Методы коррекции профильной информации в процессе компиляции

О.А. Четверина <chetverina_o@mcst.ru>
АО "МЦСТ», 117105, Россия, г. Москва, ул. Нагатинская,дом 1.

Аннотация. Эффективность проводимых компилятором оптимизирующих преобразований может быть значительно повышена с помощью получения и использования профильной информации об исполнении программы, такую работу компилятора называют profile guided optimization (PGO). После проведения преобразований, изменяющих граф потока управления, необходимо скорректировать профильную информацию, с целью сохранения ее актуальности для качественной работы последующих оптимизаций. В работе рассматриваются два способа представления профильной информации и переход между ними, описаны возможные причины возникновения дополнительной информации о потоке управления в оптимизируемом коде, требующей изменения профиля. Рассмотрены наиболее часто возникающие задачи проведения коррекции профильной информации и предложены следующие решения: алгоритм коррекции значений счетчиков ациклического участка по значениям счетчиков узлов выхода; алгоритм коррекции значения среднего числа итераций цикла; алгоритм коррекции профиля при обнаружении «противоречивого узла». Доказано, что разработанные алгоритмы коррекции значений счетчиков ациклического участка и числа итераций цикла позволяют минимально изменить соотношение значений счетчиков исходного графа. На их основе описан механизм коррекции профильной и тонкой профильной информации после проведения преобразования откртки итераций цикла. Все предложенные методы реализованы и используются в процессе компиляции оптимизирующими компиляторами lcc, lccs, lfortran, lfortrans для архитектур Эльбрус и Спарк.

Ключевые слова: профиль исполнения задачи, PGO, коррекция профиля, коррекция числа итераций цикла, расширенная профильная информация, взвешенный граф

DOI: 10.15514/ISPRAS-2015-27(6)-4

Для цитирования: Четверина О.А. Методы коррекции профильной информации в процессе компиляции. Труды ИСП РАН, том 27, вып. 6, 2015 г., стр. 49-68. DOI: 10.15514/ISPRAS-2015-27(6)-4.

1. Введение

Существенную роль в достижении максимальной производительности современных вычислительных систем играют оптимизирующие компиляторы, которые путем применения оптимизирующих преобразований кода позволяют

повысить эффективность использования вычислительных устройств. Для улучшения работы ряда преобразований разработаны механизмы, при которых компиляторам наряду с исходным кодом передается еще и дополнительная информация об исполнении программы. Один из наиболее используемых и значимых соответствующих механизмов это сбор и дальнейшее применение информации о частоте исполнения тех или иных событий [1,2,3]. Чаще всего в этом качестве используются значения счетчиков, фиксирующих число исполнения узлов и дуг (условных переходов) графа потока управления, в дальнейшем - *профильная информация*. По ним, для удобства расчета эвристик оптимизаций, вычисляются вероятности перехода от одного узла к другому, среднее число итераций цикла и другие характеристики взвешенного графа управления. Используя профильную информацию, компилятор может принимать более эффективные решения при применении таких преобразований, как предподкачка кода, программная векторизация [4], конвейеризация цикла [5], вынос инвариантных вычислений и других, что дает существенный прирост в производительности. Чаще всего компиляция с профилем производится для не очень больших пользовательских приложений, поскольку в этом случае достаточно легко произвести представительный тренировочный запуск. Но в последнее время все больше внимания уделяется возможности оптимизации с использованием профиля (PGO или FDO) и в более сложных ситуациях: на данный момент разрабатываются способы решения задачи профилирования операционных систем [6]; активно ведутся работы по использованию и улучшению качества профилирования в Just-in-Time компиляторах [7].

В то же время, несоответствие соотношений счетчиков линейных участков, используемых компилятором и возникающих при последующем исполнении программы, может приводить к ухудшению планирования кода при проведении тех же оптимизаций. Так, при неверном значении среднего числа итераций, компилятор может отказаться от программной конвейеризации цикла с большим реальным числом итераций, или, наоборот, конвейеризовать малоитерационный цикл с созданием большого числа стадий [8]. Если же распределение счетчиков ветвлений ациклических участков перестанет соответствовать реальному исполнению, то ряд преобразований может привести к перемешиванию маловероятных и высоковероятных операций с откладыванием времени исполнения последних, либо к отказу от применения некоторых оптимизаций к высоковероятным участкам [9]. Причина неточности используемой компилятором профильной информации может быть связана либо с исходной неполнотой собранной информации, либо с уменьшением ее достоверности в процессе компиляции. С целью уменьшения ошибок оптимизаций из-за неполноты собранной профильной информации может быть использовано смешанное профилирование, при котором полученные в процессе реальной работы программы данные дополняются статистически построенным, то есть предсказанным, профилем для ранее не

выполнявшихся участков кода [7]. Для повышения точности такой профиль может корректироваться с учетом профиля реального исполнения соседних процедур [10]. Со стороны же выполнения самой компиляции важной задачей является максимально точная коррекция профиля после применения изменяющих граф потока управления преобразований, то есть сохранение ее актуальности для последующих оптимизаций. Стоит отметить, что кроме возможности использовать более эффективные эвристики отдельными оптимизациями, сохранение достоверной информации о числе исполнения операций в процессе компиляции дает возможность предсказывать время работы и сравнивать качество получившегося кода в результате применения всей оптимизационной последовательности без реального запуска на машине [11,12].

В представленной работе описаны основные способы представления и коррекции профильной информации, а также дополнительные числовые характеристики, которые влияют на профиль и могут приводить к необходимости его коррекции. Поставлен ряд соответствующих задач на графах с весами и приведены разработанные способы их решения.

2. Способы представления профильной информации

Везде далее под профильной информацией подразумеваются счетчики числа исполнения узлов и дуг управляющего графа. Связано использование именно счетчиков с тем, что их проще всего собрать при исполнении с помощью инструментированного кода исходной программы. Зачастую более удачным вариантом представления этой информации внутри компилятора с точки зрения применения оптимизаций являются вероятности перехода по дугам. Удобство использования вероятностей заключается в возможности быстрой оценки и сравнения различных дуг без необходимости проводить нормировку.

Переход от счетчиков узлов и дуг к вероятностям и обратно является взаимно-однозначным с точностью до значения стартового счетчика рассматриваемой процедуры. При переходе от счетчиков к вероятностям, вероятности вычисляются как отношение счетчиков исходящих дуг к сумме значений счетчиков исходящих дуг в каждом узле. Обратный переход, то есть вычисление счетчиков по вероятностям, несколько сложнее — для ациклических (без контуров) связанных участков он производится пропагацией счетчиков по вероятностям при обходе сверху вниз с топологической сортировкой [13] для уменьшения количества вычислений:

1. Все стартовые узлы с известными значениями счетчиков обозначим *{Ni}*. Обозначим *{NOi}* - узлы с непроставленными значениями счетчиков.

2. Проставим каждому ребру, исходящему из каждого узла *{Ni}* счетчик, равный вероятности этого ребра на значение счетчика соответствующего узла *Ni.*

3. Если в *{NOi}* не осталось узлов, исполнение алгоритма закончено.

Иначе, проверяем $\{NOi\}$ на наличие таких узлов, где значения счетчиков для всех входящих дуг проставлены. Поскольку граф ациклический, то такие найдутся. Проставляем им в качестве значений счетчиков суммы счетчиков всех входящих дуг. Перекладываем их в $\{Ni\}$. Переходим в 2.

В случае же работы с циклами используется следующий алгоритм:

1. Строится дерево циклов, в котором у каждого цикла есть только одна голова [14].

2. В случае нескольких входов в цикл обходом графа цикла сверху вниз для каждого входа в цикл вычисляется счетчик, который пройдет от него до головы цикла через все обратные дуги. Полученные счетчики добавляются к счетчику головы цикла, в результате получается стартовый счетчик головы цикла C0.

3. Проходом сверху вниз от головы цикла вычисляется суммарная вероятность p пройти по обратным дугам цикла. Тогда среднее число итераций цикла:

$$I = 1 + p + p^2 + \cdots + p^n + \cdots = 1/(1-p)$$

4. Значение счетчика головы умножается на вычисленное число итераций и с помощью алгоритма для ациклического участка осуществляется пропагация с учетом значений счетчиков дополнительных входов.

При проведении значительной части преобразований, меняющих граф потока управления, для сохранения информации о профиле достаточно выставить требуемую вероятность на сформированные дуги и провести описанную пропагацию счетчиков по всему графу, либо просто провести коррекцию по уже имеющимся вероятностям. К примеру, при проведении подстановки процедур вместо вызова в точку вызова производится копирование управляющего графа вызываемой процедуры с коррекцией значений его счетчиков с учетом значения счетчика подставляемого вызова.

Но в ряде случаев в процессе проведения оптимизирующих преобразований выявляется необходимость согласовывать профильную информацию в виде счетчиков узлов и дуг или вероятностей переходов с дополнительными числовыми характеристиками, касающихся профиля, которые могут быть:

- заданы пользователем (pragma среднего числа итераций для цикла, likely/unlikely для ветвления),
- дополнительно собраны (расширенная профильная информация, такая, как вероятности выйти на определенных итерациях цикла),
- установлены компилятором эвристически при дублировании участков кода — ациклических или циклов, при построении новых ветвлений,
- вычислены точно по имеющимся операциям и константам.

Далее будут рассмотрены и предложены решения наиболее часто встречающихся задач согласования различных профильных характеристик и предложены способы коррекции профиля: коррекция значений счетчиков ациклического участка по значениям счетчиков узлов выхода (глава 3), коррекция числа итераций цикла (глава 4), коррекция профиля при обнаружении «противоречивого перехода» (глава 5), вычисление числа итераций и коррекция профильной и тонкой профильной информации после проведения откртки итераций (глава 6).

3. Алгоритм коррекции счетчиков ациклического участка по заданному входному счетчику и счетчикам выходов

Решение описанной в этой главе задачи наиболее востребовано для представленной далее коррекции числа итераций цикла, но используется и для ациклических участков при коррекции счетчиков откручиваемых итераций цикла или необходимости выставить определенную суммарную вероятность перехода на некоторый участок из стартового узла при создании нескольких дуг с неопределенными значениями вероятностей.

Задача: Рассмотрим ориентированный согласованный односвязный взвешенный ациклический граф с одним доминирующим и n концевыми узлами. Обозначим вес доминирующего узла *C0*. Рассмотрим некоторый набор новых вещественных значений *C1, C2,...Cn* таких, что *C1+C2+...+Cn=C0*. Требуется скорректировать веса узлов таким образом, чтобы в концевых узлах *N1, N2, ... Nn* стояли значения *C1, ...Cn,* соответственно. При этом нужно минимально изменить соотношение счетчиков внутри участка. Под изменением счетчиков будем подразумевать максимум отношений старых и новых счетчиков цикла всех узлов рассматриваемого графа.

Для решения задачи предлагается использовать пропагацию счетчиков снизу вверх в пропорции значении счетчиков входящих дуг в порядке обратной топологической сортировки.

Алгоритм:

1. Пометим все узлы графа как узлы с непроставленными весами. Поставим в узлы *N1, N2, ... Nn* счетчики *C1, C2,...Cn* соответственно. Набор узлов с уже проставленными в них счетчиками обозначим *{Ni}*, соответствующий набор счетчиков - *{Ci}*. Для каждого узла графа запоминаем число исходящих дуг с непроставленными счетчиками *{EOi}*.

2. Если в *{Ni}* находится только доминирующий узел участка, то работа алгоритма закончена.

3. Для всех узлов *Nk* из *{Ni}* производим следующее:

Пусть в узел *Nk* входит *m* ребер, его новый вес равен Ck, а старые веса ребер, входящих в *Nk,* обозначаются *{EC(i)} i=1,.. m.*

Присвоим каждому входящему в *Nk* ребру *j* вес *Ck*(EC(j)/ECsum),* где $ECsum=\sum EC(i)$- сумма всех старых весов ребер, входящих в узел *Nk.*

При этом из значения *EOi* для узлов, из которых выходят соответствующие ребра, вычитаем 1 для каждого ребра. Если в результате для каких-то узлов значение *EOi* стало равным 0, то добавляем их в *{Ni}.*

Исключаем узел *Nk* из *{Ni}.*

Идем в 2.

Сложность описанного алгоритма совпадает с минимальной сложностью алгоритма обратной топологической сортировки и равно *O(Edges*), где *Edges* — число ребер в цикле.

Обоснование:

1) Счетчик в доминирущем узле равен сумме счетчиков концевых узлов из-за сохранения суммы потоков [15].

2) Покажем, что приведенный алгоритм удовлетворяет условию минимального изменения счетчиков.

Во-первых, заметим, что при проведении изложенной коррекции на ациклическом участке с уже проставленными счетчиками и вероятностями и использовании исходных концевых значений будет получен граф, совпадающий с исходным, поскольку в этом случае каждый шаг будет приводить к идентичным счетчикам узлов и дуг.

Далее, рассмотрим представление исходного взвешенного графа со счетчиками в виде суммы графов, каждый из которых представляет собой проведенную описанную выше пропагацию из одного из концевых участков для узлов, из которых достижим соответствующий концевой. Счетчики указанной суммы будут совпадать со значениями исходного графа, поскольку разделение счетчиков производится с одинаковыми коэффициентами для всех схождений. То есть значение счетчика каждого узла рассматриваемого графа *Nl* можно представить как сумму:

$$C(Nl) = \sum_{i \le n} p(i,l)Ci,$$

где *p(i,l)* — вероятность дойти от концевого узла *Nk* до узла *Nl* при проходе снизу вверх.

Пусть новые счетчики концевых узлов *Ck'=a(k)*Ck, a_min=min(a(k)), a_max=max(a(k).*

Тогда, аналогично предыдущему, новые счетчики внутренних узлов после описанной пропагации равны:

$$C'(Nl) = \sum_{i \le n} a(i)p(i,l)Ci',$$

то есть *C'(Nl)/C(Nl)<a_max* и *C(Nl)/C'(Nl)<1/a_min*.

Поскольку 1/*a_min* и *a_max* — требуемые при коррекции коэффициенты изменения концевых узлов, то есть некоторых узлов графа, то максимум отношений не может быть меньше соответствующий значений, что и требовалось доказать.

4. Коррекция счетчиков узлов цикла при изменении среднего числа итераций

При проведении некоторых оптимизации (peeling, loop unswitching, loop splitting by index и т.д. [16,17]), учете дополнительной информации (pragma loop count) кода, а так же для одновременного использования статистики о локальных вероятностях и числе итераций при построении статического профиля, требуется изменить число итераций цикла. Для этого необходимо скорректировать счетчики внутренних узлов и вероятностей перехода по ветвлениям между ними. При этом, чтобы получить требуемое число итераций I, необходимо установить общую вероятность дойти до обратных дуг цикла равной $\frac{1}{I}$. В случае единственной обратной дуги в цикле это можно сделать путем изменения только вероятности перехода по обратной дуге. В общем же случае, то есть при наличии нескольких обратных дуг, изменение вероятности одного выхода приводит к изменению значений счетчиков нескольких узлов. В частности, меняются счетчики, и, следовательно, вероятности дойти до других выходов. Нетрудно убедиться в том, что по этой причине задача коррекции вероятностей соответственно заданному числу итераций является нелинейной с ограничениями. Приведем алгоритм ее решения, имеющий сложность *O(Edges)*, где *Edges* — число ребер в цикле.

Поскольку отсутствует дополнительная информация об относительном изменении счетчиков узлов внутри цикла, следует наложить дополнительное ограничение на решаемую задачу. Потребуем, как и ранее, чтобы минимально изменились значения счетчиков узлов. В таком случае полученная профильная информация будет больше соответствовать действительности, и дальнейшие преобразования станут более эффективными и обоснованными.

Чтобы избавиться от необходимости решать нелинейную задачу с прямым вычислением требуемых вероятностей переходов, предлагается представить цикл со счетчиками как взвешенный ориентированный односвязный ациклический граф, у которого один узел является доминирующим, и есть несколько концевых узлов, то есть таких, из которых не выходят дуги. Часть этих узлов соответствует переходам по обратным дугам, часть - выходам из

цикла. В таком случае, число итераций цикла задается соотношением сумм счетчиков переходов и выходов. Кроме того, счетчики во внутренних узлах цикла должны быть согласованы, то есть для всех узлов, кроме доминирующего и концевых, сумма счетчиков входящих ребер должна быть равна сумме счетчиков исходящих ребер. Теперь для коррекции числа итераций цикла можно воспользоваться алгоритмом коррекции счетчиков ациклического участка.

Алгоритм:

Рассмотрим некоторый сводимый цикл. Обозначим его дуги выхода через *{Ki}* и обратные дуги через *{Bi}* . Пусть *C0* - счетчик входа в цикл, I' - среднее число итераций, которое получилось после откртуки. Сначала мы опишем как провести необходимые вычисления для случая, когда цикл не содержит других циклов, а затем режим задачу для случая вложенных циклов.

1. Вариант внутреннего цикла:

Узлы и дуги цикла приводится к ациклическому виду (Рис.1, преобразование (1)):

Строятся два дополнительных временных узла.

Один узел соединяется со всеми обратными дугами *{Bi}*, в него ставится счетчик *C0*(I'-1)*.

Другой узел соединяется со всеми дугами выхода *{Ki}* , в него ставится счетчик *C0* .

В качестве старых счетчиков для получившихся ребер используются старые счетчики выхода из цикла и старые счетчики обратных дуг.

Применяется алгоритм коррекции счетчиков в ациклическом участке, начиная с этих двух узлов и до головы цикла. В результате, в голове цикла мы получим счетчик *C0*I'* (Рис.1, преобразование (2)).

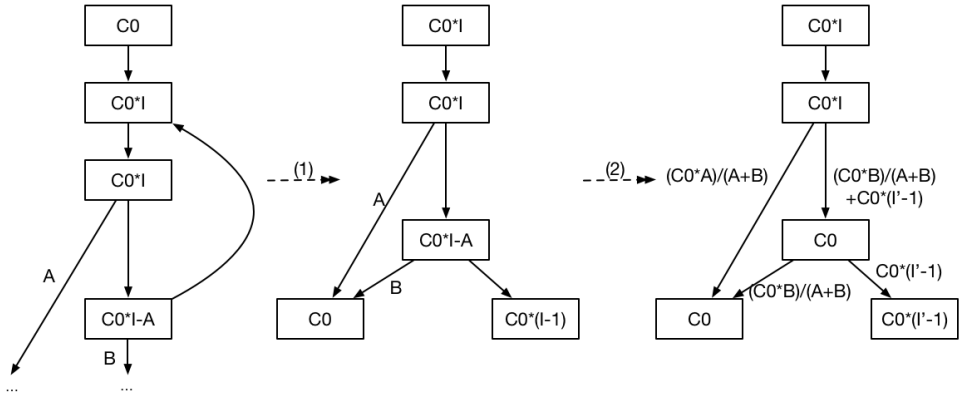

Рис. 1. Пример коррекции значений счетчиков узлов и дуг цикла

2. Вариант цикла, содержащего другие циклы:

Каждый внутренний цикл представляется как один узел, вход в который - вход в цикл, а выходы - обратные дуги цикла.

К внешнему циклу применяются операции, описанные в пункте 1.

Если после пересчета счетчиков внешнего цикла отношение счетчиков на дугах выхода внутреннего цикла не изменилось,

то все счетчики внутреннего цикла умножаются на коэффициент изменения, иначе внутренний цикл пересчитывается согласно пункту 1.

Заметим, что в случае, когда число итераций I' совпадает с начальным числом итераций I, счетчики и вероятности на соответствующих дугах останутся прежними.

5. Коррекция профиля при возникновении противоречивых переходов

При ряде оптимизирующих преобразований, таких как подстановка процедур, расщепление цикла по ветвлению, расщепление схождений производится дублирование участков графа управления или же *расщепление графа управления*. В этом случае часть узлов дублируется, и, поскольку отсутствует дополнительная информация о различии вероятностей аналогичных переходов между копиями узлов, на ветвления проставляются вероятности ветвлений исходного графа. В дальнейшем нередко оказывается, что за счет расщепления вероятность для некоторого перехода в одной из копий стала вычислимой. В этом случае может возникнуть противоречие между вероятностями на дугах и результатом вычисленного сравнения, или же *противоречивый переход,* то есть единственный выход из ненулевого по счетчику числа исполнения узла, имеющий вероятность 0. Пример возникновения такого противоречия приведен на Рисунках [2,3], после удаления невыполнимых дуг возникают узлы с только нулевыми исходящими дугами.

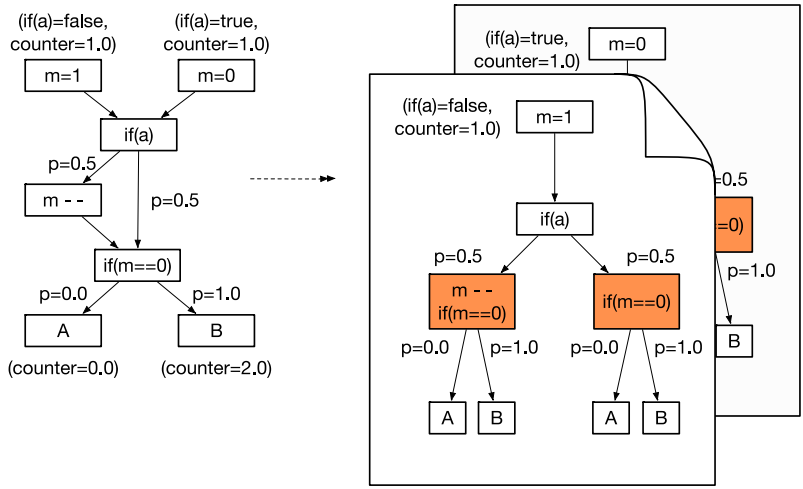

Рис.2. Применение преобразования расщепления с образованием вычислимых операций сравнения

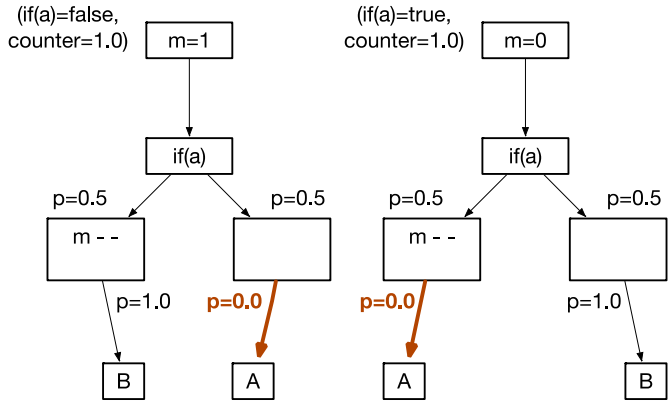

Рис.3. Противоречие после точного вычисления сравнения с удалением дуг

Причина возникающего противоречия заключается в том, что при реальном исполнении программы в узел с противоречивым ветвлением («противоречивый узел») никогда не был бы осуществлен переход, то есть все его счетчики и счетчики узлов, которые он постдоминирует, должны быть нулевыми. Правильная простановка вероятностей приведена на Рисунке 4.

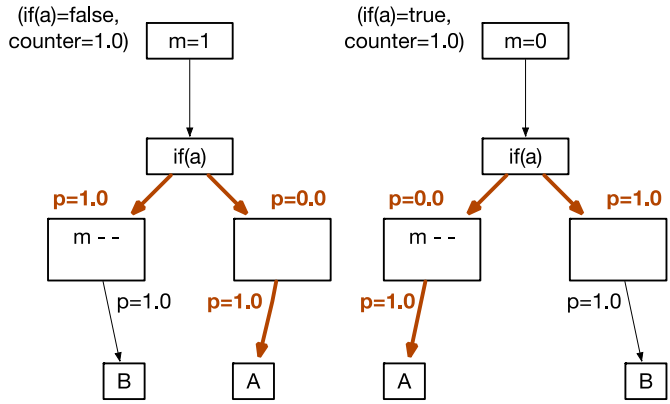

Рис. 4. Необходимая коррекция профиля

Следующий алгоритм позволяет исправить несоответствие и выявить ветвления, по которым должна быть выставлена нулевая вероятность перехода. При этом в порядке обратной топологической сортировки находятся узлы, которые постдоминирует противоречивый узел, их счетчики обнуляются и корректируются вероятности узлов, с ведущими в них ненулевыми переходами.

Алгоритм:

1. Добавим узел с возникшим противоречивым ветвлением к множеству {Node0}.
2. Выбираем произвольный узел N из {Node0}. Помечаем все входящие в него дуги.

   В узлы, из которых выходят эти дуги, записываем число выходящих из них не помеченных дуг.

   Если это число для узла = 0,

   то кладем соответствующий узел в {Node0} и удаляем его из NodeBorder, если он там есть (= если из него выходит больше одной дуги),

   иначе, если мы пометили первую дугу, то кладем соответствующий узел в {NodeBorder}.

   Удаляем узел N из {Node0}.
3. Если в {Node0} есть узел, то идем в 2.
4. Для всех узлов из {NodeBorder} делаем следующее: Между всеми не помеченными исходящими дугами узла распределяем вероятность 1.0 пропорционально начальным вероятностям дуг.

Если встретился узел, для которого все вероятности не отмеченных дуг равны 0, то кладем этот узел в {Node0}, идем в 2.

Если дошли сюда, то закончили работу.

6. Вычисление числа итераций цикла, коррекция профильной и тонкой профильной информации после преобразования открутки итераций цикла

Для эффективного проведения преобразования открутки итераций цикла используется расширенная профильная информация, или *тонкая профильная информация*, а именно, дополнительно к счетчикам числа исполнения узлов цикла собираются счетчики выхода на конкретной итерации для нескольких, далее – *K*, первых итераций цикла. Полученная информация позволяет достаточно точно определить оптимальное число итераций для открутки. При этом, кроме проведения самого преобразования, в ряде случаев существенным, с точки зрения производительности, является и использование имеющейся расширенной информации для коррекции профиля. Пример существенного различия для проведения последующих преобразований приведен на Рис.5 и Рис.6. В обоих случаях приведен пример идентичного с точки зрения профиля исходного цикла и существенно различающегося остатка после открутки одной итерации. Причина заключается в различии тонкой профильной информации — в обоих случаях среднее число итераций цикла равно 2, но в первом случае из цикла всегда осуществляется выход после второй итерации, во втором — в 90% случаев после первой и в 10% после 11-ой. В результате преобразования в первом случае остается цикл со средним числом итераций, равному 1, а во втором —10-ти.

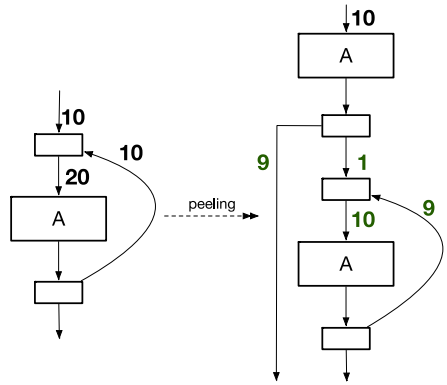

Рис.5. Результат применения открутки одной итерации в первом случае

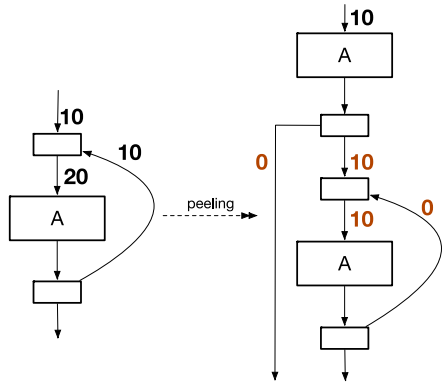

Рис.6. Результат применения открутки одной итерации во втором случае

Рассмотрим цикл со средним числом итераций I и вероятностями выхода на итерациях *p(k), k<=K,* и пусть при открутке было вынесено *m<=K* итераций. Тогда, с учетом тонкой профильной информации, меняется способ вычисления числа итераций нового цикла:

$$I' = \frac{(I - \sum_{k \le m} p(k) * k)}{(1 - \sum_{k \le m} p(k))} - m$$

С учетом наличия тонкой профильной информации меняется и механизм коррекции профиля вынесенных участков. Для каждой вынесенной итерации значения счетчиков и вероятностей пересчитываются поочередно. Как и в случае цикла строится два дополнительных узла - обобщенный выход и переход дальше. В обобщенный узел выхода ставится *C1= C0*p(k)/(1-p(1)-p(2)-.... - p(k-1))*, а в обобщенный узел перехода к следующей итерации ставится *C0-C1*. Далее, от этих узлов и до узла входа в итерацию применяется алгоритм коррекции счетчиков.

Кроме того, требуется пересчитать и тонкую профильную информацию для возможности ее дальнейшего использования. Рассмотрим цикл, у которого выносится *m* итераций.

а) При *k<=K-m* вероятности выхода из цикла точно вычисляются. Пусть $P1 = \sum_{k \le K} p(k)$ - вероятность не попасть в новый цикл, тогда вероятности выйти на итерациях нового цикла $p'(k) = p(k+m)/(1-P1)$.

б) При K-m<k<=K распределение вероятностей выходов по итерациям неизвестно. Поэтому распределим вероятности выхода таким образом, чтобы они были согласованы с числом итераций нового цикла, и чтобы при этом не менялись вероятности выхода при k<=K-m. Число итераций нового цикла:

$$I' = \sum_{k<\infty} p'(k) * k = \sum_{k \le K-m} p'(k) * k + \sum_{k>K-m} p'(k) * k \quad (1)$$

Обозначим $I1'$ и $I2'$ первое и второе слагаемое соответственно. Значения I' и $I1'$ известны, и задача состоит в том, чтобы получить вероятности $p'(k)$ из $I2'$.

Введем их сумму

$$P2' = \sum_{k>K-m} p'(k) = 1 - \sum_{k\le K-m} p'(k)$$

и произведем замену

$$l=k-(K-m),\ p''(l)=p'(l+K-m) \quad (2),$$

тогда

$$P2' = \sum_{l>1} p''(l) (3).$$

Из (1), (2) и (3) следует

$$I2' = (K-m) * P2' + \sum_{l\ge 1} p''(l) * l.$$

Таким образом, нужно подобрать такие $p''(l)$, чтобы

$$\sum_{l\ge 1} p"(l) * l = I2' - (K-m) * P2' = S \quad (4),$$

и при этом выполнялось (3).

Удачное решение дает распределение Пуассона, характеризуемое функцией вероятности $q(k) = a^k e^{-a}/k!$. Причина выбора именно этого распределения заключается в том, что оно соответствует одинаковой вероятности выйти из цикла для каждой последующей итерации, при условии прохода через голову цикла. Его первый момент, $\sum_{l\ge 1} q(l) * l$, равен a . Чтобы им воспользоваться, нам нужно скорректировать вероятности в (4), поскольку $P2'$ не равно единице. Для этого произведем замены $q(k)=p''(l)/P2'$ и $T=S/P2'$.

Получаем при $K-m<k<=K$ вероятности выхода:

$$p'(k) = P2' * q(l) = P2' * T^l * e^{-T}/l!,$$

где $P2' = 1 - \sum_{k\le K-m} p'(k)$, $T = \frac{(I-\sum p'(k)*k)}{P2'} - K - m$, $l=k-(K-m)$.

7. Заключение

В статье рассматривается проблема сохранения актуальности профильной информации в процессе компиляции. Описаны основные способы хранения информации о числе исполнения участков кода, взаимосвязь между ними и дополнительные характеристики, требующие коррекции профиля после проведения различных преобразований и анализов кода. Представлен алгоритм коррекции счетчиков ациклического участка по выходам, алгоритм

коррекции числа итераций цикла, механизм коррекции профиля при возникновении противоречивых переходов и решены задачи, возникающие при проведении преобразования открутки с использованием тонкой профильной информации.

Все предложенные методы реализованы и используются в процессе компиляции оптимизирующими компиляторами lcc, lccs, lfortran, lfortrans для архитектур Эльбрус и Спарк.

Список литературы

[1]. Chang P. P., Mahlke S. A., Hwu W. W. Using profile information to assist classic compiler code optimizations. Software Practice and Experience, V. 21, No12. -1991.-P. 1301-1321

[2]. W. Chen, R. Bringmann, S. Mahlke, S. Anik, T. Kiyohara, N. Warter, D. Lavery, W. - M. Hwu, R. Hank and J. Gyllenhaal., Using profile information to assist advanced compiler optimization and scheduling, 1993

[3]. Jan Hunicka, Profile driven optimisations in GCC, Proceedings of the GCC Developers' Summit June 22nd-24th, 2005, Ottawa, Ontario Canada

[4]. Волконский В. Ю., Ермолицкий А. В., Ровинский Е. В. Развитие метода векторизации циклов при помощи оптимизирующего компилятора. Высокопроизводительные вычислительные системы и микропроцессоры: сборник трудов ИМВС РАН, Выпуск N8, 2005.

[5]. Dmitry M Maslennikov, Vladimir Y Volkonsky: Compiler method and apparatus for elimination of redundant speculative computations from innermost loops. Elbrus International October 9, 2001: US06301706

[6]. Pengfei Yuan, Yao Guo , Xiangqun Chen, Experiences in profile-guided operating system kernel optimization, Proceedings of 5th Asia-Pacific Workshop on Systems, June 25-26, 2014, Beijing, China

[7]. Bo Wu, Mingzhou Zhou, Xipeng Shen, Yaoqing Gao, Raul Silvera, Graham Yiu, Simple profile rectifications go a long way, Proceedings of the 27th European conference on Object-Oriented Programming, July 01-05, 2013, Montpellier, France

[8]. Дроздов А.Ю., Степаненков А.М. Технология оптимизации цикловых участков процедур в компиляторах для архитектур с аппаратной поддержкой конвейризации циклов. Информационные технологии и вычислительные системы №3, М. 2004

[9]. Иванов Д.С. Распределение регистров при планировании инструкций для VLIW-архитектур. Программирование, № 6, 2010, С.74-80

[10]. Bo Wu, Zhijia Zhao, Xipeng Shen, Yunlian Jiang, Yaoqing Gao, Raul Silvera, Exploiting inter-sequence correlations for program behavior prediction, Proceedings of the ACM international conference on Object oriented programming systems languages and applications, October 19-26, 2012, Tucson, Arizona, USA [doi>10.1145/2384616.2384678]

[11]. Calin Cascaval , Luiz De Rose , David A. Padua , Daniel A. Reed, Compile-Time Based Performance Prediction, Proceedings of the 12th International Workshop on Languages and Compilers for Parallel Computing, p.365-379, August 04-06, 1999

[12]. Jeremy Lau , Matthew Arnold , Michael Hind , Brad Calder, Online performance auditing: using hot optimizations without getting burned, Proceedings of the 2006 ACM

SIGPLAN conference on Programming language design and implementation, June 11-14, 2006, Ottawa, Ontario, Canada
[13]. Ананий В. Левитин. Алгоритмы: введение в разработку и анализ — М.: «Вильямс», 2006.— С. 220-224. — ISBN 5-8459-0987-2
[14]. Steven S. Muchnick, Advanced Compiler Design & Implementation, 2003
[15]. Introduction to algorithms, Third Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, 2009
[16]. Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman "Compilers: Principles, Techniques, and Tools" Book, Pearson Education, ISBN 0-321-48681-1, 2006

Methods of Profile Information Correction during Compilation

O.A. Chetverina <chetverina_o@mcst.ru>
AO "MCST», 117105, Russian Federation, Moscow, 1 Nagatinskaya Str.

Abstract. Performed by compiler code optimizing transformations efficiency can be greatly increased by obtaining and using program profile information, such compiler work is called profile-guided optimization (PGO). After applying transformations, which change control flow graph, it is necessary to adjust the profile information in order to maintain its adequacy for the work of succeeding optimizations. This paper considers two ways of representing profile information and the transition between them, describes the possible causes of uprising additional information on optimized code control flow requiring profile updating. Most frequently encountered problems of profile information update are considered and following solutions are offered: algorithm of acyclic graph profile correction according to its end nodes values; algorithm of loop average iteration number correction; control flow with "controversial node" profile correction algorithm. It is proved that the suggested algorithms of acyclic region counters and the loop iterations counter correction allow to change the counters ratio with original graph counters minimally. With the use of proposed algorithms, the mechanism of correction profile and thin profile information after loop peeling is described. All described methods are implemented in optimizing compilers lcc, lccs, lfortran, lfortrans for Elbrus and Sparc architectures.

Keywords: program profile, PGO, profile correction, loop iterations counter correction, extended profile information, weighted graph

DOI: 10.15514/ISPRAS-2015-27(6)-4

For citation: Chetverina O.A. Methods of Profile Information Correction during Compilation. *Trudy ISP RAN/Proc. ISP RAS,* vol. 27, issue 6, 2015, pp. 49-66 (in Russian). DOI: 10.15514/ISPRAS-2015-27(6)-4

References

[1]. Chang P. P., Mahlke S. A., Hwu W. W. Using profile information to assist classic compiler code optimizations. Software Practice and Experience, V. 21, No12. -1991.-P. 1301-1321

[2]. W. Chen, R. Bringmann, S. Mahlke, S. Anik, T. Kiyohara, N. Warter, D. Lavery, W. - M. Hwu, R. Hank and J. Gyllenhaal., Using profile information to assist advanced compiler optimization and scheduling, 1993

[3]. Jan Hunicka, Profile driven optimisations in GCC, Proceedings of the GCC Developers' Summit June 22nd-24th, 2005, Ottawa, Ontario Canada

[4]. Volkonskiy V.Y., Ermolitskii A.V., Rovinskii E.V. Razvitie metoda vektorizacii ciklov pri pomoshchi optimiziruyushchego kompilyatora. [Development of loops vectorization method using an optimizing compiler]. Vysokoproizvoditel'nye vychislitel'nye sistemy i mikroprocessory: sbornik trudov IMVS RAN [High-performance computing systems and microprocessors: a collection of IMVS RAS works], Vypusk N8, 2005 (in Russian)

[5]. Dmitry M Maslennikov, Vladimir Y Volkonsky: Compiler method and apparatus for elimination of redundant speculative computations from innermost loops. Elbrus International October 9, 2001: US06301706

[6]. Pengfei Yuan, Yao Guo , Xiangqun Chen, Experiences in profile-guided operating system kernel optimization, Proceedings of 5th Asia-Pacific Workshop on Systems, June 25-26, 2014, Beijing, China

[7]. Bo Wu, Mingzhou Zhou, Xipeng Shen, Yaoqing Gao, Raul Silvera, Graham Yiu, Simple profile rectifications go a long way, Proceedings of the 27th European conference on Object-Oriented Programming, July 01-05, 2013, Montpellier, France

[8]. Drozdov A.YU., Stepanenkov A.M. Tekhnologiya optimizacii ciklovyh uchastkov procedur v kompilyatorah dlya arhitektur s apparatnoj podderzhkoj konvejrizacii ciklov. [Procedures loop areas optimization technology for architectures with hardware loops pipelining] Informacionnye tekhnologii i vychislitenye sistemy.[Information Technology and computer systems] №3, M. 2004 (in Russian)

[9]. D.S. Ivanov, Register allocation with instruction scheduling for VLIW-architectures. Programming and Computer Software. November 2010, Volume 36, Issue 6, pp 363-367

[10]. Bo Wu, Zhijia Zhao, Xipeng Shen, Yunlian Jiang, Yaoqing Gao, Raul Silvera, Exploiting inter-sequence correlations for program behavior prediction, Proceedings of the ACM international conference on Object oriented programming systems languages and applications, October 19-26, 2012, Tucson, Arizona, USA [doi>10.1145/2384616.2384678]

[11]. Calin Cascaval , Luiz De Rose , David A. Padua , Daniel A. Reed, Compile-Time Based Performance Prediction, Proceedings of the 12th International Workshop on Languages and Compilers for Parallel Computing, p.365-379, August 04-06, 1999

[12]. Jeremy Lau , Matthew Arnold , Michael Hind , Brad Calder, Online performance auditing: using hot optimizations without getting burned, Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation, June 11-14, 2006, Ottawa, Ontario, Canada

[13]. Ananias V. Levitin . Algorithms : Introduction to the design and analysis [Algoritmy: vvedenie v razrabotku i analiz] - M .: "Williams", 2006.— C. 220-224. — ISBN 5-8459-0987-2 (in Russian)

[14]. Steven S. Muchnick, Advanced Compiler Design & Implementation, 2003

[15]. Introduction to algorithms, Third Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, 2009

[16]. Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman "Compilers: Principles, Techniques, and Tools" Book, Pearson Education, ISBN 0-321-48681-1, 2006