

Технология оптимизации циклов для архитектур с аппаратной поддержкой конвейеризации

А.Ю. Дроздов, А. М. Степаненков

Аннотация. В работе предлагается технология оптимизации цикловых участков процедур, основанная на архитектурной поддержке конвейеризации циклов. Она включает в себя различные типы 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 оптимизирующая компиляторная технология, основанная на архитектурной поддержке конвейеризации циклов. Необходимыми свойствами архитектуры в данном случае являются возможность исполнения операций в конвейерном режиме, наличие спекулятивного режима исполнения операций [4] и наличие вращающихся или цикловых регистров. Эти регистры образуют систему с несколькими поколениями, в которой передача значения регистра от младшего поколения к старшему происходит автоматически в каждой итерации цикла. Примерами подобных архитектур являются Intel Itanium, Intel Itanium 2 [5]. Технология, получившая название OVERLAP, была отработана в компиляторном проекте, выполненном в процессе создания вычислительных средств серии «Эльбрус». Она 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 представляющей собой альтернативу программным методам обработки циклов [3], которые предъявляют меньшие требования к возможностям архитектуры (в частности, не обязательны вращающиеся регистры). Основой программных методов совмещения итераций является дублирование тела цикла, тогда как OVERLAP позволяет получать более оптимальный код цикла, не прибегая к дублированию.

1. Класс наложенных циклов

Технология OVERLAP распространяется на класс циклов, которые можно назвать «наложенными». Они должны удовлетворять следующим требованиям:

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

Наложённый цикл не содержит вызовы процедур.

У наложенного цикла может быть только один выход и один «закручивающий» переход к голове цикла [1].

Первое требование применительно к архитектурам, не допускающим условное выполнение операций, фактически означает, что цикл должен состоять из одного узла или цепочки узлов, не

являющихся глобальными метками. Для архитектур, поддерживающих предикатные вычисления, данное ограничение можно сформулировать в виде следующего утверждения.

Утверждение 1. Для преобразования тела цикла в один узел достаточно выполнить следующие условия:

- цикл не должен содержать вложенных циклов;
- цикл не должен содержать динамических переходов (в частности переключателей);
- цикл не должен содержать глобальных меток;
- цикл должен быть сводимым [1];
- цикл не должен содержать операции, имеющие побочный эффект вне контекста цикла (то есть операции, не допускающие спекулятивное выполнение), если
 - для них архитектурой не предусмотрено условное выполнение,
 - они находятся в узлах, не постдоминирующих голову цикла.

Отметим, что третье требование не является принципиальным и имеет лишь технический характер. Цикл с несколькими закручивающимися переходами и выходами всегда трансформируется в эквивалентный, но с одним выходом и одним закручивающимся переходом.

Согласно [1], с помощью оптимизирующих преобразований можно расширять класс наложенных циклов, снимая приведенные выше требования. Вызовы процедур (требование 2) можно устранять с помощью подстановки процедур. 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). Существует достаточное количество алгоритмов сведения циклов. 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.

Тогда,

если типы переходов узла перенумерованы от 1 до m ,

T_1, \dots, T_m - относительные времена исполнения (времена запуска) переходов каждого типа,

C_1, \dots, C_m - числа, задающие среднее число повторений перехода каждого типа,

то время исполнения T_N узла N определяется формулой

$$T_N = \sum_{j=1}^m C_j (T_j + 1).$$

Время исполнения узла следует трактовать следующим образом:

если в качестве числа повторений перехода данного типа в узле N взять реальное число его исполнений на некоторых входных данных задачи, а в качестве времени запуска взять связанный с ним относительный номер такта планирования,

то время исполнения T_N есть время работы части соответствующего узлу N кода программы, 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, \dots, O_k - операции узла N . Рассмотрим функцию $S(O_1, \dots, O_k)$, принимающую значения в Z_+^k - пространстве k -векторов с целыми неотрицательными координатами. Функция S , которая ставит в соответст-

вие каждой операции неотрицательное целое число, интерпретируемое как 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 так далее. Оптимальное планирование, при котором учитываются только длины задержек между операциями, называется *идеальным* планированием. Оно 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 время запуска переходов определенного типа, имеющих серию повторений, ассоциируется с первым переходом этой серии (в дальнейшем для краткости он будет обозначаться как «ранний»).

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, характеризующей 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.

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

Рис. 1. Наложённый цикл и соответствующая ему последовательность операций

Обозначим длину задержки между операциями $oper1$ и $oper2$ через $l(oper1,oper2)$. Для данного примера зададим:

$l(BEG, READ) = 0$, $l(BEG, CMPL) = 0$, $l(READ, DIV) = 2$, $l(DIV, WRITE) = 10$, $l(WRITE, ADD) = 0$, $l(CMPL, SUB) = 0$, $l(CMPL, BRANCH) = 3$, $l(ADD, BRANCH) = 0$, $l(SUB, BRANCH) = 0$, $l(BRANCH, END) = 0$.

Тогда время раннего перехода BRANCH равно:

времени раннего ADD,

времени раннего WRITE

и сумме $l(BEG, READ) + l(READ, DIV) + l(DIV, WRITE) = 12$.

Взяв число повторений BRANCH равным 99, а число повторений неявного перехода, соответствующего выходу из цикла, равным 1, получим, что идеальное время исполнения цикла, приведённого в примере, 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 пред-

ставлена та же последовательность операций цикла с разбиением на *стадии* и с распределением цикловых регистров по результату планирования.

Смысл стадий состоит в том, что на их границе передаются значения 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 с расчетом на 13 стадий. Функция планирования наложенного цикла определяется по аналогии с функцией планирования узла. Результатом функции планирования наложенного цикла является вектор упорядоченных пар $\langle t, s \rangle$, где t – неотрицательное положительное число (время запуска операции), s – натуральное число (номер стадии операции).

Теперь найдём время исполнения цикла. Чтобы получить значения аргументов операций, относящихся к каждой стадии, в общем случае необходимо выполнение операций всех предыдущих стадий. Поэтому операции последней стадии, соответствующие первой итерации цикла, могут исполняться спустя 12 тактов работы цикла. Эти такты работы наложенного цикла называются его *прологом*. Если спланирован цикл в n команд и m стадий, то 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 так далее. 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 приведенном примере это время равно $1 * ((13-1) * 1 + 100 * 1) = 112$ тактов. Это означает, что время исполнения наложенного цикла может быть меньше идеального времени исполнения (в нашем случае время исполнения наложенного цикла в 10 раз меньше идеального времени исполнения).

3. Нижняя оценка времени исполнения наложенного цикла

В предыдущем разделе 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 стадий и число широких команд. 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 существенно зависит от параметров архитектуры. Пусть n_a – минимальное число команд, необходимое для упаковки операций цикла (*размер цикла по ресурсам*).

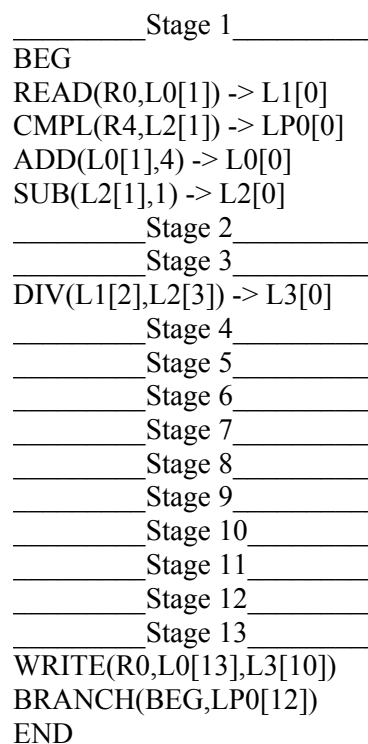


Рис. 2. Последовательность операций наложенного цикла с разбиением на стадии

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

В примере из предыдущей главы в теле цикла присутствовали операции, вырабатывающие значения, которое они сами использовали на следующей итерации (ADD и SUB).

Операция ADD не может выполняться в следующей итерации раньше, чем сформируется значение, которое она вырабатывает в текущей итерации, то есть размер наложенного цикла не может быть меньше задержки $l(\text{ADD}, \text{ADD}) = 1$. В общем случае размер наложенного цикла не превышает длины максимальной рекуррентности цикла.

Рекуррентность является сильно связным компонентом в графе зависимостей наложенного цикла [2] (граф зависимостей наложенного цикла отличается от графа зависимостей скалярного участка наличием зависимостей по обратной дуге, называемых «обратными»).

Определим понятие длины рекуррентности. Будем считать, что рекуррентность имеет класс 1, последних итерациях цикла, используя разумное количество вычислений.

В итоге, ограничение формулируется следующим образом. Существует класс операций, вызывающих побочный эффект, которые требуют планирования на последней стадии. Пусть op_1, \dots, op_d - операции, требующие планирования на последней стадии (по определению считаем закручивающий переход операцией с последней стадии, поэтому $d > 0$). Определим для наложенного цикла число n_s . Тогда $n_s = \max_i (T_e(\text{Br}) - T_i^l + 1)$, где $T_e(\text{Br})$ – время раннего закручивающего перехода, T_i^l - время позднего [2] операции $op_i, i = 1, \dots, d$.

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

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 $T_L \geq C_L \max\{T_e(L), A_L \max\{n_a(L), n_r(L), n_s(L)\}\}$;

если $A_L \geq T_e(L)$, то $T_L \geq C_L \left(\frac{T_e(L)}{\max\{n_a(L), n_r(L), n_s(L)\}} [-1 + A_L] \max\{n_a(L), n_r(L), n_s(L)\} \right)$

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

Наложённые 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 циклы. Их оптимизация практически ничем не отличается от оптимизации ациклических участков, и направлена на уменьшение длины критического пути, то есть на уменьшение $T_e(L)$ [2]. 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 $n_a(L)$. Их задачей 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 расщеплением в общем случае разбивает цикл на три цикла: исходный цикл без условия и альтернатив и два цикла, содержащих альтернативы. В двух частных случаях эта оптимизация наиболее эффективна, так как не требует построения дополнительных операций в телах полученных циклов. Это случаи инвариантного условия и некоторого класса условий, накладываемых 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, в каждом из которых отсутствуют вычисления, стоящие под условием выхода значения индуктивной переменной за свои границы.

Среди оптимизаций низкого уровня повышается значение эквивалентных преобразований, сокращающих число операций, - сбора общих подвыражений, удаления избыточных пересылок и чтений из памяти и других. 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, состоящую в перестановке индуктивных переменных охватывающего и внутреннего циклов. Следовательно, если охватывающий цикл повторялся в среднем большее число раз, чем внутренний, то после перестановки большее число повторений будет иметь внутренний (наложенный) цикл.

Ещё более эффективной в этом случае является оптимизация вследствие слияния охватывающего и внутреннего циклов. Ее суть заключается в замене двух индуктивных переменных на одну, которая пробегает область, являющуюся объединением областей изменения исходных переменных сливаемых циклов. 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, то есть вносить основной вклад во время ис-

полнения цикла. Простейший способ оптимизации рекуррентности состоит в том, чтобы применить на операциях рекуррентности весь арсенал классических оптимизаций, предназначенных для сокращения критических путей в графах зависимостей, взяв в качестве целевой функции длину рекуррентности. Далее рассмотрим оптимизации, приводящие к исчезновению рекуррентностей.

Большинство рекуррентностей являются «искусственными». Они возникают в тех случаях, когда классический индексный анализ не может выявить отношение зависимости между двумя контекстными операциями. Разрыв таких зависимостей может привести к исчезновению сильно связанных компонентов в графе зависимостей (рекуррентностей).

Наиболее эффективным способом разрыва статически неразрешённых зависимостей между парами контекстных операций является построение динамических проверок вне тела цикла. При положительном результате обеспечивается независимость этих операций. В случае проверки с отрицательным результатом управление передаётся на копию цикла, для которого соответствующие пары контекстных операций считаются зависимыми.

Если невозможно осуществить проверки вне тела цикла, то для любой пары (чтение, запись) можно разорвать зависимость динамическим сравнением адресов на равенство. Для выровненных адресов достаточно построить не более 5 операций, чтобы установить независимость операций. К ним относятся:

- 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, которые спланированы выше перехода на компенсирующий код и для которых возможен процесс восстановления в компенсирующем коде. Наиболее эффективно применение данной оптимизации при наличии в аппаратуре специальной поддержки. В архитектуре «Эльбрус» 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 исполнения операции записи в память не может быть меньше размера стадии [2]. Следовательно, для минимизации этой разницы необходимы оптимизации, способные разрывать зависимости между операцией чтения и ее преемниками в графе зависимостей, тем самым, увеличивая её время позднего исполнения 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 наличие зависимости пары «запись, чтение», применяются оптимизации, описанные в предыдущем параграфе.

Рассмотрим более подробно случай совпадения адресов. Здесь возможны следующие ситуации.

Операция записи доминирует операцией 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 зави-

симось между чтением и записью разрывается, но в нужную часть результата операции чтения дополнительно вставляется записываемое значение.

Если операция записи не доминирует чтение, то в контексте наложенного цикла это означает, что она стоит под предикатом. Данный 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 $opreg_a \rightarrow R0$, пишущая в регистр, который операция записи использует в качестве аргумента. Данная зависимость разрывается с помощью её переноса на операцию пересылки. Действительно, введя операцию пересылки значения регистра $R0$ в некоторый регистр $R10$ ($ADD(R0,0) \rightarrow R10$) и заменив аргумент операции записи $R0$ на $R10$, можно перенести антизависимость от операции $opreg_a$ с операции записи на операцию пересылки ADD .

4.4 Открутка пролога наложенного цикла

Выше были описаны основные оптимизации, используемые 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 «открутке» наложенного цикла, то есть 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 наложенный цикл (имеется в виду большое количество незаполненных, полупустых или пустых инструкций). Применить открутку можно ко всему прологу, и лишь к его начальной части, но единицей открутки должна быть целая стадия. При этом, если обрабатывается i -ая стадия, то вместе с ней следует обработать 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, намечено 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. (Операции, относящиеся к одной стадии, но разным итерациям, отличаются 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, открутка пролога наложенного цикла является необходимым техническим преобразованием.

5. Технология отката

Эффективность OVERLAP-технология была проиллюстрирована в главе 2. Однако, в этой связи необходимо принять во внимание основные ограничения для наложенных циклов, приведенные в главе 1. Одним из наиболее жестких является требование представления тела цикла в виде одного узла, требующее применить технологию планирования наложенного цикла. В связи с этим ограничением уместен вопрос - 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 в качестве не наложенного цикла меньше времени исполнения в качестве наложенного цикла.

Действительно, исследуем цикл на Рис. 3, среднее число повторений которого равно R . Рассмотрим оптимальное планирование этого цикла. Пусть

T'_{AB} - время запуска перехода из узла А в узел В,
 T'_{AC} - время планирования перехода из узла А в узел С,
 С,

T'_{BC} - время запуска перехода из В в С,
 T'_{CA} - время планирования перехода из С в А,
 T'_{exit} - время запуска выхода из цикла.

Определим следующие отношения:

$$\begin{aligned} T_{AB} &= T'_{AB} + 1, \\ T_{AC} &= T'_{AC} + 1, \\ T_{BC} &= T'_{BC} + 1, \\ T_{CA} &= T'_{CA} + 1, \\ T_{exit} &= T'_{exit} + 1. \end{aligned}$$

Без ограничения общности будем считать сумму повторений внешних переходов на голову цикла равной 1. Тогда оптимальное время исполнения цикла как не наложенного равно

$$T_L = R \left(\frac{T_{AC} + T_{AB} + T_{BC}}{2} \right) + (R-1) T_{CA} + T_{exit}.$$

Предположим, что В - узел, содержащий абсолютно параллельные вычисления, то есть каждая операция, отличная от перехода, а также 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, то есть минимального числа инструкций, необходимых для упаковки операций цикла, которое, в силу выбора узла В, не меньше T_{BC} . Таким образом, время исполнения наложенного цикла $T_L^{ovl} \geq RT_{BC}$.

Далее будем добавлять в узел В операции то тех пор, пока не будет выполнено условие: $T_{BC} > T_{AB} + T_{AC} + 2T_{CA} + 2T_{exit}$. Тогда $T_L < R \left(\frac{T_{AC} + T_{AB} + T_{BC} + 2T_{CA} + 2T_{exit}}{2} \right) < RT_{BC}$ 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 мето-

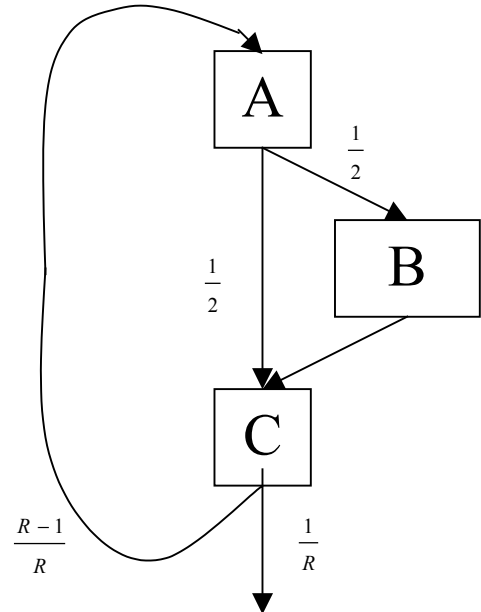


Рис. 3. Пример наложенного цикла

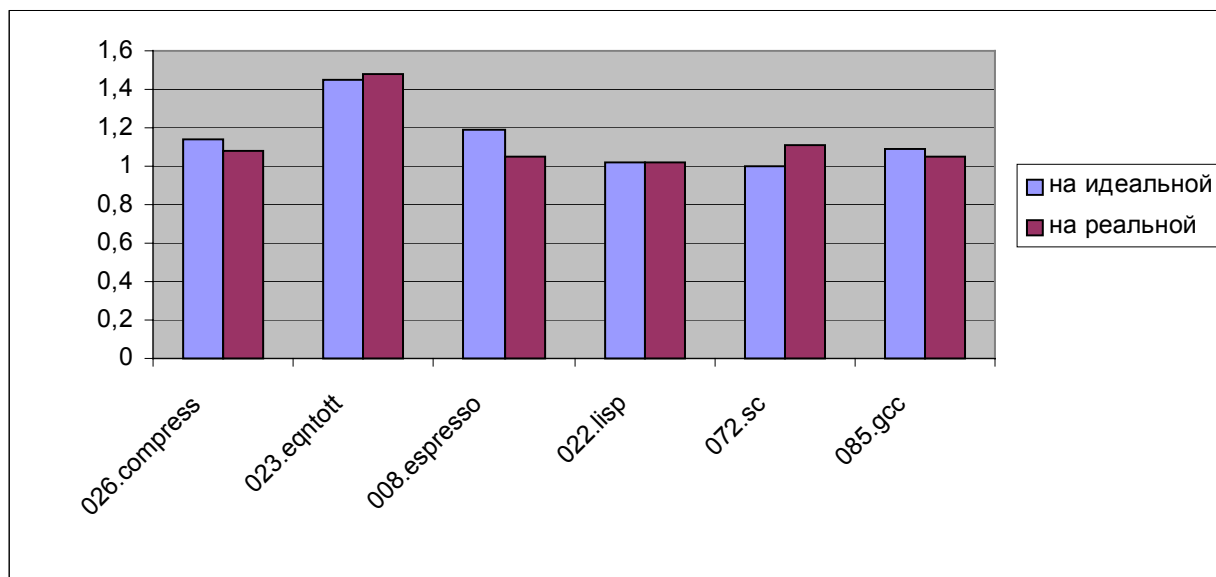
дами. Для решения этой проблемы в технологии OVERLAP предусмотрен откат. Для каждого наложенного цикла на ранней стадии компиляции делается его точная копия, которая не считается наложенным циклом. Наложённый цикл и его копия проходят через основные фазы компиляции, однако к первому применяется технология 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.

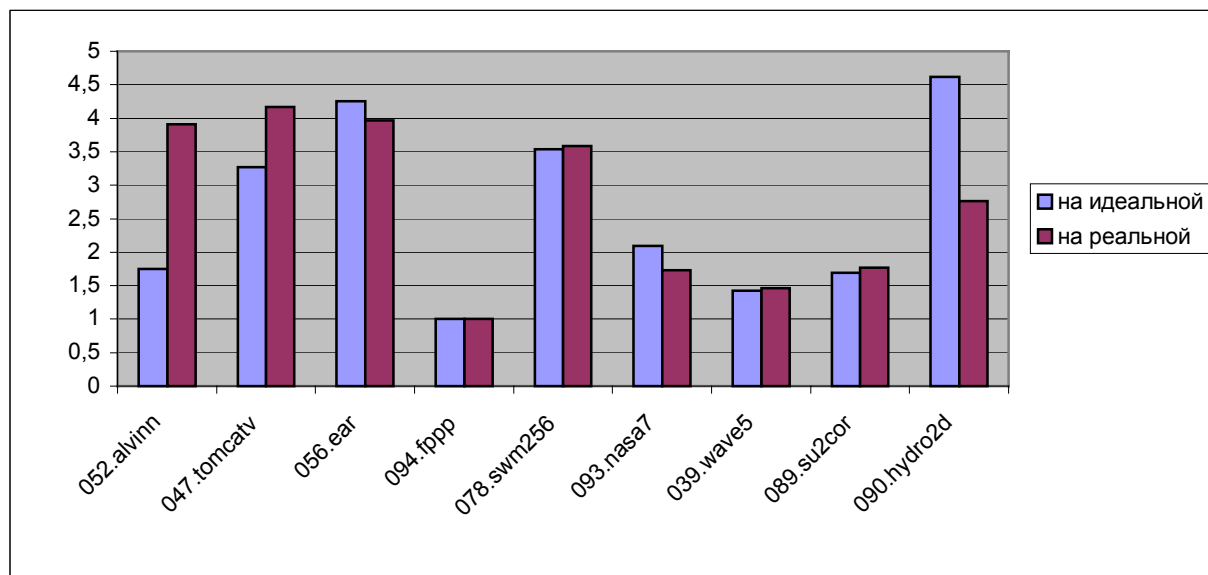
6. Экспериментальные результаты

В предыдущих параграфах была описана технология оптимизации циклов, основанная на аппаратной поддержке. На Рис. 4 приведена статистика эффективности технологии OVERLAP на задачах пакета *sres92* отдельно для целочисленных задач и задач, содержащих вычисления над числами с плавающей точкой. Результаты получены с использованием симулятора машины с архитектурой «Эльбрус». Время исполнения задач пакета измерялось в тактах.

В процессе эксперимента для каждой задачи пакета определялось отношение её времени исполнения, полученного для варианта компиляции с применением программных методов оптимизации циклов, к времени исполнения, соответствующего компиляции 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. Из графиков следует, 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 операции



Операции над числами с плавающей точкой

Рис. 4. Отношение времени исполнения задач пакета spec92, откомпилированных с применением программных методов оптимизации циклов и с применением технологии OVERLAP

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.

Литература

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 Symposium 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

Степаненков Антон Михайлович. Родился в 1977 году. Окончил МГУ им. М.В. Ломоносова в 2000 году, автор 3 статей. Область научных интересов - однородные схемы из автоматов, оптимизирующие компиляторы. Научный сотрудник ЗАО МЦСТ