Методы контекстного межпроцедурного распространения свойств значений переменных программы

Дроздов А. Ю., Сыркин А. Г.

Институт микропроцессорных вычислительных систем РАН

sasha@mcst.ru, syrkin@mcst.ru

Введение

В большинстве современных оптимизирующих компиляторов возникает необходимость распространения некоторых свойств значений переменных [4, 5]. Наиболее простым примером такого свойства является равенство значения переменной некоторой константе при всех запусках программы. Чтение таких переменных можно заменить константой. Но есть и другие свойства, например диапазоны изменения значений или выравнивания адресов. В работе будет описан общий алгоритм анализа, распространяющего свойства значений и его применение для указанных примеров. Результаты такого анализа могут использоваться при замене вызова процедуры на вызов её упрощённой копии (Procedure Cloning, PC), подстановке тела процедуры вместо её вызова (Procedure Inlining, PI), поиске имени вызываемой по указателю процедуры, удалении мертвого кода, преобразованиях циклов и при проведении многих других оптимизаций. Более подробное описание этих преобразований можно найти в [1].

В процессе распространения происходит статический сбор информации обо всех значениях переменных, которые могут быть присвоены во всех точках программы при всех возможных её запусках. Предполагается подача на компиляцию всей программы целиком. Выполняется обход программы сверху вниз, начиная со стартовой процедуры. Описываемый анализ является чувствительным к управлению внутри процедуры и к различным вызывающим контекстам [2, 5, 6]. Собирается информация по всем возможным путям в управлении процедуры при всех возможных вызовах. Далее для краткости и простоты изложения описываемые свойства значений переменных будем называть просто *значениями*.

Можно работать с любыми переменными, но тогда необходимо иметь вычисленные значения указателей. В том случае, если анализ указателей не проведён, можно работать только с переменными, на которые не брался адрес. Каждой переменной программы соответствует некий блок памяти. Для обобщения и удобства работы с переменными введём ряд понятий [2].

Определение 1. *Множество локализаций* есть подмножество целых чисел вида $\{f + is \mid i \in Z\}$, которое описывается упорядоченная парой $\langle f, s\rangle \in Z \times N$, где f называется смещением, а $s-$ шагом. Для однозначности предполагается, что если $s \neq 0$, то $0 \leq f < s$.

Определение 2. *Локализацией* называется часть или весь блок памяти, соответствующий переменной, который представляется парой, состоящей из данного блока и множества локализаций, описывающего эту часть.

1. Распространяемые значения

Рассмотрим три наиболее важных типа значений, которые распространяются описанным ниже алгоритмом.

1) *Константы*, причем предполагаются все виды констант: целочисленные, с плавающей точкой и адресные (которые могут содержать адрес переменной).
2) *Диапазоны* изменения значений переменной, которые представляют собой упорядоченную пару целочисленных или с плавающей точкой констант.
3) *Выравнивания* адресов, которые реализованы в виде списка локализаций. Распространение выравниваний является частным случаем анализа указателей, поскольку выполняется только для переменных, на которые не брался адрес [2].

В каждом их этих типов необходимо построить некое специальное значение, назовем его *неизвестное значение*, которое нужно для обозначения случаев, когда алгоритм не в силах определить конкретное значение. А так же его используют в случае постоянного появления при каждой итерации новых значений для схождения анализа. При работе с рассматриваемыми типами значений требуется небольшой набор функций, которые полностью отражают всю специфику каждого из них, необходимую при анализе. Можно обойтись следующими тремя функциями:

Проверка значений на равенство. В случае констант и диапазонов проверяет точное совпадение. А в случае выравниваний совпадение списков с точностью до перестановки. С целью повышения точности, неизвестное значение считается не равным другим значениям.

Расширение значений есть функция с двумя аргументами, которая выдаёт значение, обобщающее данные. В случае констант расширение двух неравных констант возвращает неизвестное значение, а в случае двух равных констант – одну из них. Для диапазонов действует полная аналогия, поскольку выполняется слияние констант – границ диапазонов. А в случае выравниваний происходит объединение списков и удаление в результирующем списке повторений.

Обработка выражений возвращает значение выражения. Реализация этой функции сильно зависит от представления. Например, в случае констант применяется свертка арифметических операций. Более подробно об этом будет сказано ниже.

2. Частичные трансферные функции

Основной аналитической структурой, используемой при проведении распространения, является *частичная трансферная функция* (ЧТФ), которая содержит информацию о поведении процедуры в различных вызывающих контекстах. Формальное определение ЧТФ можно найти в [3]. Для каждого конкретного вызова процедуры происходит построение её ЧТФ, соответствующей вызывающему контексту, после чего созданная ЧТФ применяется только в подходящих вызывающих контекстах. По данной процедуре можно построить одну или несколько ЧТФ в зависимости от необходимости экономии памяти и ускорения анализа или увеличения его точности. При схеме с ограниченным количеством ЧТФ возникает необходимость сделать ЧТФ более грубой с целью соответствия её различным вызывающим контекстам. При схеме анализа с одной ЧТФ происходит комбинирование информации о различных вызывающих контекстах. Например, для клонирования процедур необходимо создавать несколько ЧТФ, по которым будут строиться соответствующие клоны. Основными компонентами ЧТФ являются:

Структура связывания, которая содержит информацию о значениях формальных параметров данной процедуры и значениях локализаций с глобальными блоками, которые используются в самой процедуре или в процедурах, вызываемых из нее. Указанные значения соответствуют данному вызывающему контексту. Значения формальных параметров хранятся в виде массива, а значения локализаций с глобальными блоками – в виде хеш-таблицы, индексированной локализациями.

Список присваиваний процедуры, отражающий её внутреннее строение, в котором хранится информация обо всех присваиваниях данной процедуры, упорядоченный с учетом порядка их доминирования [1]. Строится хеш-таблица, в которой локализации ставиться в соответствие список *присваиваний в локализацию*, отсортированный в порядке доминирования. Присваивание в локализацию есть пара из узла управляющего графа и значения, которое было присвоено в данном узле. Построенный список не должен быть слишком большим, поскольку количество присваиваний в конкретную переменную в рамках одной процедуры невелико.

Структура результата процедуры содержит информацию том, как процедура влияет на вызывающий контекст. В ней хранятся значения, возвращаемые процедурой, и значения локализаций с глобальными блоками, в которые были присваивания в самой процедуре или в процедурах, вызываемых из нее. Как и в случае структуры связывания значения локализаций с глобальными блоками хранятся в виде хеш-таблицы индексированной локализациями. В принципе, результат процедуры можно было бы не хранить, поскольку есть возможность почерпнуть эту информацию из списка присваиваний, но это замедлило бы работу алгоритма, поскольку каждый раз при применении ЧТФ пришлось бы обходить весь список присваиваний.

Список узлов схождения процедуры содержит информацию об узлах схождения и соответствующих им локализациях по которым надо вычислять φ-функции. Указанное вычисление φ-функции в узле состоит в создании присваивания в локализацию значения равного расширению пришедших по всем дугам сходящимся в этом узле значений. Данная информация организована в виде хеш-таблицы индексированной узлами, в которой хранятся списки локализаций.

Кроме вышеперечисленных компонентов, в ЧТФ хранятся списки узлов с вызовами setjmp и longjmp. Также имеется ряд флагов, в частности, флаг изменения процедуры, который поднимается в случае изменения значений, находящихся в ЧТФ. Схема ЧТФ представлена на рис. 1.

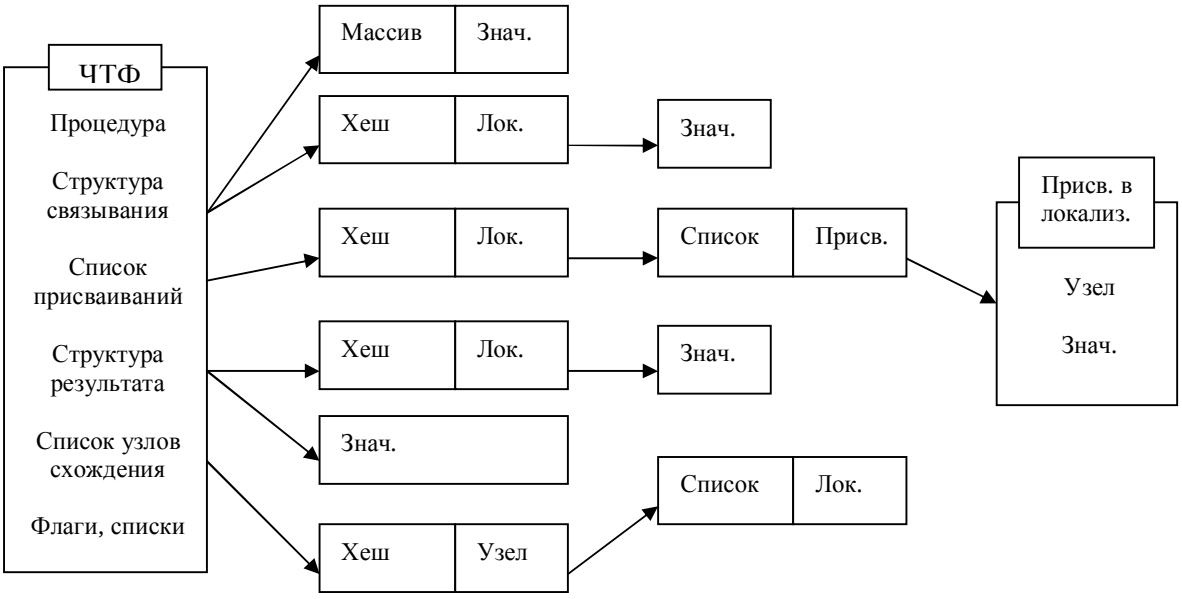

Рис. 1. Схема ЧТФ.

Для работы с ЧТФ используются две основные функции SetValue и GetValue. В дальнейших описаниях алгоритма будут использованы следующие обозначения:
GetLocAssignListFromPTF(*dst, ptf*) – получение по ЧТФ *ptf* списка присваиваний в локализацию *dst*.
AddAssignToPTF(*dst, val, n, ptf*) – добавление к списку присваиваний ЧТФ присваивания значения *val* в локализацию *dst* в узле *n*.
InsertNewAssingInListBefore(*val, n, alist, a*) – добавление присваивания значения *val* в узел *n* перед присваиванием *a* в списке alist.
AddAssingToPTF(*dst, val, n, ptf*) - добавление к списку присваиваний ЧТФ присваивания в локализацию dst значения *val* в узел *n*.
SetPTFChangeValue(*prf*) – установка флага изменения ЧТФ ptf.
SetAssignValue(*asgn, new_val*) – установка значения new_val присваивания присваивания asign в локализацию.
AddLocToFhiFuncsPTF(*dst, m, ptf*) - добавление узла m и локализации dst в список узлов схождения ЧТФ ptf.
GetFormalValue (dst, ptf) – получение значения формального параметра dst.
GetGlobalValue (dst, ptf) – получение и сохранение в структуре связывания ЧТФ ptf значения глобальной локализации dst.

Для создания или коррекции присваивания значения в локализацию в данном узле используется функция SetValue (алгоритм 1). В качестве параметров она получает: локализацию, в которую происходит присваивание, присваиваемое значение; узел управляющего графа, где происходит присваивание; обрабатываемую ЧТФ, в список присваиваний которой вносится изменение. При работе этой функции выполняется проход по списку присваиваний в данную локализацию, если таковой существует, и поиск присваивания в узле, который доминирует над данным узлом (строки 8-17). После чего, если присваивания с таким узлом не было, создается новый элемент в списке присваиваний (строка 20). Если списка не было вообще, то такой список создается с добавлением данного присваивания. В случае, когда присваивания с таким узлом не было, поднимается флаг изменения ЧТФ. Если присваивание с таким узлом уже было, то происходит расширение значения в этом узле, и в случае его изменения, поднятие флага изменения ЧТФ (строки 22-30). С использованием итерационного фронта доминирования (Iterated Dominance Frontier, IDF) [1], для всех узлов схождения, которые достижимы из исходного узла, данная локализация добавляется в список узлов схождения (строки 31-33).

Алгоритм 1: Установка значения.

```
void SetValue( Loc dst, Value val, Node n, PTF ptf)
begin
1:   Assign asgn;
2:   AssignList alist = GetLocAssignListFromPTF( dst, ptf);
3:   if ( there is no alist )
4:      asgn = AddAssignToPTF( dst, val, n, ptf);
5:   else
6:     asgn = NULL;
8:     for ( each Assign a in alist )
```

```
9:          if ( node of a dominates n )
10:             if ( node of a == n )
11:                asgn = a;
12:             else
13:                asgn = InsertNewAssingInListBefore( val, n, alist, a);16
14:             endif
15:             break;
16:          endif
17:       endfor
18:    endif
19:    if ( asgn == NULL)
20:       asgn =  AddAssingToPTF( dst, val, n, ptf);
21:    endif
22:    if ( is new assign )
23:       SetPTFChangeValue( prf);
24:    else
25:       Value new_val = WidenValues( val, value of asng);
26:       if ( is value change )
27:          SetPTFChangeValue( ptf);
28:          SetAssignValue( asgn, new_val);
29:       endif
30:    endif
31:    for ( each Node m from IDF of n )
32:       AddLocToFhiFuncsPTF( dst, m, ptf);
33:    endfor
endproc
```

Для получения значения данной локализации в узле используется функция GetValue (алгоритм 2), условно говоря, обратная процедуре SetValue. В качестве параметров она получает: локализацию, значение которой ищется; узел управляющего графа; обрабатываемая ЧТФ, в которой осуществляется поиск. При работе этой функции выполняется проход по списку присваиваний в данную локализацию, если таковой существует, и производится поиск присваивания в узле, который доминирует над данным узлом (строки 3-7). Если такое присваивание найдено, то функция возвращает присвоенное значение (строка 5). В противном случае, при поиске значения локализации с блоком формального параметра или глобальной переменной, возвращается значение, взятое из структуры связывания (строки 8-14). В том случае, если значение локализации с глобальным блоком еще не подсчитано в структуре связывания, то происходит его вычисление в вызывающем контексте с помощью рекурсивного вызова GetValue с последующим занесением его значения в структуру связывания. Таким образом, происходит поиск значения в глубь стека процедур.

Алгоритм 2: Получение значения.

```
Value GetValue( Loc dst, Node n, PTF ptf)
begin
1:    Assign asgn;
2:    AssignList alist = GetLocAssignListFromPTF( dst, ptf);
3:    for ( each Assign a in alist )
4:       if ( node of a dominates n )
5:       return GetAssignValue( a);
6:       endif
7:    endfor
8:    if ( dst block is formal param )
9:       return GetFormalValue ( dst, ptf);
10:   else if ( dst block is global )
11:      return GetGlobalValue ( dst, ptf);
```

```
12:  else
13:      return unknown;
14:  end
endproc
```

3. Обработка одной процедуры

Для анализа процедуры используется специальное разбиение управляющего графа на более мелкие узлы. При таком разбиении узлы схождения управления отделяются от узлов с вызовами процедур и записями. В одном узле может находиться, либо запись, либо вызов, исключением является случай записи возвращаемого значения процедуры.

В процессе анализа часто возникает необходимость подсчета значения выражения содержащего константы и переменные в данном узле управляющего графа. Для этого происходит обход дерева выражений в порядке Reverse Post Order (RPO) [1] с целью вычисления значений соответствующих подвыражений для использования их в объемлющих выражениях. Выполнение обхода в порядке RPO гарантирует, что при обработке дерева подвыражений каждая операция будет обработана после того как будут обработаны операции, которые вырабатывают значения ее аргументов. В том случае, когда требуется вычислить значение переменной, используется функция GetValue, которой подается соответствующая локализация, узел предшествующий данному узлу, и обрабатываемая ЧТФ. Очевидно, что такой узел должен быть единственным, поскольку при разбиении узлы схождения отделялись от других узлов. Если обрабатывается операция взятия адреса переменной, то строится соответствующая адресная константа. Описанный метод напрямую использует свойства и конкретный смысл вычисляемых значений.

По данной процедуре в конкретном вызывающем контексте происходит итерационная обработка всех узлов управляющего графа с подсчетом значений в каждом узле и занесением полученной информации в ЧТФ. Чтобы обеспечить обработку предшественников узла раньше, чем преемников, обход узлов происходит в согласно RPO нумерации [1]. Обходы процедуры продолжаются до тех пор, пока не перестанут происходить изменения в ЧТФ. Такие изменения могут произойти только в одном из трёх случаев:

1. Создается новое присваивание.
2. Происходит расширение значения у какого-либо существующего присваивания.
3. Меняется значение в структуре связывания.

Результатом этой обработки является полностью сформированная ЧТФ, которая соответствует данному вызывающему контексту.

В процессе обработки узлы управляющего графа делятся на три группы: присваивания, вызовы и узлы схождения. Обработка вызова процедуры будет рассмотрена в следующем пункте.

Присваивания состоят из двух подгрупп – скалярные присваивания, в которых записывается только одно значение, и агрегатные присваивания, в которых происходит запись сразу многих значений. Например, в программах, написанных на языке C, агрегатными присваиваниями являются присваивания структуры целиком, остальные присваивания являются скалярными. Обрабатываются только записи в конкретные переменные, которые участвуют в анализе.

При обработке скалярного присваивания по переменной из левой части строится соответствующая локализация, а по правой части происходит вычисление записываемого значения. После этого происходит занесение в список присваиваний ЧТФ нового или коррекция уже существующего присваивания, с использованием функции SetValue.

Аналогично, при обработке агрегатного присваивания происходит занесение в ЧТФ присваиваний во всевозможные локализации, соответствующие полям структуры. Если происходит инициализация массивом констант, то эти значения для присваиваний берутся из самого массива, а если происходит присваивание другой структуры, то берутся значения, хранящиеся в ней, поиск которых осуществляется в списке присваиваний ЧТФ с помощью GetConst.

В процессе обработки присваиваний каждому узлу схождения ставится в соответствие список локализаций, значения которых могут пройти через этот узел. Данная информация хранится в списке присваиваний. При обработке узла схождения делается проход по этому списку с вычислением значений хранящихся в нем локализаций во всех предшествующих узлах и последующим вычислением их расширенного значения. Далее происходит занесение в ЧТФ присваиваний в эти локализации расширенных значений в обрабатываемом узле. То есть просто происходит комбинирование значений, приходящих по всем входящим дугам управления.

После завершения обработки процедуры создается ее структура результата для последующего применения ЧТФ. Для этого происходит обход всех выходов из процедуры. И для каждой локализации с глобальным блоком, в которую было присваивание в этой ЧТФ, происходит подсчет ее значений во всех выходах из процедуры с целью последующего поиска расширенного значения для сохранения в структуре результата. Если процедура имеет результат, то в качестве возвращаемого значения для занесения его в структуру результата берется расширение возвращаемых значений по всем выходам.

4. Обработка вызова процедуры

При обработке узла вызова (алгоритм 3) анализ должен учесть влияние, которое оказывает вызов процедуры на вызывающий контекст. Обработка разбивается на три основных этапа: построение списка процедур, которые могут быть вызваны; поиск или построение соответствующих этим претендентам ЧТФ, которые подходят для данного вызывающего контекста; и добавление присваиваний в обрабатываемом узле вызова, соответствующих влиянию всех претендентов на вызывающий контекст.

В процессе анализа создаётся *стек вызовов*, в котором хранятся пары ЧТФ и узлов вызова. Перед анализом вызываемой процедуры создается ЧТФ, которая кладется в стек вызовов, а после завершения анализа вынимается из него. Анализ начинается со стартовой процедуры программы, для которой создается ЧТФ и кладется на дно стека. Например, в программах, написанных на языке С, это процедура main.

В случае вызова процедуры по имени список вызываемых процедур состоит только из этой процедуры. А в случае вызова по косвенности список состоит из всех процедур, на которые брался адрес. Указанный список составляется заранее, перед началом анализа. В принципе, этот список можно сокращать, если использовать результаты работы межпроцедурного анализа указателей [2]. Если в этом списке найдется не библиотечная процедура, которая не имеет реализации, то анализ прекращает свою работу, и его результаты удаляются без использования (строка 9). Такая ситуация возможна только в том случае, если на компиляцию была подана

программа не целиком. Большинство библиотечных процедур не может изменить значения переменных, на которые не брался адрес. Исключением является небольшое число библиотечных процедур, которые в качестве параметра могут получить указатель на процедуру и вызвать ее внутри себя. Примером такой процедуры является signal – библиотечная процедура языка С. Поэтому при вызове таких процедур списком вызываемых процедур является список всех процедур, на которые брался адрес. А для всех остальных библиотечных процедур обработка не происходит (строка 6).

В процессе обработки вызова происходит обход по построенному списку вызываемых процедур с целью поиска подходящих для данного вызывающего контекста ЧТФ и их применения. Если ЧТФ процедуры из списка нет в стеке вызовов (строка 12), то очевидно вызов не рекурсивный. В этом случае происходит поиск подходящей ЧТФ этой процедуры. ЧТФ считается подходящей для вызывающего контекста, если значения ее формальных параметров в структуре связывания равны значениям ее фактических параметров в данном вызывающем контексте и если значения локализаций с глобальными блоками в ее структуре связывания равны значениям этих локализаций в данном контексте. Если такая ЧТФ не существует, то она строится. Если же есть ограничение на количество ЧТФ, то происходит загрубление уже существующей ЧТФ. В случае, когда ЧТФ процедуры есть в списке, то очевидно вызов рекурсивный, и тогда для применения просто берется эта найденная ЧТФ и загрубляется в соответствии с текущим вызывающим контекстом. Загрубление ЧТФ происходит с использованием расширения значений в структуре связывания и проведением повторной обработки процедуры. Для вычисления значений фактических параметров используется функция вычисления значения выражения, описанная в предыдущем пункте. Если в случае рекурсивного вызова, найденная в стеке ЧТФ обработана только частично, то структура результата еще не сформирована и применение не происходит. В этом случае осуществляется поднятие флага необходимости повторного обхода у всех ЧТФ находящихся в стеке, и при второй итерации обработки стартовой процедуры эти флаги вступают в силу. При наличии в вызываемой процедуре вызовов longjmp создаем присваивания во все локализации, в которые были присваивания в текущей процедуре во всех узлах setjmp тех процедур, которые находятся в данный момент стеке. Необходимые списки хранятся в ЧТФ и пополняются при её применении и обработке.

Далее приведено схематическое описание функции обработки вызова EvalCall. В дальнейших описаниях алгоритма будут использованы следующие обозначения:

FindCallTargets(*n*) – построение списка процедур, которые могут быть вызваны из узла n.

EvalUnknownCall(*n, ptf*) – создание присваивания неизвестного значения в результат прерации вызова узла n в ЧТФ ptf.

RecordActuals(*n, ptf, proc*) – выдача структуры связывания процедуры proc, которая соответствует вызывающему контексту узла n в ЧТФ ptf.

GetPTFCallStack(*proc*) – поиск ЧТФ процедуры proc в стеке вызовов.

GetPTF(*bind, proc, n, ptf*) – поиск или построение для процедуры proc ЧТФ, подходящей для структуры связывания bind.

EvalProc(*tgt_ptf*) – обработка процедуры, описанная в предыдущем пункте.

ApplySummry(*tgt_ptf, n, ptf*) – применение результата.

UpdatePTFBind(*tgt_ptf, bind, n, ptf*) – приведение структуры связывания ЧТФ tgt_ptf в соответствие в вызывающим контекстом, хранящимся в bind.

Алгоритм 3: Обработка вызова.

```
void EvalCall( Node n, PTF ptf)
begin
1:   List call_list = FindCallTargets( n);
2:   for ( each Proc proc from call_list )
4:      if( is proc library )
5:         EvalUnknownCall( n, ptf);
6:         continue;
7:      endif
8:      if( is proc not realized )
9:        break;
10:     Bind bind = RecordActuals( n, ptf, proc);
11:     tgt_ptf = GetPTFCallStack( proc);
12:     if ( tgt_ptf == NULL )
13:        tgt_ptf = GetPTF( bind, proc, n, ptf);
14:        if ( is new tgt_ptf || ( is tgt_ptf mast eval && second iter main proc ) )
15:           SetPTFNoMustEval( tgt_ptf);
16:           PushCallStack( tgt_ptf, n);
17:           EvalProc( tgt_ptf);
18:           PopCallStack( );
19:        endif
20:        ApplySummry( tgt_ptf, n, ptf);
21:     esle
22:        if ( UpdatePTFBind( tgt_ptf, bind, n, ptf) )
23:           SetPTFChangeValue( tgt_ptf);
24:        endif
25:        if ( is tgt_ptf eval )
26:           ApplySummry( tgt_ptf, n, ptf);
27:        else
28:           SetPTFStackNoMustEval( tgt_ptf);
29:        endif
30:  endfor
endproc
```

5. Экспериментальные результаты

Целью проводимых исследований был подсчет количества точек программы, куда удалось распространить константы в ходе межпроцедурного распространения констант. Также проведены замеры времени исполнения в тактах кода программы. Замеры проводились в рамках работы над оптимизирующим компилятором проекта “Эльбрус” [7, 8] на реальных задачах пакета SPEC92, в который входят следующие тесты, написанные на языках C и Fortran.

Пакет SPECint92

- 008.espresso – задача минимизации булевских функций
- 022.li – интерпретатор языка lisp
- 023.eqntott – получение таблиц истинности
- 072.sc – электронные таблицы

Пакет SPECfp92

- 039.wave – решение уравнений Максела
- 056.ear – модель ушной полости
- 093.nasa7 – 7 математических программ
- 072.fpppp – квантовая химия

Табл. 1

Сводная таблица результатов исследований

Тест	Количество срабатываний	Такты без анализа	Такты с анализом	Улучшение по количеству тактов в %
008.espresso	440	45596595	45591980	0.01
022.li	105	4288609	4237288	1.21
023.eqntott	155	304116899	177726748	71.11
072.sc	726	98549021	72386014	36.14
039.wave5	1015	1151623186	1124810846	2.38
056.ear	239	67080783	60161608	1.15
093.nasa7	318	2618759910	2614996786	0.14
094.fpppp	130	2118341786	2058406888	2.91

Оценка эффективности рассмотренного анализа описывается отношением времени исполнения в тактах кода программы, полученной с его применением к количеству тактов без него. На обоих вариантах кода применялся PI и разнообразные цикловые оптимизации. Результаты анализа использовались для непосредственного распространения констант и для проведения PC. На тестах 023.eqntott и 072.sc произошло копирование процедур, которое сделало возможным применение ряда других оптимизаций. Это позволило добиться значительного, до 70%, улучшения показателей. На остальных тестах произошло улучшение в среднем около 1,5 %, что характерно для локальных оптимизирующих преобразований.

Заключение

В работе предложен обобщенный алгоритм межпроцедурного распространения различных видов информации о значениях переменных программы. Описаны применения этого алгоритма для случаев межпроцедурного распространения констант, диапазонов и выравниваний. Использование результатов распространения указанных значений позволяет улучшать производительность на реальных задачах на 70%.

Список литературы

1. **Steven S. Muchnick.** Advanced Compiler Design and Implementation – Morgan Kauffman, San Francisco, 1997, chapter 7.2.
2. **R. Wilson**. “Efficient, context–sensitive pointer analysis for C programs”. Phd thesis, Stanford University, 1997.
3. **Brian R. Murphy and Monica S. Lam**. “Program Analysis with Partial Transfer Functions”. Computer Systems Laboratory, Stanford University, 2000.
4. **Mooly Sagiv, Thomas Preps, Susan Horwotz** “Precise Dataflow Analysis with Applications Constant Propagation”. TAPSOFT 1995. pages 651-665.
5. **Laure J. Hendren, Marian Emami, Rakesh Chiya, Clark Verbrugge**. “A Practical Context-Sensitizes Interprocedural Analysis Framework For C Compilers”. ACAPS Technical Memo 72, 24 July 1993.
6. **M. Hind and A. Pioli**. “Assessing the effects of flow-sensitivity on pointer alias analyses”. Lecture Notes in Computer Science, 1503, pages 57-81. Springer-Verlag, 1998.

7. **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.
8. **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.