Методы комбинирования алгоритмов анализа и оптимизаций в современных оптимизирующих компиляторах.

Дроздов А. Ю., Степаненков А. М.,

ИМВС РАН, г. Москва

{sasha|stepa_am}@mcst.ru

Аннотация

В работе предлагается оригинальный способ организации оптимизаций в виде управляемых пакетов. Он позволяет более эффективно использовать оптимизации по сравнению с классической организацией, представляющей собой последовательный запуск преобразований, и при этом сократить время компиляции программы.

Введение

В современных оптимизирующих компиляторах, имеющих в своём распоряжении большое количество оптимизирующих преобразований, на первый план выходят проблемы структурирования оптимизаций и организации их взаимодействия. Для эффективного использования своих потенциальных возможностей компилятору необходимо правильно подобрать последовательность запусков оптимизаций, так как оптимизации могут быть конфликтующими – применимость одной приводит к неприменимости другой, и наоборот, связанными – неприменимость одной влечёт неприменимость другой. Также очень важно для промышленного компилятора, имея внушительный список оптимизаций, уложиться в допустимое время при компиляции программы. Правильная организация оптимизаций в значительной степени способствует этому. Конкурентоспособный оптимизирующий компилятор должен иметь возможность динамически развиваться. В этой связи важно как быстро можно адаптировать старые оптимизации к нововведениям. При хорошей схеме структурирования оптимизаций, обеспечивающей контроль сразу над многими оптимизациями, можно быстро вводить новые требования и обеспечивать необходимые свойства. В работе рассматривается способ организации оптимизирующих преобразований в виде управляемых пакетов оптимизаций, позволяющий эффективно решать упомянутые проблемы. Данный способ является альтернативой классической последовательной организации оптимизаций [3]. Тема комбинирования различных методов анализа потока данных и оптимизаций затрагивалась и ранее [4]. В настоящей работе даётся общий подход к решению проблемы организации оптимизаций, не зависящий от их внутренней структуры и приводятся примеры конкретных реализаций управляемых пакетов оптимизаций в рамках промышленного компилятора для архитектуры Эльбрус-3М [5,6].

1. Структура оптимизации.

Введём понятие *оптимизации*, которое будем в дальнейшем использовать. Оптимизация – это эквивалентное преобразование над *промежуточным представлением* компилятора. Промежуточное представление есть набор внутренних структур данных

компилятора, представляющих программу и достаточных для получения её двоичного кода. Любая оптимизация состоит из трёх основных частей.

1. Техническая часть.

   Применение оптимизации. Осуществляется преобразование промежуточного представления.

2. Аналитическая часть.

   Проверка применимости оптимизации – доказательства эквивалентности преобразования, которое осуществляет оптимизация.

3. Интеллектуальная часть.

   Проверка эффективности оптимизации. Пусть зафиксирована Q – функция на промежуточном представлении, принимающая значение на множестве неотрицательных вещественных чисел (*функция качества*), Ir – промежуточное представление до оптимизации, Ir′ - промежуточное представление после оптимизации. Скажем, что оптимизация эффективна, если $Q(Ir') \geq Q(Ir)$.

   Структура оптимизации показана на рис. 1.

Как правило, для любой оптимизации можно выделить *объект* применения. Объект применения – это некоторое подмножество промежуточного представления, с которым работает оптимизация. Например, операция, пара операций, узел управляющего графа, цикл [1], процедура. Оптимизации на уровне операций принято называть низкоуровневыми оптимизациями, оптимизации, работающие на уровне узлов управляющего графа – макрооптимизациями.

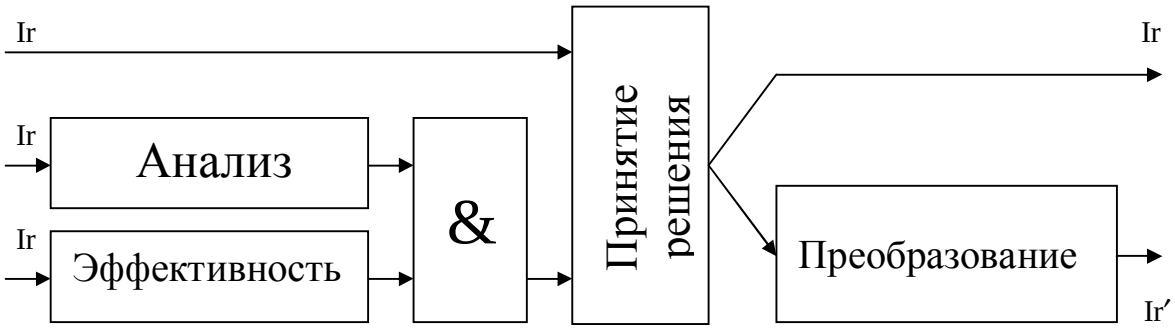

Рис. 1. Структура оптимизации.

2. Структура управляемого пакета оптимизаций.

Управляемый пакет оптимизаций – это пара <P,M>, где P – пакет оптимизаций, M – менеджер пакета оптимизаций. Пакет оптимизаций представляет собой набор оптимизаций с одним и тем же объектом применения. Менеджер оптимизаций обладает свойствами оптимизации. Он может выполнять дополнительный анализ на применимость и эффективность оптимизаций, входящих в его пакет. Менеджер оптимизаций может выполнять ряд преобразований на промежуточном представлении. В этом случае у оптимизаций пакета есть возможность не выполнять преобразования, а заказать, в общем случае, композицию преобразований у своего менеджера. Помимо этого, менеджер оптимизаций определяет порядок запуска оптимизаций для объекта. Объектом применения менеджера оптимизаций или управляемого пакета оптимизаций является набор объектов применения оптимизаций пакета. Например, если объект применения оптимизации – операция, то в качестве объекта применения менеджера пакета можно взять узел управляющего графа, если объект применения оптимизации – цикл, то

объектом применения управляемого пакета может служить дерево циклов [1]. Кратко структура управляемого пакета изображена на рис. 2.

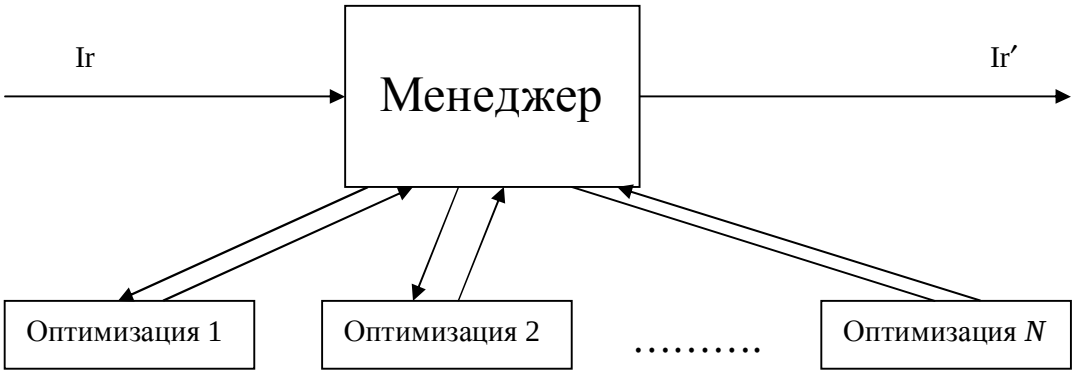

Рис. 2. Структура управляемого пакета оптимизаций

Далее приведём конкретные примеры управляемых пакетов оптимизаций, реализованных в компиляторе в рамках проекта Эльбрус-3М.

3. Пакет низкоуровневых оптимизаций на потоке данных.

Объектом применения оптимизаций пакета является операция. Дадим краткое описание оптимизаций, входящих в пакет.

1) Удаление мёртвого кода. Оптимизация удаляет операции, результаты которых далее нигде не потребляются, т.е. операции, присутствие которых в коде никак не влияет на поведение программы при исполнении.
2) Подстановка констант. Если удалось определить, что результат операции является константным, она удаляется, а в местах его использования подставляется константа.
3) Сбор общих подвыражений. Если имеются две эквивалентные операции, то одна из этих операций удаляется, во все места использования результата удаляемой операции подставляется результат эквивалентной операции. Если операции выполнялись условно, то оставшаяся операция ставится под предикат, равный логическому ИЛИ предикатов эквивалентных операций.
4) Эквивалентные преобразования над выражениями из операций представления. Например, x+0=x, x*1=x, x*8=x<<3, (x+1)+2=x+3.
5) Удаление избыточных чтений из памяти. Если для операции чтения из памяти имеется запись в память с не меньшим форматом по тому же адресу, доминирующая [1] операцию чтения, и между этими операциями нет операций, пишущих в ту же ячейку памяти, то операция чтения удаляется, а в местах её использования подставляется записываемое значение.
6) Удаление избыточных записей в память. Обрабатывается два случая. В первом записываемое значение является результатом чтения из памяти по тому же адресу, и между этим чтением и записью нет операций, пишущих в ту же ячейку памяти. Во втором для операции записи существует другая операция записи с не меньшим форматом по тому же адресу, постдоминирующая [1]

первую операцию, и между двумя операциями записи нет чтений из ячейки памяти, с которой работает первая запись. В обоих случаях операции записи в память являются избыточными, и поэтому удаляются.

Опишем менеджер пакета оптимизаций. Минимальным объектом применения менеджера пакета является узел управляющего графа, максимальным – вся процедура. При обработке процедуры узлы управляющего графа обходятся в порядке, который обеспечивает свойство, что при переходе к следующему узлу, все его предшественники в графе уже пройдены (при этом обратные дуги [1] не учитываются). Внутри узла операции обрабатываются в порядке их следования. Последовательность операций узла можно упорядочить по некоторому правилу и использовать это свойство при оптимизации. Один из способов упорядочения – по временам раннего или временам планирования [2] операций узла. Далее для каждой операции запускается пакет оптимизаций в том порядке, в котором они были представлены выше. Запуск пакета для операции осуществляется в цикле, условием выхода из которого является неприменимость ни одной из оптимизаций или удаление операции.

Оптимизации пакета не выполняют преобразования промежуточного представления, а заказывают набор преобразований у менеджера пакета. Менеджер пакета предоставляет следующие типы преобразования промежуточного представления.

1) Подстановка вместо результата операции константы.
2) Удаление операции по одному из правил.
3) Замена результата операции на результат уже существующей операции.
4) Замена результата операции на результат дерева новых операций, новые операции создаются по некоторому их формальному описанию.

Вынесение преобразований представления с уровня оптимизаций пакета на уровень менеджера пакета имеет много преимуществ. Во-первых, усиливается контроль над оптимизациями и увеличивается гибкость пакета к изменениям промежуточного представления. Зная тип преобразования, которое необходимо выполнить, менеджер может в зависимости от контекста, в котором он вызывается, отменять ту или иную оптимизацию, некорректную в этом контексте. Поскольку все преобразования выполняет менеджер, при изменении свойства промежуточного представления достаточно отразить это изменение только в нём. Во-вторых, многие потоковые оптимизации существенно различаются только анализом и сильно пересекаются по типам преобразований промежуточного представления. Выделение основных классов преобразований, применяющихся в низкоуровневых оптимизациях, и их поддержка менеджером пакета избавляет от излишнего дублирования текстов и существенного облегчает отладку компилятора.

Управляемый пакет потоковых оптимизаций имеет высокую эффективность при сравнительно небольших затратах по времени. Большинство оптимизаций используют одинаковые структуры данных, требующие инициализации. При объединении оптимизаций в пакет инициализация дополнительных структур данных делается один раз. Известно, что применение одних оптимизаций даёт возможность применять другие. Добиться эффективности, соизмеримой с эффективностью управляемого пакета, можно многократными вызовами отдельных оптимизаций, но этот способ требует больших временных ресурсов. В управляемом пакете многократные вызовы оптимизаций осуществляются только для тех операций, которые этого требуют. Для большинства операций делается всего один проход по всем оптимизациям.

4. Пакет цикловых макрооптимизаций.

Объектом применения оптимизаций пакета является цикл. Перечислим оптимизации, входящие в пакет.

1) Слияние внутреннего цикла с охватывающим циклом. Если i – индуктивная переменная [1] внешнего цикла, j – индуктивная переменная внутреннего цикла, тогда вводится такая индуктивная переменная k, что для любого индекса [1] цикла f(i,j) ставится в соответствие индекс $f'(k)$, $f(i,j) = f'(k)$ на каждой итерации цикла. Внутренний и внешний цикл преобразуются в новый цикл с индуктивной переменной k.
2) Перестановка циклов. Если i – индуктивная переменная внешнего цикла, j – индуктивная переменная внутреннего цикла, тогда вводится в качестве индуктивной переменной внешнего цикла j, внутреннего – i, и заменяются индексы f(i,j) на $f'(j,i)$ так, что $f(i,j) = f'(j,i)$ на каждой итерации внутреннего цикла.
3) Полная раскрутка цикла. Если число итераций цикла – константа C, цикл можно представить в виде повторения C раз его тела.
4) Частичная откруткa цикла. В случае, когда число повторений цикла не является константой, можно сделать несколько повторений тела цикла на основе профильной информации [1] с передачей управления на исходный цикл. Причём вероятность перехода на цикл после откруткu становится близкой к нулю.
5) Расщепление цикла по инвариантному условию. Для цикла создаётся точная копия. Управление передаётся на цикл или его копию в зависимости от предиката инвариантного условия. В цикле инвариантное условие считается тождественной истиной, в копии – тождественной ложью.
6) Расщепление цикла по индексу. Если в цикле имеется условие, ограничивающее изменение его индуктивной переменной, он разбивается на два цикла. В первом ограничение из условия берётся в качестве верхней границы, во втором цикле – в качестве нижней границы. В обоих циклах убираются вычисления, стоящие под условием выхода значения индуктивной переменной за свои границы.
7) Расщепление цикла по произвольному условию. В случае более общего условия для некоторого класса циклов возможно расщепление цикла на три цикла. Первый цикл не содержит альтернатив условия. Второй цикл содержит первую альтернативу условия. Третий цикл содержит вторую альтернативу условия. Число итераций первого цикла равно числу итераций исходного цикла. Среднее число итераций второго цикла равно среднему числу повторений первой альтернативы. Среднее число итераций третьего цикла равно среднему числу повторений второй альтернативы. Значения переменных исходного цикла, используемых в альтернативах, запоминаются в массивы в первом цикле и считываются во втором и третьем циклах.
8) Разрыв статически неразрешённых зависимостей по памяти. Во многих случаях для пары операций чтения или записи в память в цикле невозможно статически определить, могут ли они обращаться к одной и той же ячейке памяти во время исполнения цикла. В некоторых случаях можно построить цепочку условий вне цикла, выполнимость которых обеспечивает независимость по памяти некоторых пар. В этом случае строится точная копия цикла, на которую передаётся управление в случае истинности цепочки условий, и в которой указанные пары операций считаются независимыми.

Объектом применимости управляемого пакета является дерево циклов. Перед описанием менеджера пакета сформулируем основные задачи, которые он должен решать.

Первая задача – минимизация времени оптимизации цикловых участков. Как правило, анализ применимости и эффективности цикловых макрооптимизаций и преобразования промежуточного представления не являются критичными по времени. Значительную часть времени работы в таких оптимизациях занимает построение глобальных (на всю процедуру) аналитических структур данных, которые являются общими для всех оптимизаций. Причём в силу нетривиальности осуществляемых оптимизациями преобразований промежуточного представления, коррекция этих структур данных может быть нелокальной (в отличие от пакета низкоуровневых оптимизаций, где коррекция является несложной и передана менеджеру пакета). Поэтому часто самым хорошим решением при коррекции аналитических структур данных является их перестроение. Требуется так организовать работу с оптимизациями, чтобы сократить число перестроений глобальных структур данных.

Многие из описанных оптимизаций, входящих в пакет, приводят к значительному увеличению кода процедуры. Бесконтрольное увеличение кода процедуры приводит к резкому возрастанию времени обработки компилятором такой процедуры. От многих глобальных оптимизаций, имеющих нелинейную сложность, нужно будет отказаться. В результате полученный код процедуры будет некачественным. Вторая задача менеджера пакета цикловых макрооптимизаций – не допускать значительного роста кода и при этом эффективно использовать оптимизации, активно дублирующие циклы.

Третья задача – разрешение конфликтов между оптимизациями. Области применимости некоторых оптимизаций существенно пересекаются. Можно применить несколько оптимизаций к одному и тому же циклу и выполнить по сути одно и то же преобразование разными способами (например, оптимизации расщепления цикла). В некоторых случаях применимость одной оптимизации зависит от применимости другой. Это также необходимо учитывать при управлении пакетом.

Опишем алгоритм обработки дерева циклов.

Введём некоторые обозначения. Для каждого цикла введём состояние. Возможны четыре различных состояния: UNDEF, PROCESSED, WAIT, CLOSED. В начале все циклы находятся в состоянии UNDEF. На рис. 3 приведён алгоритм, по которому работает менеджер пакета цикловых макрооптимизаций.

```
LoopMacroOptimizationManager()
{
    MAX_INCREASE = CalcMaxIncrease();
    loops_list = FormInnermostLoopsList();
    SortLoopsListByProfile( loops_list);
    ConvertLoopsListState( loops_list, UNDEF, PROCESSED);
    loop = GetFirstLoopFromList(loops_list);
    while ( 1 )
    {
        if ( ApplyOptimizationsForLoop( loop, loops_list) )
        {
            if ( IsLoopNotDead( loop) )
            {
                SetLoopState( loop, WAIT);
            }
            loop = GetNextLoopFromList( loops_list, loop);
        } else
        {
```

```
                SetLoopState( loop, CLOSED);
                loop = UpdateLoopsListAfterCloseLoop( loops_list, loop);
            }
            if ( loop != NULL )
            {
                continue;
            }
            loop = GetFirstLoopFromList(loops_list);
            if ( loop == NULL )
            {
                break;
            }
            ReconstrAllStructures();
            SortLoopsListByProfile(loops_list);
            ConvertLoopsListState( loops_list, WAIT, PROCESSED);
            loop = GetFirstLoopFromList(loops_list);
        }
}

UpdateLoopsListAfterCloseLoop( loops_list, loop)
{
        pred = GetLoopPredInTree( loop);
        if ( IsAllLoopSuccsClosed( pred) )
        {
            SetLoopState( pred, PROCESSED);
            InsertLoopInLoopListAfterLoopByProfile( loops_list, pred, loop);
        }
        return RemoveLoopFromList( loops_list, loop);
}
```

Рис. 3. Алгоритм обработки дерева циклов менеджером пакета цикловых макрооптимизаций.

CalcMaxIncrease() вычисляет максимальный допустимый размер процедуры после всех преобразований.
FormInnermostLoopsList() формирует список самых вложенных циклов (не имеющих преемников в дереве циклов).
SortLoopsListByProfile(loops_list) сортирует список циклов в соответствии с их важностью (на основе профильной информации).
ConvertLoopsListState(loops_list, state1, state2) заменяет состояние цикла *state1* на *state2* в списке циклов.
GetFirstLoopFromList(loops_list) получение первого цикла из списка циклов.
ApplyOptimizationsForLoop(loop) пытается применять цикловые оптимизации в том порядке, в котором они были описаны выше. Если некоторая оптимизация применилась, происходит выход из процедуры с возможной коррекцией списка циклов (добавлением новых списков, возникших при преобразовании). При этом недопустимо увеличение размера процедуры. Также для каждого цикла и оптимизации есть ограничение на рост кода, которое зависит от важности цикла, оптимизации и резерва общего роста кода процедуры.
IsLoopNotDead(loop) – проверка, что цикл был удалён при оптимизации.
SetLoopState(loop, state) установка состояния цикла.
ReconstrAllStructures() перестроение всех аналитических структур.

GetLoopPredInTree(loop) – получение предшественника цикла в дереве циклов.
IsAllLoopSuccsClosed(loop) – проверка, что все преемники цикла в дереве циклов имеют состояние CLOSED.
InsertLoopInLoopListAfterLoopByProfile(loops_list, loop1, loop2) вставление в список циклов цикла *loop1* в соответствии с приоритетом, основанном на профильной информации, но не раньше цикла *loop2*.
RemoveLoopFromList(loops_list, loop) удаление цикла из списка, возвращается цикл, следующий за удаляемым.

Обработка дерева циклов происходит от листьев к корню. Список циклов представляет собой сечение дерева циклов, состоящее из независимых циклов. Вначале он состоит из самых вложенных циклов. В случае, когда ни одна из оптимизаций пакета не применилась к циклу, он удаляется из списка, и происходит попытка включить в список его предшественника в дереве. Для сохранения независимости сечения необходимо убедиться, что все преемники кандидата на включение были уже обработаны. Если к циклу была применена некоторая оптимизация, он переводится в состояние ожидания. Необходимо скорректировать глобальные структуры данных, которые стали некорректными для этого цикла, прежде чем можно будет снова запускать для него оптимизации. В силу независимости сечения циклы обрабатываются параллельно, то есть не происходит перестроения всех структур данных после применения оптимизирующего преобразования к одному циклу. При этом оптимизации могут быть применены к другим циклам сечения. Это существенно сокращает число перестроений глобальных структур данных, особенно для невысоких и широких деревьев циклов.

В представленной схеме невозможно бесконтрольное увеличение размера процедуры, так как менеджер пакета не допускает применение оптимизаций, проводящих к увеличению кода выше предельно допустимого. В то же время это ограничение сглаживается тем, что вначале обрабатываются самые важные циклы, затем менее значимые, согласно профилю. Таким образом, при обработке циклов, имеющих высокие приоритеты согласно профильной информации, резервов для дублирования кода должно быть ещё достаточно. Это верно для самых вложенных циклов. Для часто исполняемого, но не самого вложенного цикла, возможна более ранняя обработка менее значимых циклов, когда один из преемников цикла находится в состоянии ожидания. В этом случае, как отмечалось выше, менеджер пакета не допускает увеличения размера процедуры при обработке цикла, если это приводит к исчерпанию резервов, отведенных для других, ещё не обработанных, циклов с более высокими приоритетами.

Конфликтующие оптимизации, выполняющие схожие преобразования, упорядочены по возрастанию их общности. Сначала запускаются более специализированные, и, следовательно, более эффективные оптимизации, затем рассчитанные на самый общий случай. Основным критерием при разрешении конфликтов между оптимизациями является их эффективность. Как и в случае с пакетом низкоуровненных оптимизаций, запуск пакета макрооптимизаций также зациклен. Это решает проблему связанных оптимизаций (применимость одной оптимизации зависит от результата работы другой).

5. Макропакеты.

Выше была предложена простейшая реализация цикловых макрооптимизаций в виде управляемого пакета оптимизаций. Рассмотрим более сложную организацию оптимизаций, позволяющую увеличить эффективность их применения. Многие цикловые оптимизации при принятии решений используют ресурсные оценки. Это означает, что они наиболее эффективны на оптимизированном промежуточном представлении. На

представлении, содержащем мёртвый код и дублирующие вычисления, интеллектуальная часть оптимизации может отработать ошибочно, т.е. оптимизированное позже представление, на котором не применялась данная цикловая макрооптимизация, может оказаться более эффективным, чем оптимизированное позже представление, на котором она не применялась. Поскольку многие оптимизации пакета используют технику дублирования представления, после их применения в управляющем графе могут возникать пути, которые никогда не исполняются, т.е. возникать мёртвый код на уровне управления. Присутствие избыточных вычислений может приводить и к отказу от оптимизации менеджером пакета на основании ограничения роста кода процедуры.

Проблема решается применением управляемого пакета низкоуровневых оптимизаций и канонизации управления. Канонизация представления заключается в удаление избыточных ветвлений в управляющем графе на основании константности предикатов переходов с последующим вызовом сборщика мусора, удалении опустевших узлов, слиянии эквивалентных по управлению соседних узлов. Канонизацию управления следует запускать после работы низкоуровневых потоковых оптимизаций. Вызывать оптимизации можно каждый раз перед перестроением глобальных аналитических структур данных. Это не приводит к заметному увеличению времени работы всего алгоритма, так как низкоуровневые оптимизации и канонизация управления имеют меньшую сложность по сравнению с обновлением глобальных аналитических структур данных, однако значительно повышает его эффективность.

Пакет оптимизирующих преобразований, некоторые из которых представляют собой управляемые пакеты оптимизаций, называется макропакетом. Макропакеты с управлением называется управляемыми макропакетами.

Рассмотренный выше способ организации цикловых макрооптимизаций с применением оптимизаций низкого уровня и канонизации управления является примером управляемого макропакета.

6. Экспериментальные результаты.

Как уже отмечалось выше, одним из преимуществ организации оптимизаций в виде управляемых пакетов является сокращение времени обработки программы компилятором. Особенно важен этот показатель для макрооптимизаций, занимающих основную часть от времени работы компилятора. При этом самым трудоёмким процессом является построение глобальных аналитических структур данных, необходимых для их работы. С помощью организации макрооптимизаций в виде управляемых пакетов удаётся во многих случаях сократить число перестроений глобальных структур данных. В табл. 1 приведена статистика соотношения числа срабатываний макрооптимизаций и числа перестроений аналитических структур на задачах пакета spec92. Лучшие показатели достигаются на задачах, где достаточно большое число применений оптимизаций и где процедуры имеют невысокие, но широкие деревья циклов. Действительно, поскольку при применении оптимизаций пакета на процедуру необходимо, по крайней мере, одно перестроение структур данных, число таких применений должно быть больше единицы. Широкие деревья циклов обеспечивают большую параллельность при применении оптимизаций. Из табл. 1 следует, что «хорошую» структуру, позволяющую существенно сократить число перестроений глобальных структур данных, имеют задачи, содержащие вычисления над числами с плавающей точкой, 039.wave5 и 090.hydro2d.

Табл. 1.

Сравнение числа срабатываний макрооптимизаций и числа перестроений глобальных аналитических структур данных.

Тест	Число срабатываний (n)	Число перестроений (r)	n/r
026.compress	1	1	1.000
023.eqntott	20	13	1.538
008.espresso	189	109	1.734
022.lisp	28	28	1.000
072.sc	51	39	1.308
085.gcc	202	126	1.603
052.alvinn	8	6	1.333
047.tomcatv	1	1	1.000
056.ear	12	8	1.500
094.fppp	8	5	1.600
078.swm256	7	5	1.400
093.nasa7	32	21	1.524
039.wave5	141	42	3.357
089.su2cor	29	17	1.706
090.hydro2d	77	22	3.500

Заключение

В данной работе предложен способ организации оптимизаций в виде управляемых пакетов. Он обеспечивает более эффективное использование оптимизаций по сравнению с классической организацией, представляющей собой последовательный запуск преобразований. Организация оптимизаций в виде управляемых пакетов позволяет уменьшить затраты при реализации новых оптимизаций и развитии компилятора, сократить время компиляции программы.

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. D.F. Bacon, S. L. Graham, O.J. Sharp, “Compiler Transformations for High-Performance Computing”, ACM Computing Surveys, Vol. 26, No 4, December 1994.
4. Clifford N. Click, “Combining Analyses, Combining Optimizations”, A thesis submitted in partial fulfillment of the requirements for the degree Doctor of Philosophy. Houston, Texas, February 1995.
5. 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.
6. 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.