УДК 004.4'422

Румянцев Л.А.
Московский физико-технический институт

Эффективное встраивание дополнительных потоковых оптимизаций в структуру бинарного транслятора

Задача заключается во встраивании в бинарный транслятор таких потоковых оптимизаций как локальные реассоциации и вынос инвариантов цикла [1]. Простой пример на рис.1 показывает, как эти оптимизации связаны друг с другом. Здесь локальные реассоциации преобразовывают две исходные операции сложения в две новые, у первой из которых оба аргумента - инварианты цикла. При этом регистр $v3$ сам становится инвариантом после выноса операции из цикла соответствующей оптимизацией. Это способствует тому, что оставшаяся в цикле операция будет снова преобразована с помощью локальных реассоциаций в сочетании со следующими операциями в цикле. То есть, эти оптимизации помогают друг другу распространятся вперед.

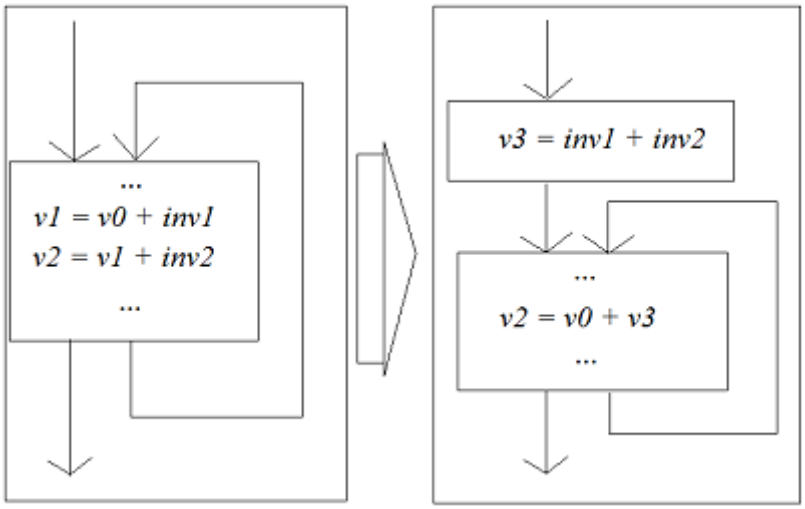

Рис. 1. Преобразование внутреннего представления: $v0$, $v1$, $v2$ – регистры, $inv1$, $inv2$ – инвариантные в цикле регистры, $v3$ – новый инвариантный в цикле регистр

Рассмотрим реализацию в целевом трансляторе оптимизаций потока данных осуществляемых при проходе внутреннего представления программы [2]. Некоторые оптимизации и сбор определённой вспомогательной информации можно осуществить только при проходе внутреннего представления снизу вверх (например, замена результата на результат из последующих операций, продвижение масок используемых битов). Другие только при проходе сверху вниз (замена на константу, продвижение масок известных битов). Из этого следует, что хотя бы по одному разу снизу вверх и сверху вниз внутреннее представление проходить обязательно. Для эффективной работы транслятора желательно, чтобы эти проходы осуществлялись *только* по одному разу. Это осуществимо с помощью следующей схемы. При проходе внутреннего преставления все оптимизации пытаются примениться к текущей операции. При анализе применимости оптимизаций учитываются предыдущие операции, уже оптимизированные. Это можно назвать «лавинным» эффектом распространения оптимизаций, когда оптимизации, проведённые на предыдущих операциях, позволяют выполниться оптимизациям на следующих операциях. Эта схема исключает последовательное применение каждой оптимизации с отдельным проходом по всему внутреннему представлению и повторением некоторых оптимизаций по несколько раз.

При таком подходе локальные оптимизации [1] следует реализовывать на проходе сверху вниз. Удобно разбить их на две части. Часть, которая обнаруживает конструкции подходящие под определённое правило и предлагает соответствующие трансформации. И часть, которая проверяет применимость рекомендованных первой частью трансформаций (например, можно ли продлить время жизни аргументов новых операций, полезность данной трансформации) и, если проверка проходит успешно, соответственным образом преобразует внутреннее представление. Локальные реассоциации легко реализуемы с помощью отдельного правила локальных оптимизаций. Связка локальные реассоциации – вынос инвариантов хорошо вписывается в схему прохода внутреннего представления сверху вниз. Позволяя срабатывать друг другу, как было описано вначале, они могут выносить из цикла целые цепочки связанных между собой операций.

При реализации локальных реассоциаций ограничителем эффективности работы является часть локальных оптимизаций, отвечающая за обнаружение правил, так как поиск этих правил будет осуществляться для *каждой* операции. Одной из главных задач

здесь является быстрое получение отрицательного ответа, если текущая операция не образует ни одно из правил. Второй является получение рекомендаций по трансформациям за константное время. Первая из этих задач реализуется с помощью расположения проверок на отрицательный ответ так, чтобы наиболее частые из них (например, проверка на имя операции, проверка, что рассматриваемая операция находится вне цикла) выполнялись раньше других. Если же правило было найдено, требуется быстро получить рекомендации по трансформациям. Реализовать этот поиск через условные операторы if-else, проверяющие имена операций из найденного правила и выдающие соответствующую рекомендацию, было бы не эффективно, так как количество таких проверок растет с количеством операций, к которым это правило применимо. Поэтому имеет смысл реализовать поиск результатов через таблицу индексируемую именами операций в правиле. При этом большая часть таблицы будет пустой, так как результаты существуют только для определённых сочетаний имен операций. Но суммарный размер таблицы получается около двадцати килобайт и по современным меркам не является большой платой за константное время доступа к результату.

При попытке выноса операции из цикла используется быстрый анализ наличия записи в регистр в подграфе. Для корректного применения, в этот анализ была добавлена функциональность, позволяющая игнорировать записи в указанной операции. В качестве игнорируемой указывается сама операция – кандидат на вынос. На данный момент в схеме используется 38 правил локальных реассоциаций, в планах еще 39 менее распространённых. При тестировании заметных замедлений в работе транслятора выявлено не было. Таким образом, добавляя новую функциональность в бинарный транслятор, разработчик должен учитывать все аспекты, влияющие на общую производительность, учитывать особенности текущей реализации и максимально использовать уже готовые инструменты.

СПИСОК ЛИТЕРАТУРЫ

1. *Muchnick S.S.* Advanced Compiler Design and Implementation. - San Francisco: Morgan Kauffman, 1997.
2. *Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д.* Компиляторы: принципы, технологии и инструментарий, 2-е изд. – М.: И.Д. Вильямс, 2008.