Технология оптимизации цикловых участков процедур в компиляторах для архитектур с аппаратной поддержкой конвейеризации циклов.

Дроздов А. Ю., Степаненков А. М.,

ИМВС РАН, г. Москва

{sasha|stepa_am}@mcst.ru

Аннотация

В работе предлагается технология оптимизации цикловых участков процедур, основанная на архитектурной поддержке конвейеризации циклов. Она включает в себя различные оптимизации, технологические фазы, алгоритмы планирования циклов методом наложения итераций и распределения цикловых регистров. Данная технология является альтернативой программным методам обработки циклов и во многих случаях более эффективна по сравнению с ними.

Введение

В статье описывается компиляторная технология, основанная на архитектурной поддержке конвейеризации циклов. Ключевыми необходимыми свойствами таких архитектур являются возможность исполнения операций в конвейерном режиме, наличие спекулятивного режима исполнения операций [4] и наличие вращающихся (или цикловых) регистров – регистров с несколькими поколениями. Передача значения циклового регистра от младшего поколения к старшему происходит автоматически на каждой итерации цикла. Примерами подобных архитектур являются Intel Itanium, Intel Itanium 2 [5], Эльбрус-3М [6,7]. Указанная технология была отработана в компиляторном проекте для архитектуры Эльбрус-3М и получила название OVERLAP-технологии. OVERLAP включает в себя различные оптимизации: специализированные и стандартные [1,9], технологические фазы [10], алгоритмы планирования циклов методом наложения итераций и распределения цикловых регистров [8]. OVERLAP является альтернативой программным методам обработки циклов [3]. Подобные методы налагают меньшие требования на возможности архитектуры (в частности, не обязательны вращающиеся регистры). Основой программных методов совмещения итераций является дублирование тела цикла. OVERLAP позволяет получать более оптимальный код цикла, не прибегая к дублированию.

1. Класс допустимых циклов.

Опишем класс циклов, для которых применима технология OVERLAP. Будем называть такие циклы *наложенными*. Перечислим основные свойства, которым должны удовлетворять наложенные циклы.

1) Тело наложенного цикла можно преобразовать в один узел управляющего графа [1,10].

2) Наложенный цикл не содержит вызовы процедур.
3) У наложенного цикла может быть только один выход и один переход на голову (*закручивающий* переход) [1].

Раскроем более подробно первое требование. Для архитектур, не допускающих условное выполнение операций, оно фактически означает, что цикл должен состоять из одного узла или цепочки узлов, не являющихся глобальными метками [1]. Для архитектур, поддерживающих предикатные вычисления, данное ограничение можно переформулировать следующим образом.

Утверждение 1. Для преобразования тела цикла в один узел достаточно выполнения следующий условий:

а) цикл не должен содержать других циклов, т.е. должен быть самым вложенным;

б) цикл не должен содержать динамических переходов (в частности переключателей);

в) цикл не должен содержать глобальных меток;

г) цикл должен быть сводимым [1];

д) цикл не должен содержать операции, имеющие побочный эффект вне контекста цикла (не допускающие спекулятивное выполнение), для которых архитектурой не предусмотрено условное выполнение, и находящихся в узлах, не постдоминирующих [1] голову цикла.

Отметим, что третье требование не является принципиальным, а носит лишь технический характер. Цикл с несколькими закручивающими переходами и выходами всегда трансформируется в эквивалентный, но с одним выходом и одним закручивающим переходом.

С помощью оптимизирующих преобразований можно расширять класс наложенных циклов, снимая описанные ограничения. Вызовы процедур (требование 2) можно устранять с помощью подстановки процедур [1]. Если узел с вызовом процедуры, вложенный цикл или узел, содержащий переключатель, не постдоминируют голову цикла и имеют существенно меньшее число повторений по сравнению с головой цикла, их можно исключить с помощью выделения в теле цикла горячего региона и оформления его в виде вложенного цикла (оптимизация nesting [1]). Существует достаточное количество алгоритмов сведения циклов [1]. Есть и другие цикловые оптимизации, позволяющие расширить класс наложенных циклов, которые будут описаны ниже.

2. Время исполнения наложенного цикла.

Пусть каждому переходу в управляющем графе поставлено в соответствие неотрицательное вещественное число – среднее число повторений этого перехода. Определим время исполнения произвольного узла N управляющего графа G. Пусть $Br_1,\ldots,Br_m$ - переходы узла N. $C_1,\ldots,C_m$ - средние числа повторений соответствующих переходов. Пусть также каждому переходу поставлено в соответствие неотрицательное целое число – относительное время начала исполнения (*время запуска*) перехода в узле N. Обозначим $T_1,\ldots,T_m$ - времена запуска переходов $Br_1,\ldots,Br_m$. Тогда время исполнения T_N узла N определяется по формуле $T_N = \sum_{j=1}^{m} C_j(T_j+1)$. Содержательно, время исполнения узла означает следующее. Если в качестве числа повторений перехода взять реальное число исполнений перехода на некоторых входных данных задачи, а в качестве времени запуска взять относительный номер такта планирования данного перехода в узле N, то время исполнения T_N есть время работы части кода программы, соответствующего узлу N, запущенной на тех же входных данных. Время исполнения цикла есть сумма времён исполнения узлов, принадлежащих циклу.

При фиксированном соответствии между переходами и средними числами их повторений время исполнения узла является функцией от времён запуска переходов этого узла. Пусть $O_1,\ldots,O_k$ - операции узла N. Рассмотрим функцию $S(O_1,\ldots,O_k)$, принимающую значения в Z_+^k - пространстве k-векторов с целыми неотрицательными координатами. Функция S, ставящая в соответсвие каждой операции неотрицательное целое число, интерпретируемое как время начала исполнения (*время запуска*) операции, называется *функцией планирования* или просто *планированием* узла N. Следовательно, T_N можно считать функцией от планирования S: $T_N = T_N(S)$. Планирование называется *оптимальным*, если на нём T_N достигает минимума. Оптимальное планирование всегда существует и находится перебором. Планирование и, соответственно, время исполнения в такой интерпретации зависят от параметров архитектуры, ширины командного слова, длины задержек между операциями, количества доступных регистров и т.д. Оптимальное планирование, при котором учитываются только длины задержек между операциями, называется *идеальным* планированием. Это планирование соответствует идеальной архитектуре с неограниченными ресурсами. Для каждой операции на основании задержек между операциями определяется время раннего её исполнения [2].

Утверждение 2. Идеальное планирование существует. При идеальном планировании времена запуска переходов с ненулевым числом повторений определены однозначно и равны временам раннего.

Из определения идеального планирования видно, что оно не может быть хуже любого другого планирования и, соответственно, время исполнения узла для идеального планирования (*идеальное время*) можно использовать в качестве нижней оценки времени исполнения узла при работе с любой архитектурой с заданными длинами задержек между операциями.

Теперь определим время исполнения наложенного цикла. Сначала на примере покажем, что время исполнения наложенного цикла может быть меньше идеального времени исполнения цикла. Рассмотрим цикл, состоящий из последовательности операций, представленной на рис. 1.

```
for ( i = 100, j = 0; i  > 0; i -- )
{
    a[j] = a[j] / i;
     j++;
 }
```

```
BEG
READ(R0,R1) -> R2
DIV(R2,R3) -> R2
WRITE(R0,R1,R2)
CMPL(R4,R3) -> P0
ADD(R1,4) -> R1
SUB(R3,1) -> R3
BRANCH (BEG, P0)
END
```

Рис. 1. Наложенный цикл и соответствующая ему последовательность операций.

В данном примере введём следующие длины задержек *l*(oper1,oper2), где oper1, oper2 – операции: *l*(BEG, READ) = 0, *l*(BEG,CMPL) = 0, *l*(READ,DIV) = 2, *l*(DIV,WRITE) = 10, *l*(WRITE,ADD) = 0, *l*(CMPL,SUB) = 0, *l*(CMPL,BRANCH) = 3, *l*(ADD,BRANCH) = 0, *l*(SUB,BRANCH) = 0, *l*(BRANCH,END) = 0.

Время раннего перехода BRANCH равно времени раннего ADD, равно времени раннего WRITE и равно *l*(BEG,READ)+*l*(READ,DIV)+*l*(DIV,WRITE)=12. Число повторений BRANCH равно 99, число повторений неявного перехода, соответствующего выходу из цикла рано 1. Тогда идеальное время исполнения цикла из приведённого примера равно 1200. Теперь рассмотрим один из вариантов планирования этого цикла как наложенного

для архитектуры, допускающей упаковку перечисленных операций в одно командное слово. На рис. 2 представлена та же последовательность операций цикла с разбиением на *стадии* и с распределением цикловых регистров по результату планирования.

```
_________Stage 1__________
BEG
READ(R0,L0[1]) -> L1[0]
CMPL(R4,L2[1]) -> LP0[0]
ADD(L0[1],4) -> L0[0]
SUB(L2[1],1) -> L2[0]
_________Stage 2__________
_________Stage 3__________
DIV(L1[2],L2[3]) -> L3[0]
_________Stage 4__________
_________Stage 5__________
_________Stage 6__________
_________Stage 7__________
_________Stage 8__________
_________Stage 9__________
_________Stage 10_________
_________Stage 11_________
_________Stage 12_________
_________Stage 13_________
WRITE(R0,L0[13],L3[10])
BRANCH(BEG,LP0[12])
END
```

Рис. 2. Последовательность операций наложенного цикла с разбиением на стадии.

Смысл стадий состоит в том, что на их границе происходит передача значений между поколениями цикловых регистров: $L_j[i] = L_j[i-1]$, $\forall j$. Размер стадии равен числу инструкций, необходимых для упаковки операций цикла. Указанный цикл спланирован в 13 стадий. Функция планирования наложенного цикла определяется по аналогии с функцией планирования узла. Результатом функции планирования наложенного цикла является вектор упорядоченных пар <t,s>, где t – неотрицательное положительное число (время запуска операции), s – натуральное число (номер стадии операции). Теперь найдём время исполнения цикла. Для получения значения аргументов операций с каждой стадии, в общем случае, необходимо выполнения операций со всех предыдущих стадий. Поэтому операции с последней стадии, соответствующие первой итерации цикла, могут исполняться спустя 12 тактов работы цикла. Эти 12 тактов работы наложенного цикла называются его *прологом*. В общем случае, если цикл спланирован в n команд и m стадий, прологом являются первые (m-1)n тактов его работы. Операции последней стадии, соответствующие 2-ой итерации цикла исполняются в 14-ом такте, 3-ей – в 15-ом и т.д. Таким образом, время одного исполнения наложенного цикла складывается из пролога и произведения среднего числа повторений цикла на число команд, требующихся для его упаковки. Время исполнения наложенного цикла равно произведению времени одного исполнения на сумму повторений всех переходов на этот цикл, отличных от закручивающего. В нашем примере это время равно $1*((13-1)*1+100*1)=112$ тактов. Таким образом, время исполнения наложенного цикла может быть меньше идеального

времени исполнения, в приведённом примере время исполнения наложенного цикла в 10 раз меньше идеального времени исполнения.

3. Нижняя оценка времени исполнения наложенного цикла.

В предыдущем параграфе была получена формула для времени исполнения наложенного цикла. При фиксированном соответствии между переходами и числами повторений время исполнения наложенного цикла есть функция от планирования $T_{ovl} = T_{ovl}(S)$. Планирование формирует *размер* наложенного цикла n – число команд и число стадий. Отметим, что число стадий зависит от размера наложенного цикла. Зафиксируем размер наложенного цикла n. *Идеальным* числом стадий для размера цикла n назовём число $(T_e(Br)+n-1)/n$, где $T_e(Br)$ – время раннего закручивающего перехода цикла.

Утверждение 3. При фиксированном n число стадий планирования цикла с размером n не меньше идеального числа стадий для размера n.

Таким образом, задача получения нижней оценки времени исполнения наложенного цикла сводится к получению нижней оценки на размер цикла.

Размер наложенного цикла в первую очередь зависит от параметров архитектуры. Пусть n_a - минимальное число команд, необходимое для упаковки операций цикла (*размер цикла по ресурсам*).

Утверждение 4. Для наложенного цикла $n_a \le n$.

В примере из предыдущего параграфа в теле цикла присутствовали операции, вырабатывающие значения, которое сами потребляли на следующей итерации (ADD и SUB).

Операция ADD со следующей итерации не может выполниться раньше, чем сформируется значение, которое она вырабатывает на текущей итерации, т.е. размер наложенного цикла не может быть меньше задержки l(ADD,ADD) = 1. В общем случае размер наложенного цикла не превышает *длины максимальной рекуррентности* цикла. Рекуррентность – это сильно связная компонента в графе зависимостей [2] наложенного цикла. Граф зависимостей наложенного цикла отличается от графа зависимостей скалярного участка наличием зависимостей по обратной дуге (*обратных* зависимостей). Определим понятие длины рекуррентности. Рекуррентность называется рекуррентностью класса 1, если существует замкнутый путь в рекуррентности, содержащий лишь одну обратную зависимость. Пусть определены рекуррентности класса k-1. Рекуррентностью класса k называется рекуррентность, в которой есть путь, содержащий в точности k обратных зависимостей, и не являющийся композицией путей, содержащих меньше, чем k обратных зависимостей. Одна рекуррентность может принадлежать сразу нескольким классам, в то же время в силу конечности множеств операций и зависимостей, для любого цикла существует константа K такая, что не существует рекуррентности класса с номером, большим K. Пусть R – некоторая рекуррентность цикла, принадлежащая классам k_1, …, k_r. Пусть l_1, …, l_r - длины максимальных путей, содержащих k_1, …, k_r обратных зависимостей. Тогда длиной рекуррентности R называется $l_R = \max_i \frac{l_i}{k_i}$ В цикле из примера имеются две рекуррентности {ADD} и {SUB}, обе класса 1 и длины 1. Максимальной рекуррентностью называется рекуррентность с максимальной длиной. Пусть n_r - длина максимальной рекуррентности наложенного цикла, либо 0, в случае отсутствия рекуррентностей.

Утверждение 5. Для наложенного цикла $n_r \le n$.

Рассмотрим ещё одно ограничение на размер наложенного цикла. Для этого проследим за выполнением операций READ и WRITE из приведённого выше примера. Обе операции выполняются, как и остальные операции цикла, 112 раз. Цикл же имеет число повторений 100. Следовательно, 12 раз эти операции выполняются, в общем случае, с некорректными аргументами. Так как операция READ спланирована на первой стадии, то для неё эти исполнения выпадают на последние итерации цикла, в то время как для WRITE – на первые итерации цикла. Операция READ может исполняться спекулятивно, т.к. не несёт никакого побочного эффекта (напомним, что наличие спекулятивного режима исполнения – одно из обязательных требований к архитектуре). Операцию WRITE нельзя спекулятивно исполнять, т.к. она вызывает побочный эффект. Таким образом, необходимо отменить исполнение операцииWRITE на первых 12 итерациях цикла. Это можно сделать, используя аппаратную поддержку, или программно, заведя счётчик пролога, и поставив операцию WRITE под предикат «FALSE, пока пролог». Заметим, что операцию WRITE можно планировать только на последнюю стадию, т.к. в общем случае невозможно сформировать предикат, отменяющий выполнение операции на нескольких последних итерациях цикла, используя разумное количество вычислений. Таким образом, ограничение формулируется следующим образом. Существует класс операций (вызывающих побочный эффект), которые требуют планирования на последней стадии. Определим для наложенного цикла число n_s. Пусть op_1, …, op_d - операции, требующие планирования на последней стадии (по определению считаем закручивающий переход операцией с последней стадии, поэтому $d > 0$), тогда $n_s = \max_i (T_e(Br) - T_i^l + 1)$, где $T_e(Br)$ – время раннего закручивающего перехода, T_i^l - время позднего [2] операции op_i, $i = 1,..,d$.

Утверждение 6. Для наложенного цикла $n_s \le n$.

Действительно, размер последней стадии не может превышать n_s, в то же время размер наложенного цикла равен размеру любой его стадии по определению.

Из утверждений 3-6 следует теорема о нижней оценке времени исполнения наложенного цикла.

Теорема. Пусть L – наложенный цикл, C_L - сумма повторений всех переходов на цикл, отличных от закручивающего, A_L - среднее число повторений цикла, $n_a(L)$ – размер цикла по ресурсам, $n_r(L)$ – длина максимальной рекуррентности, $n_s(L)$ – минимальный размер последней стадии, $T_e(L)$ - время раннего закручивающего перехода, Тогда время исполнения наложенного цикла $T_L \ge C_L \max\{T_e(L), A_L \max\{n_a(L), n_r(L), n_s(L)\}\}$; если $A_L \ge T_e(L)$, то $T_L \ge C_L \left[\frac{T_e(L)}{\max\{n_a(L), n_r(L), n_s(L)\}}\right] [-1 + A_L) \max\{n_a(L), n_r(L), n_s(L)\}$

Теорема о нижней оценке очень важна в OVERLAP-технологии. Она используется в алгоритме планирования наложенного цикла и является основной целевой функцией всех оптимизаций, применяющихся к наложенному циклу. Далее мы рассмотрим основные оптимизации, направленные на минимизацию функции T_L.

4. Оптимизации наложенных циклов.

Наложенные циклы можно разбить на два класса. К первому относятся те, для которых доминирует $T_e(L)$ из формулы нижней о нижней оценке времени исполнения. Как правило, это циклы с малым числом повторений A_L и слабо распараллеленные. Оптимизация таких циклов практически ничем не отличается от оптимизации

ациклических участков, и направлена на уменьшение длины критического пути [2], т.е. на уменьшение $T_e(L)$. Для нас интерес будут представлять наложенные циклы из второго класса, для которых длина критического пути $T_e(L)$ не даёт большого вклада во время их исполнения. Основной задачей оптимизаций в таком случае является минимизация констант цикла $n_a(L)$, $n_r(L)$, $n_s(L)$.

4.1. Ресурсные оптимизации.

Рассмотрим основные оптимизации, предназначенные для минимизации $n_a(L)$. Ключевой задачей таких оптимизаций является сокращение числа операций цикла. Здесь основным методом оптимизации является расщепление цикла на несколько циклов, для каждого из которых число команд, необходимых для упаковки тела цикла, строго меньше числа команд для исходного цикла.

Оптимизация nesting формирует в теле цикла регион, имеющий большое число повторений, оформляя его в виде вложенного цикла. Во вложенный цикл не попадают маловероятные вычисления исходного цикла. Таким образом, новый цикл концентрирует в себе все вычисления исходного, но пакуется в меньшее число команд.

Оптимизация расщепления цикла по условию разбивает цикл в общем случае на три цикла: исходный цикл без условия и альтернатив, и два цикла, содержащих альтернативы. В двух частных случаях эта оптимизация наиболее эффективна, т.к. не требует построения дополнительных операций в телах полученных циклов. Это случаи инвариантного условия и некоторый класс условий на индуктивную переменную цикла, имеющую нижнюю и верхнюю границы [1]. В первом случае инвариантное условие выносится из тела цикла, а цикл дублируется. В одной копии условие считается истинным, в другой ложным. Во втором случае цикл разбивается на два цикла, отличающихся от исходного границами изменения индуктивной переменной, и в каждом из которых отсутствуют вычисления, стоящие под условием выхода значения индуктивной переменной за свои границы.

Из оптимизаций низкого уровня повышается значение эквивалентных преобразований, сокращающих число операций, - сбор общих подвыражений, удаление избыточных пересылок и чтений из памяти и др. При этом не допускаются оптимизации, сокращающие критический путь, которые приводят к дублированию кода.

Для наложенных циклов с большим числом повторений, содержащих мало вычислений, важную роль играет оптимизация раскрутки цикла. Цель оптимизации состоит в минимизации отношения $\frac{n_a^{(k)}}{k}$, где k – степень раскрутки цикла, $n_a^{(k)}$ - число инструкций, необходимых для планирования цикла при степени раскрутки k.

Для циклов с большим числом повторений наиболее эффективно применение OVERLAP, т.к. эффект от пролога в них минимален. Поэтому для наложенных циклов важны оптимизации, увеличивающие среднее число повторений цикла. К паре внутренний цикл, охватывающий цикл иногда можно применить оптимизацию перестановки циклов. Суть её состоит в перестановке индуктивных переменных охватывающего и внутреннего циклов. Следовательно, если охватывающий цикл повторялся в среднем больше чем внутренний, то после перестановки большее число повторений будет иметь внутренний (наложенный) цикл. Ещё более эффективной в этом случае является оптимизации слияния охватывающего и внутреннего циклов, суть которой в замене двух индуктивных переменных на одну, которая пробегает объединение областей изменения исходных переменных. Число повторений нового цикла равно произведению чисел повторения охватывающего и внутреннего циклов.

4.2. Оптимизации рекуррентностей.

Время исполнения наложенного цикла зависит от длины максимальной рекуррентности. В некоторых случаях длина максимальной рекуррентности может существенно превышать размер цикла по ресурсам и размер его последней стадии, т.е. вносить основной вклад во время исполнения цикла. Простейший способ оптимизации рекуррентности – это применение всего арсенала классических оптимизаций, предназначенных для сокращения критических путей в графах зависимостей, на операциях рекуррентности, взяв в качестве целевой функции длину рекуррентности. Далее рассмотрим оптимизации, приводящие к исчезновению рекуррентностей.

Большинство рекуррентностей являются «искусственными», т.е. возникают из-за не существующих в действительности зависимостей между операциями. Основная их часть – это зависимости по памяти для тех случаев, когда классический индексный анализ не может определить ни зависимость, ни независимость между двумя контекстными операциями. Разрыв таких зависимостей может привести к исчезновению сильно связных компонент в графе зависимостей (рекуррентностей).

Наиболее эффективным способом разрыва статически неразрешённых зависимостей между парами контекстных операций является построение динамических проверок вне тела цикла, выполнимость которых обеспечивает независимость нужных пар контекстных операций. В случае невыполнимости управление передаётся на копию цикла, для которого соответствующие пары контекстных операций считаются зависимыми.

В случаях, когда невозможно осуществить проверки вне тела цикла, для любой пары (чтение, запись) можно разорвать зависимость динамическим сравнением адресов на равенство. В случае выровненных адресов достаточно построить не более 5 операций, чтобы установить независимость операций. Это операции сложения двух составляющих адреса для каждой операции (их может и не быть, если у адресов операций одна составляющая); операция наложения маски (побитовое И) на адрес операции с меньшим форматом – выравнивание к большему формату (её может не быть в случае совпадения форматов контекстных операций); операция сравнения на равенство результатов указанных выше операций; операция перехода под предикатом сравнения на компенсирующий код, в котором осуществляется коррекция результатов вычислений в предположении зависимости контекстных операций. Компенсирующий код состоит из копий операций, зависящих от операции чтения, спланированных выше перехода на компенсирующий код и для которых процесс восстановления в компенсирующем коде возможен. Наиболее эффективно применение данной оптимизации при наличии в аппаратуре специальной поддержки. В архитектуре Эльбрус-3М для одной операции чтения достаточно построения двух дополнительных операций для разрыва зависимостей со всеми конфликтующими записями в память.

4.3. Уменьшение длины стадии.

Как следует из теоремы о нижней оценке размер цикла и время его исполнения могут определяться размером последней стадии. Напомним, что на последней стадии планируются операции, имеющие побочный эффект. В большинстве случаев это операции записи в память. Разность между временем закручивающего перехода и временем позднего [2] операции записи в память не может быть меньше размера стадии. Следовательно, для минимизации этой разницы необходимы оптимизации, умеющие разрывать зависимости между операцией чтения и преемниками в графе зависимостей, тем самым, увеличивая её время позднего и уменьшая ограничение на размер последней стадии.

Преемниками операции записи в графе зависимостей часто становятся операции чтения из памяти. В случае несовпадения адресов, т.е. в том случае, когда индексный

анализ не смог определить факта зависимости или независимости пары (запись, чтение), применяются оптимизации, описанные в предыдущем параграфе. Случай совпадения адресов рассмотрим более подробно. Имеются следующие возможные ситуации. Операция записи доминирует операцию чтения. Если формат операции записи не меньше формата операции чтения, то применяется классическая оптимизация удаления избыточных чтений из памяти, а в качестве результата чтения берётся записываемое значение. В противном случае зависимость между чтением и записью разрывается, но в нужную часть результата операции чтения дополнительно вставляется записываемое значение. Если операция записи не доминирует чтение, то в наложенном цикле это означает, что она стоит под предикатом. Этот случай отличается от предыдущего тем, что соответствующие операции пересылки (вставки) записываемого значения помимо предиката операции чтения должны иметь предикат операции записи, т.е. выполняться под предикатом логическое «И» указанных выше предикатов.

Ещё один тип зависимостей для операции записи в память – антизависимость для одного из её аргументов. Это означает, что есть некоторая операция oper_a -> R0, пишущая в регистр, который операция записи использует в качестве аргумента. Данная зависимость разрывается с помощью её переноса на операцию пересылки. Действительно, построив операцию пересылки значения регистра R0 в некоторый новый регистр R10: ADD(R0,0) -> R10 и заменив аргумент операции записи R0 на R10, можно перенести антизависимость от операции oper_a с операции записи на операцию пересылки ADD.

4.4 Откртка пролога наложенного цикла.

Выше были описаны основные оптимизации, использующиеся в технологии OVERLAP, наиболее эффективные для циклов со средним и большим числом повторений. Для циклов с малым числом повторений, как правило, большой вклад во время исполнения цикла вносит критический путь цикла или, другими словами, длина его итерации, как скалярного участка. Мы отмечали, что для оптимизации критического пути наложенного цикла можно использовать классические оптимизации. Однако для наложенных циклов есть специфическая оптимизация, позволяющая сократить время его исполнения. Речь идёт о частичной откртке наложенного цикла, т.е. о выполнении части или всех операций пролога цикла до начала работы наложенного цикла. Данное преобразование очень эффективно при наличии больших ресурсных резервов перед входом в наложенных цикл, т.е. большого количества незаполненных, полупустых или пустых инструкций. Откртить можно как весь пролог, так и лишь его начальную часть, но единицей откртки должна быть целая стадия. При этом, если откручивается i-ая стадия, то вместе с ней следует откртить один раз все стадии $j < i$.

Например, пусть имеется цикл, спланированный в 5 стадий размера 1. Пролог такого цикла занимает 4 итерации. Счётчик пролога равен 4. Допустим, мы хотим сократить счётчик пролога до 1. Для этого до цикла необходимо выполнить операции с первой стадии первой итерации цикла, операции с первой стадии второй итерации цикла и со второй стадии первой итерации цикла, операции с первой стадии третьей итерации цикла, со второй стадии второй итерации цикла и с третьей стадии первой итерации цикла. Отличаются открученные операции с одной стадии и разных итераций номерами поколений цикловых регистров. После этого можно считать, что пролог наложенного цикла стоит из одной инструкции.

Заметим, что для архитектур, в которых не все операции допускают предикатный режим исполнения, например, операции записи, требующие, как было установлено выше, отмены в прологе цикла, откртка пролога наложенного цикла является необходимым техническим преобразованием.

5. Технология отката.

На примере цикла из параграфа 2 было показано, насколько эффективна OVERLAP-технология. Однако в параграфе 1 были сформулированы основные ограничение для наложенных циклов. Одно из самых жёстких – представление тела цикла в виде одного узла, это требует технология планирования наложенных циклов. В связи с этим ограничением возникает вопрос, а не являются ли программные методы оптимизации более эффективными для какого-то подкласса класса наложенных циклов.

Утверждение 7. Для любого вещественного $R \geq 1$ существует цикл со средним числом повторений, равным R, время исполнения которого как не наложенного меньше времени исполнения как наложенного.

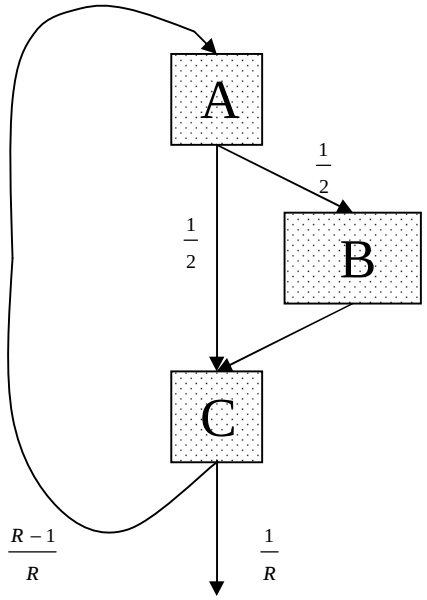


Рис. 3. Пример наложенного цикла.

Действительно, рассмотрим цикл, приведённый на рис. 3. Среднее число повторений такого цикла равно R. Рассмотрим оптимальное планирование этого цикла. Пусть T'_{AB} - время запуска перехода из узла A в узел B, T'_{AC} - время планирования перехода из узла A в узел C, T'_{BC} - время запуска перехода из B в C, T'_{CA} - время планирования перехода из C в A, и T'_{exit} - время запуска выхода из цикла. Определим $T_{AB}=T'_{AB}+1$, $T_{AC}=T'_{AC}+1$, $T_{BC}=T'_{BC}+1$, $T_{CA}=T'_{CA}+1$, $T_{exit}=T'_{exit}+1$. Без ограничения общности будем считать сумму повторений внешних переходов на голову цикла равной 1. Тогда оптимальное время исполнения цикла как не наложенного равно $T_L = R(\frac{T_{AC}+T_{AB}+T_{BC}}{2})+(R-1)\,T_{CA}+T_{exit}$.

Возьмём в качестве узла B узел, содержащий абсолютно параллельные вычисления, т.е. каждая операция, отличная от перехода, начальной и конечной операций узла, не зависит ни от какой другой операции, отличной от перехода, начальной и конечной операций узла. Для такого узла время оптимального планирования его единственного перехода на единицу меньше минимального числа инструкций, необходимых для упаковки операций узла. Теперь оценим время исполнения наложенного цикла, используя теорему о нижней оценке. Размер наложенного цикла не меньше ресурсного ограничителя, т.е. минимального числа инструкций, необходимых для упаковки операций цикла, которое, в

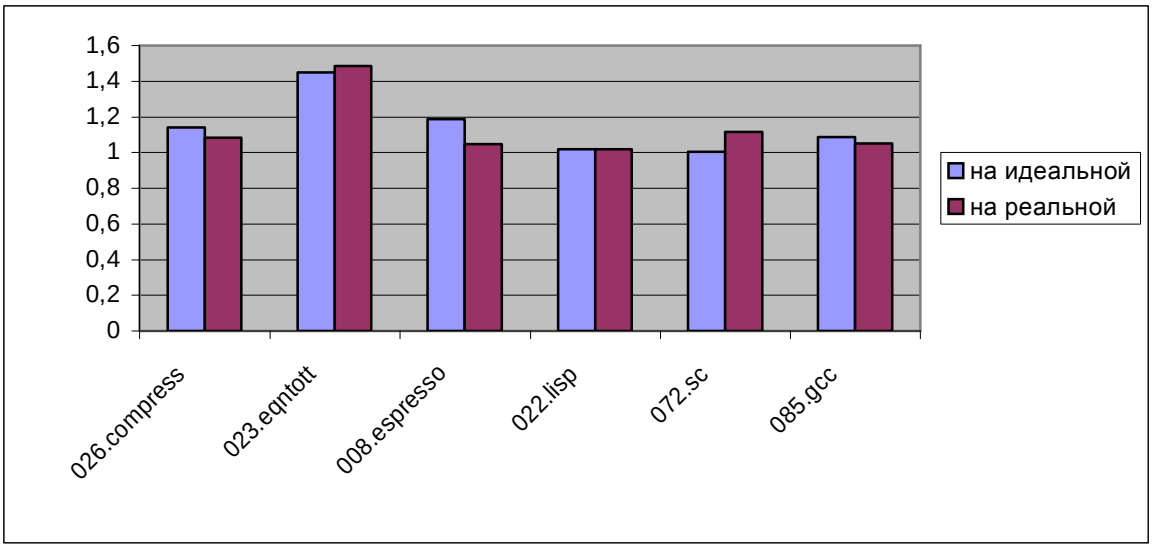
силу выбора узла B, не меньше T_{BC}. Таким образом, время исполнения наложенного цикла $T_L^{ovl} \geq RT_{BC}$. Далее будем добавлять в узел B операции то тех пор, пока не будет выполнено: $T_{BC} > T_{AB} + T_{AC} + 2T_{CA} + 2T_{exit}$. Тогда $T_L < R(\frac{T_{AC} + T_{AB} + T_{BC} + 2T_{CA} + 2T_{exit}}{2}) < RT_{BC}$ и, следовательно, $T_L^{ovl} \geq RT_{BC} > T_L$.

Доказательство утверждения выявляет подкласс наложенных циклов, для которых описываемая технология может давать плохие результаты – это циклы с несбалансированными близкими по вероятности альтернативами. С другой стороны, наложенные и неналоженные циклы по-разному оптимизируются, при планировании как наложенного, так и неналоженного циклов используются не переборные, а приближённые алгоритмы. Поэтому во многих случаях трудно предсказать, как целесообразнее обрабатывать цикл: с помощью аппаратной поддержки или программными методами. Для решения этой проблемы в технологии OVERLAP предусмотрен откат. Для каждого наложенного цикла на ранней стадии компиляции делается его точная копия, которая не считается наложенным циклом. Наложенный цикл и его копия проходят через основные фазы компиляции, однако к первому применяется технология OVERLAP, а второй обрабатывается всеми известными программными методами. После планирования наложенного цикла и его копии становятся известными времена их исполнения. В результате выбирается цикл с меньшим временем исполнения, а цикл с большим временем исполнения удаляется из кода.

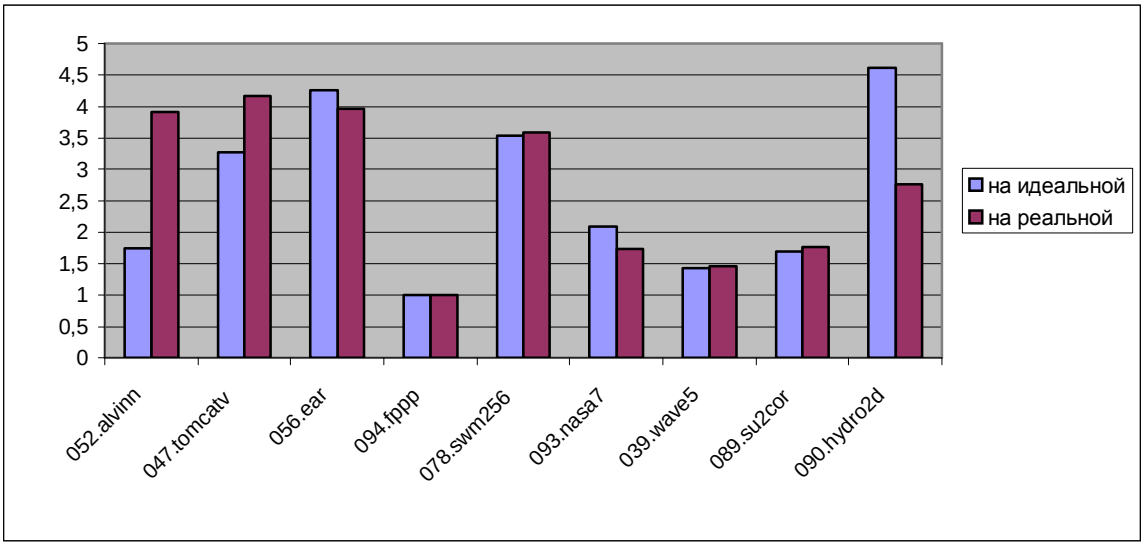
Описанный метод отката является хорошим выходом при принятии решения о применении к некоторому объекту программы сложных компиляторных технологий, так как во многих случаях невозможно дать оценку эффективности той или иной технологии, не применив её в полном объёме.

6. Экспериментальные результаты.

В предыдущих параграфах была описана технология оптимизации циклов, основанная на аппаратной поддержке. Были рассмотрены примеры циклов, для которых технология очень эффективна, были построены и контрпримеры. На рис. 4 приведена статистика эффективности технологии на задачах пакета spec92 отдельно для целочисленных задач и задач, содержащих вычисления над числами с плавающей точкой. В ней для каждой задачи пакета указано отношение её времени исполнения в тактах на симуляторе машины Эльбрус-3М в случае, когда задача компилировалась с применением программных методов оптимизации циклов, к времени исполнения её кода, полученного компилятором с применением технологии OVERLAP. Симулятор моделировал машину с идеальной подсистемой памяти (все чтения и записи попадают в кэш первого уровня) и машину с реальной подсистемой памяти. Видно, что технология OVERLAP наиболее эффективна для задач, основное время исполнения которых занимают циклы с достаточно большим числом повторений, содержащие операции с длинными задержками (вычисления, использующие числа с плавающей точкой). Это тесты 052.alvinn – 090.hydro2d. Среди целочисленных задач лучший показатель имеет 023.eqntott, так как более 90% времени его исполнения занимает цикл, для которого применима OVERLAP-технология.



Целочисленные задачи пакета spec92.



Задачи пакета spec92, использующие вычисления над числами с плавающей точкой.

Рис. 4. Сравнение времени исполнения задач пакета spec92 с применением программных методов оптимизации циклов и с применением технологии OVERLAP на симуляторе машины Эльбрус-3М с идеальной и реальной подсистемами памяти.

Заключение

В данной работе предложена технология оптимизации цикловых участков процедур для архитектур с аппаратной поддержкой конвейеризации циклов. Использование этой технологии вместо известных на сегодняшний день программных методов позволяет компилятору получать существенно более эффективный код для задач, содержащих большое количество вычислений в циклах.

Список литературы.

1. Steven S. Muchnick, “Advanced Compiler Design and Implementation”, Morgan Kauffman, San Francisco, 1997
2. J.R. Ellis, “Bulldog: A Compiler for VLIW Architectures”, Doctoral Dissertation, MIT Press Cambridge MA 1985.
3. Alexander Aiken, Alexandru Nicolau, Steven Novack. Resource-Constrained Software Pipelining, IEEE Transactions on Parallel and Distributed Systems, December 1995, pp. 1248-1270
4. M. S. Schlansker, B. R. Rau. EPIC: An Architecture for Instruction-Level Parallel Processors: Technical Report HPL-1999-111 / Compiler and Architecture Research Hewlett-Packard Laboratories, Palo Alto, February 2000.
5. Walter Triebel. Itanium Architecture for Software Developers. Intel Press, 2000.
6. Babayan B. A. E2k Technology and Implementation. // Proceedings of the Euro-Par 2000 – Parallel Processing: 6th International. – V. 1900/2000. – January, 2000. – P. 18-21.
7. K. Dieffendorf. The Russians Are Coming. Supercomputer Maker Elbrus Seeks to Join x86/IA-64 Melee // Microprocessor Report, V. 13, № 2. February 15, 1999. P. 1-7.
8. A. E. Eichenberger, E. S. Davidson, “Stage Scheduling: A Technique to Reduce the Register Requirements of a Modulo Schedule”, Proceedings of the 28-th Annual IEEE/ACM International Syposium in Microarchitecture, pp 338-349, November 1995.
9. D.F. Bacon, S. L. Graham, O.J. Sharp, “Compiler Transformations for High-Performance Computing”, ACM Computing Surveys, Vol. 26, No 4, December 1994.
10. J. R. Allen, K. Kennedy, C. Porterfield, and J. Warren. Conversion of control dependence to data dependence // In Proceedings of the Tenth Annual ACM Symposium on the Principles of Programming Languages, January 1983, P. 177-189