиков входящих в него или выходящих из него ребер. В глобальный вход не входит ни одно ребро, так что для этого узла счетчиком является сумма счетчиков выходящих ребер. Аналогично, счетчиком глобального выхода является сумма счетчиков входящих в него ребер. Вероятностью ребра CFG назовем отношение его счетчика к счетчику узла, из которого данное ребро выходит.

Если для CFG задана профильная функция, то совокупность счетчиков узлов и ребер и вероятностей ребер будем называть профилем исполнения CFG.

3.2 Создание случайного графа потока управления

Создание случайного CFG со статическим профилем исполнения начнем с создания элементарного CFG, состоящего из двух узлов (глобального входа и глобального выхода), соединенных переходом (см. рисунок 1, а). При этом, для единственного ребра e задан начальный счетчик $\omega(e)$ (равный счетчику глобального входа и глобального выхода) и вероятность передачи управления по этому ребру $\tau(e) = 1$.

К данному элементарному CFG будем многократно последовательно применять одно из атомарных преобразований для получения случайного CFG с корректным статическим профилем исполнения.

Первое из рассматриваемых атомарных преобразований – это помещение узла на выбранное ребро. Рассмотрим произвольное ребро $e = v_1v_2$. Требуется добавить новый узел v на ребро e. При этом ребро e удалится, а вместо него в CFG добавятся два других ребра: $e_1 = v_1v, e_2 = vv_2$ (см. рисунок 1, б). Обозначим это преобразование через $T_n = T_n(e)$ (node adding). Профильные характеристики новых объектов CFG определяются по следующим формулам:

$$\omega(v) = \omega(e),\ \omega(e_1) = \omega(e_2) = \omega(e),\ \tau(e_1) = \tau(e),\ \tau(e_2) = 1. \tag{2}$$

Следующее рассматриваемое преобразование – добавление кратного ребра. Рассматривается произвольное ребро $e = v_1v_2$. В процессе преобразования данное ребро удаляется и вместо него добавляется два кратных ребра с теми же концами v_1 и v_2 и с тем же направлением. Дополнительным параметром преобразования является коэффициент отношения α, в котором поток управления, проходивший через ребро e, делится между двумя новыми ребрами (см. рисунок 1, в). Обозначим это преобразование $T_e = T_e(e, \alpha)$ (multiple edge adding). Профильные характеристики новых объектов CFG определяются по следующим формулам:

$$\omega(e_1) = \frac{\alpha}{1+\alpha}\omega(e),\ \omega(e_2) = \frac{\omega(e)}{1+\alpha},\ \tau(e_1) = \frac{\alpha}{1+\alpha}\tau(e),\ \tau(e_2) = \frac{\tau(e)}{1+\alpha}. \tag{3}$$

Для создания циклов в CFG рассмотрим преобразования добавления петли к выбранной вершине. Рассмотрим произвольную вершину v. Требуется добавить в CFG петлю с концами в данной вершине v и с заданной вероятностью τ. Заметим, что мы не можем добавить петлю к глобальному входу или к глобальному выходу, а значит, можно считать, что у рассматриваемого узла v есть входные и выходные ребра. Преобразования по добавлению петли к произвольному узлу обозначим $T_l = T_l(v, \tau)$ (loop adding). Предположим, что у расматриваемой вершины v есть ровно одно входяшее ребро e_1 и ровно одно выходящее ребро e_2 (см. рисунок 1, г). Формулы для определения профильных характеристик новых и изменившихся объектов имеют следующий вид:

$$\omega(e) = \frac{\tau}{1-\tau}\omega(v),\ \tau(e) = \tau,\ \omega(v') = \frac{\omega(v)}{1-\tau},\ \tau(e_2') = 1-\tau. \tag{4}$$

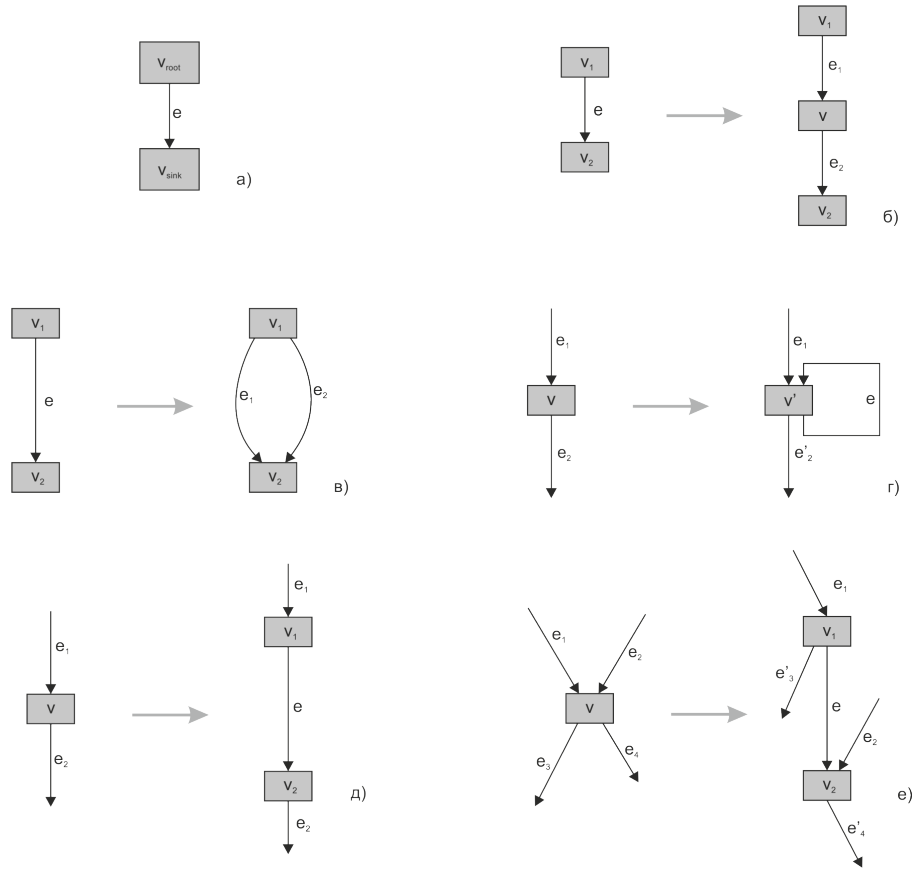

Рис. 1. Атомарные преобразования графа потока управления. а) Элементарый CFG. б) T_n – помещение вершины на ребро. в) T_e – добавление кратного ребра. г) T_l – добавление петли. д) T_x – растягивание вершины. е) T_r – растягивание вершины с перераспределением инцидентных ребер.

Если у вершины v несколько выходных ребер (e_i), то их вероятности корректируются аналогично: $\tau(e'_i) = (1-\tau)\tau(e_i)$.

Можно заметить, что с помощью трех описанных преобразований невозможно создать полноценный цикл. Допустим мы применили преобразование T_l, получив таким образом цикл. Заметим, что узел, к которому мы добавили петлю, является для этого цикла точкой входа и точкой выхода. С помощью описанных трех преобразований этот факт никак изменить нельзя.

Следующее рассматриваемое преобразование – растягивание вершины в ребро. Оно позволяет разнести точку входа и точку выхода цикла.

Итак, пусть имеется вершина v. Сначала предположим, что у этой вершины одно входное ребро (e_1) и одно выходное ребро (e_2). Вместо данной вершины требуется создать две вершины (v_1 и v_2), связать их ребром $e = v_1v_2$ и перенаправить ребро e_1 на вершину v_1, ребро e_2 сделать выходящим из вершины v_2 (см. рисунок 1, д). Преобразование растягивания вершины v в ребро обозначим $T_x = T_x(v)$ (vertex extension). Ниже приведены формулы для определения профильных характеристик новых объектов:

$$\omega(v_1) = \omega(v_2) = \omega(v),\ \omega(e) = \omega(v),\ \tau(e) = 1. \tag{5}$$

К тому же заметим, что на результат никак не повлияло предположение о том, что в узел v входит одно ребро и выходит одно ребро.

В предыдущем преобразовании после создания вместо одной вершины v двух вершин v_1, v_2, соединенных ребром, инцидентные вершине v ребра перенаправлялись таким образом, чтобы поток управления через новое созданное ребро e был равен $\omega(v)$. При этом, все ребра $E^I(v)$ становились входными для вершины v_1, а ребра $E^O(v)$ – выходными для вершины v_2. Следующее дополнительное преобразование позволяет распределить входные ребра

$E^I(v)$ между вершинами v_1, v_2 и, аналогично, распределить выходные ребра $E^O(v)$ между этими же вершинами v_1, v_2. Рассмотрим данное преобразование более детально.

Пусть дана вершина v. Для простоты будем считать, что в нее входит два ребра (e_1, e_2) и выходит два ребра (e_2, e_4). Требуется заменить вершину v на пару вершин v_1, v_2, соединенных ребром e. При этом e_1 должно входить в v_1, а e_2 – в v_2, e_3 должно выходить из v_1, а v_4 – из v_2 (см. рисунок 1, е). Данное преобразование назовем растягиванием вершины с перераспределением инцидентных ребер и обозначим $T_r = T_r(v)$ (vertex extension with **r**earrangement). Профильные характеристики новых и изменившихся объектов CFG определяются по следующим формулам:

$$\omega(v_1) = \omega(e_1),\ \omega(v_2) = \omega(e_4),\ \omega(e) = \omega(e_1) - \omega(e_3),\ \tau(e) = 1 - \frac{\omega(e_3)}{\omega(e_1)},\ \tau(e_3') = \frac{\omega(e_3)}{\omega(e_1)},\ \tau(e_4') = 1. \quad (6)$$

При выполнении преобразования следует учесть, что направление ребра e заранее не известно, и зависит от счетчиков ребер e_1 и e_3. Приведенное соотношение для $\omega(e)$ в формулах 6 верно для случая $\omega(e_1) \geq \omega(e_3)$. Если же $\omega(e_1) < \omega(e_3)$, то ребро e будет иметь обратное направление, и соотношение для $\omega(e)$ нужно взять с обратным знаком.

Если снять ограничение, что в вершину v входит только два ребра и из нее выходит только два ребра, то соотношения 6 принципиально не изменятся. Вместо счетчиков ребер e_1, e_2, e_3, e_4 будут выступать суммы счетчиков ребер из соответствующих множеств. Вероятности выходящих из вершин v_1 и v_2 ребер считаются аналогично делением счетчика ребра на счетчик соответствующего узла.

Заметим, что преобразование T_n является частным случаем преобразования T_x, которое, в свою очередь, является частным случаем преобразования T_r. Таким образом, мы имеем три независимых друг от друга преобразования: T_e, T_l, T_r.

Верно следующее утверждение. Для произвольного CFG с глобальным входом и глобальным выходом ($D = (V_D, E_D)$) может быть найдена последовательность преобразований T_e, T_l, T_r, переводящая элементарный CFG (D_0) в данный граф D.

Доказательство будем проводить по индукции по количеству вершин в графе (n). При $n = 2$ CFG содержит только две вершины - глобальный вход и глобальный выход. От элементарного CFG данный граф может отличаться только количеством ребер. Все ребра рассматриваемого графа выходят из глобального входа и заходят в глобальный выход, то есть являются кратными. Все эти кратные ребра могут быть построены с помощью преобразования T_e.

Допустим, утверждение теоремы верно для всех $k \leq n$. Рассмотрим произвольный граф потока управления D, содержащий $n + 1$ вершину. Сначала удалим из него все кратные ребра, так как они могут быть построены с помощью преобразования T_e. После этого удалим из графа все петли, так как они могут быть построены с помощью пребразования T_l. Получим, таким образом, граф D' без петель и кратных ребер. Теперь рассмотрим в данном графе произвольное ребро $e = v_1v_2$, не соединяющее глобальный вход с глобальным выходом.

Объединим вершины v_1 и v_2. Это означает, что нужно удалить ребро e, а вместо пары вершин v_1, v_2 оставить только одну вершину, которую обозначим v. Для этой вершины v должны выполняться следующие условия:

$$E^I(v) = (E^I(v_1) \cup E^I(v_2))\backslash\{e\},\ E^O(v) = (E^O(v_1) \cup E^O(v_2))\backslash\{e\}. \quad (7)$$

Полученный после объединения вершин v_1, v_2 граф обозначим D''.

После объединения вершин v_1, v_2 в графе D'' могли появиться петли. Так как глобальный вход CFG не может иметь входящих ребер, а глобальный выход – исходящих, то для обеспечения корректности графа D'' необходимо показать, что ни одна из образовавшихся петель не инцидентна глобальному входу или глобальному выходу. Рассмотрим петлю в графе D'' (это может быть только v_1v_1 или v_2v_2). Наличие такой петли означает, что либо в графе D' изначально была петля v_1v_1 или v_2v_2, либо петлей стало одно из ребер v_1v_2 или v_2v_1.

Так как из графа D были удалены все петли, то наличие ребер v_1v_1 или v_2v_2 в графе D' исключено. Так как из графа D были удалены все кратные ребра (в том числе и кратные ребру $e = v_1v_2$), то возможность возникновения петли в графе D' из ребра v_1v_2 исключена. Значит петля возникла из ребра v_2v_1. Таким образом, мы приходим к выводу, что граф D содержал ребра v_1v_2, v_2v_1, а значит, ни одна из вершин v_1, v_2 не является ни глобальным входом, ни глобальным выходом.

То есть граф D'' является корректным CFG, содержащим n вершин. По предположению индукции граф D'' может быть получен из D_0 с помощью последовательности преобразований T_e, T_l, T_r, а значит то же можно и сказать о графе D:

$$D_0 \overset{T_e,T_l,T_r}{\rightarrow} D'' \overset{T_r}{\rightarrow} D' \overset{T_e,T_l}{\rightarrow} D. \tag{8}$$

Так как верна база индукции и индуктивный переход, то утверждение теоремы верно для любого n.

С помощью описанного алгоритма можно создать граф потока управления произвольной структуры, при этом, варьируя частоту и очередность применения преобразований, можно получать графы произвольного размера, степени разветвленности и глубины гнезд циклов. Созданные таким образом модели CFG могут быть использованы для исследования глобальных оптимизаций.

3.3 Применение случайных графов потока управления для исследования оптимизации расположения в памяти линейных участкой программы

Для исследования оптимизации расположения в памяти линейных участков промежуточного представления были сгенерированы случайные графы, содержащие от 10 до 1000 вершин. К данным случайным графам была применена оптимизация. Эффективность оптимизации оценивалась по следующим характеристикам:

$$F = \frac{|E'|}{|E|},\ F_\omega = \frac{\sum_{e \in E'} \omega(e)}{\sum_{e \in E} \omega(e)}, \tag{9}$$

где E – множество всех провалов, E' – множество использованных провалов, $\omega(e)$ – счетчик провала e. Таким образом характеристика F представляет собой долю использованных провалов, а F_ω – долю использованных провалов с учетом профиля исполнения.

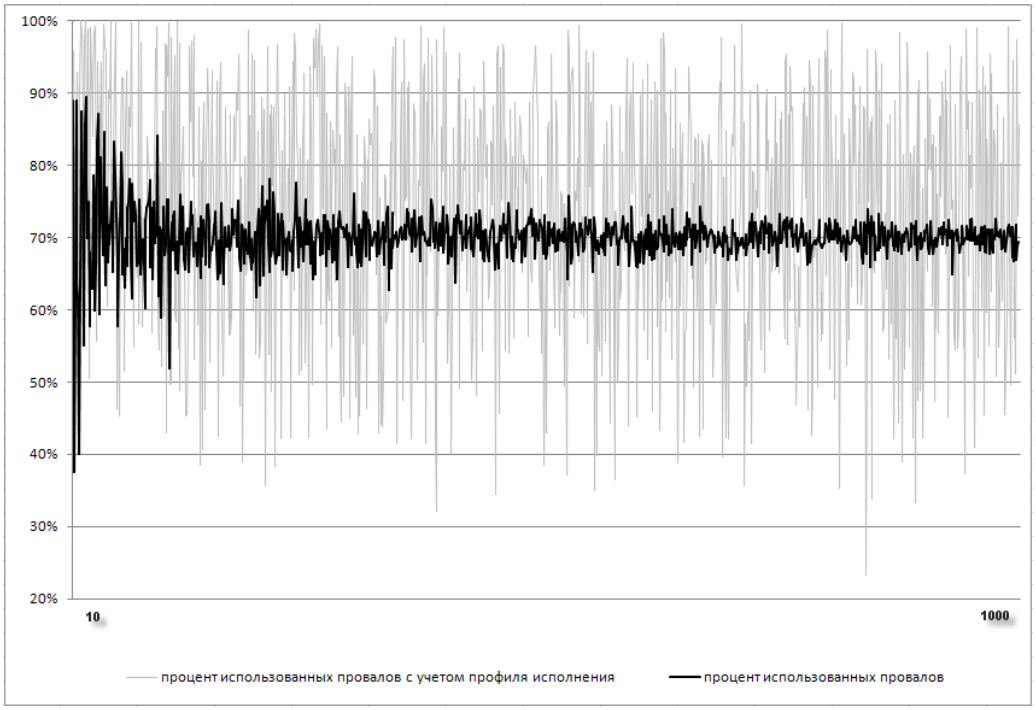

Рис. 2. Эффективность оптимизации использования провалов на случайных графах с количеством вершин от 10 до 1000.

На рисунке 2 приведен график зависимости величин F, F_ω от количества вершин графа. При этом видно, что величина F изменяется не слишком сильно, в отличие от величины F_ω, которая меняется в широких пределах, так как зависит от случайно распределенного профиля исполнения CFG.

Оптимизация расположения в памяти линейных участков промежуточного представления была реализована в компиляторе базового уровня [10] системы двоичной трансляции для архитектуры «Эльбрус» [11]. Для данной

оптимизации были получены характеристики F, F_ω на задачах пакета SPEC CPU2000 [12]. Результаты приведены на рисунке 3.

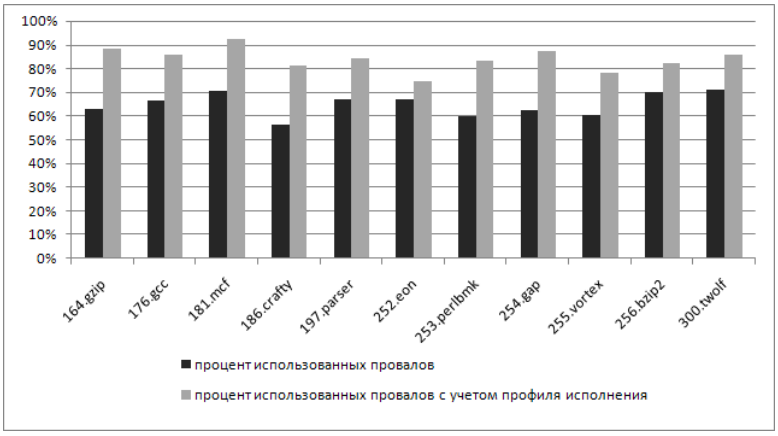

Рис. 3. Эффективность оптимизации использования провалов на CFG задач из пакета SPEC CINT2000.

По данным рисунков 2 и 3 видно, что предложенная модель случайного графа потока управления хорошо соотносится с графами потока управления для реальных задач и оценка эффективности анализируемой оптимизации на этой модели является адекватной.

4 Заключение

В статье предложена методика создания случайных графов потока управления, обладающих корректным профилем исполнения. Данная методика включает в себя генерацию случайного графа потока управления с помощью последовательного применения атомарных преобразований к элементарному графу. После применения каждого атомарного преобразования гарантируется корректность профиля исполнения. Полученная таким образом модель опробована для анализа эффективности оптимизации расположения в памяти линейных участков промежуточного представления, реализованной в системе двоичной трансляции для архитектуры «Эльбрус». Сравнение результатов применения оптимизации на случайных графах потока управления и на графах реальных задач показали адекватность модели.

Список литературы

[1] В. Ю. Волконский: Оптимизирующие компиляторы для архитектуры с явным параллелизмом команд и аппаратной поддержкой двоичной совместимости. *Информационные технологии и вычислительные системы*, 2004, № 3, С. 4-26.

[2] Prasad Kulkarni, Mattew Arnold, Michael Hind: Dynamic compilation: the benefits of early investing. *VEE'07 Proceedings of the 3rd international conference on Virtual execution environments*, 2007, P. 94-104.

[3] John Aycock: A brief history of just-in-time. *ACM Computing Surveys (CSUR)*, Volume 35 Issue 2, 2003, P. 97-113.

[4] L. Baraz, T. Devor, O. Etzion, S. Goldenberg, A. Skaletsky, Y. Wang, Y. Zemach: IA-32 Execution Layer: a two-phase dynamic translator designed to support IA-32 applications on Itanium-based systems. *Microarchitecture, MICRO-36, Proceedings, 36th Annual IEEE/ACM International Symposium*, 2003, P. 191-201.

[5] J. C. Dehnert, B. K. Grant, J. P. Banning, R. Johnson, T. Kistler, A. Klaiber, J. Mattson: The transmeta code morphing software: using speculation, recovery, and adaptive retranslation to address real-life challenges. *Proceedings of the International Symposium on Code Generation and Optimization*, 2003.

[6] S. Muchnick: Advanced Compiler Design and Implementation. *Morgan Kaufmann Publishers*, 1997.

[7] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ulman: Compilers: Principles, Techniques, and Tools. *Prentice Hall*, 2nd ed, 2006.

[8] А. А. Рыбаков: Анализ алгоритмов оптимизации расположения в памяти линейных участков программы. *Известия вузов. Электроника*, 2013, номер 1, С. 47-53.

[9] В. Н. Касьянов, В. А. Евстигнеев: Графы в программировании: обработка, визуализация и применение. *«БХВ-Петербург»*, 2003.

[10] А. А. Рыбаков, М. В. Маслов: Быстрый регионный компилятор системы двоичной трансляции для архитектуры «Эльбрус». *V Международная научно-практическая конференция «Современные информационные технологии и ИТ-образование», сборник избранных трудов*, под редакцией проф. В. А. Сухомлина, Москва, 2010, с. 436-443.

[11] Н. В. Воронов, В. Д. Гимпельсон, М. В. Маслов, А. А. Рыбаков, Н. С. Сюсюкалов: Система динамической двоичной трансляции x86 -> «Эльбрус». *Вопросы радиоэлектроники*, серия ЭВТ (электронная вычислительная техника), Москва, 2012, выпуск 3, с. 89-108.

[12] Standard Performance Evaluation Corporation, www.spec.org