

# Эффективная реализация графа потока зависимостей

Дроздов А. Ю., Тютюник О. М., Шилов В.В.

Институт микропроцессорных вычислительных систем РАН

[sasha@mcst.ru](mailto:sasha@mcst.ru), [charmony@mcst.ru](mailto:charmony@mcst.ru)

## Введение

Потоковый анализ является важной частью процесса оптимизирующей трансляции программ. Теоретически любая проблема потокового анализа может быть решена с использованием лишь управляющего графа (УГ) программы. Однако для эффективного проведения потокового анализа в литературе были предложены несколько других аналитических структур данных, наиболее известными из которых являются *цепочки чтение-запись* [1], *форма единственного присваивания* [1] и *граф потока зависимостей* [2] (Dependence Flow Graph, далее DFG). Строгие определения и сравнительный анализ представлений можно найти, например, в [2].

Общая идея, лежащая в основе создания названных выше представлений – использование неоднородного распределения значимой для потокового анализа информации по управляющему графу программы. Понятно, что в типичных программах операции записи и чтения в определенный объект расположены в сравнительно небольшом количестве узлов управляющего графа по сравнению с общим количеством узлов в графе, и, соответственно, нет смысла рассматривать при анализе те узлы управляющего графа, которые не содержат значимой для потокового анализа информации. В то же время желательно, чтобы представление содержало в себе достаточное количество информации о потоке управления программы, так как ее отсутствие ведет к потере точности анализа и снижению эффективности оптимизаций. Из всех представлений наиболее удачно сочетает свойство разреженности и наличие информации о потоке управления именно DFG.

В работе исследуется вопрос применимости DFG в промышленных компиляторах – анализируются возникающие проблемы и их природа. Предлагается реализация DFG, позволяющая использовать эту структуру данных при анализе реальных программ, учитывая жесткие требования по скорости работы и занимаемой памяти, предъявляемые к промышленным компиляторам. Показывается возможность эффективного проведения оптимизаций на предложенной версии DFG.

## 1. Некоторые сведения о графе потока зависимостей

Основные сведения о DFG можно найти в [2], мы же приведем только некоторые определения и примеры. Для удобства изложения всюду в разделе 1 будем предполагать, что в управляющем графе программы в каждом узле может содержаться только одна операция, а узлы, имеющие несколько входящих или исходящих дуг, не содержат операций. Это позволит нам разбить все узлы управляющего графа на 6 типов: Начало, Конец, Чтение, Запись, Схождение, Расхождение. В узлах типа Чтение содержатся операции, читающие значение некоторого объекта. В узлах типа Запись содержатся операции, записывающие значение в некоторый объект. Узлы типов Схождение и Расхождение не содержат операций, они отображают соответственно схождение и расхождение потоков управления. Таким образом, несколько предшественников могут иметь только узлы типа Схождение, несколько приемников могут иметь только узлы типа Расхождение.

Для построения DFG управляющий граф программы разбивается на так называемые участки с одним входом и одним выходом (Single Entry Single Exit Regions, далее SESE-регионы).

**Определение 1.** Пара дуг ( $e_1, e_2$ ) управляющего графа образует SESE-регион, если дуга  $e_1$  доминирует над  $e_2$ ,  $e_2$  постдоминирует над  $e_1$ , и любой цикл в управляющем графе, содержащий  $e_1$ , содержит также и  $e_2$ , и наоборот.

В работе [3] показано, что два любых SESE-региона управляющего графа могут быть или вложенными друг в друга, или не пересекаться, т.е. если некоторая дуга  $e_3$  принадлежит одновременно различным SESE-регионам  $R_1$  и  $R_2$ , то либо  $R_2$  вложен в  $R_1$  (любая дуга из  $R_2$  принадлежит также  $R_1$ ), либо  $R_1$  вложен в  $R_2$  (любая дуга из  $R_1$  принадлежит также  $R_2$ ).

Узлами графа потока зависимостей являются операции, фиктивные узлы Начало и Конец, а также узлы специального вида Схождение и Расхождение (по аналогии с узлами типа Схождение и Расхождение управляющего графа). В DFG могут быть отображены зависимости между операциями различных типов. Следующее определение соответствует дуге DFG, отображающей зависимость типа “Запись-Чтение”. Определения для дуг, соответствующих другим типам зависимостей, строятся аналогично.

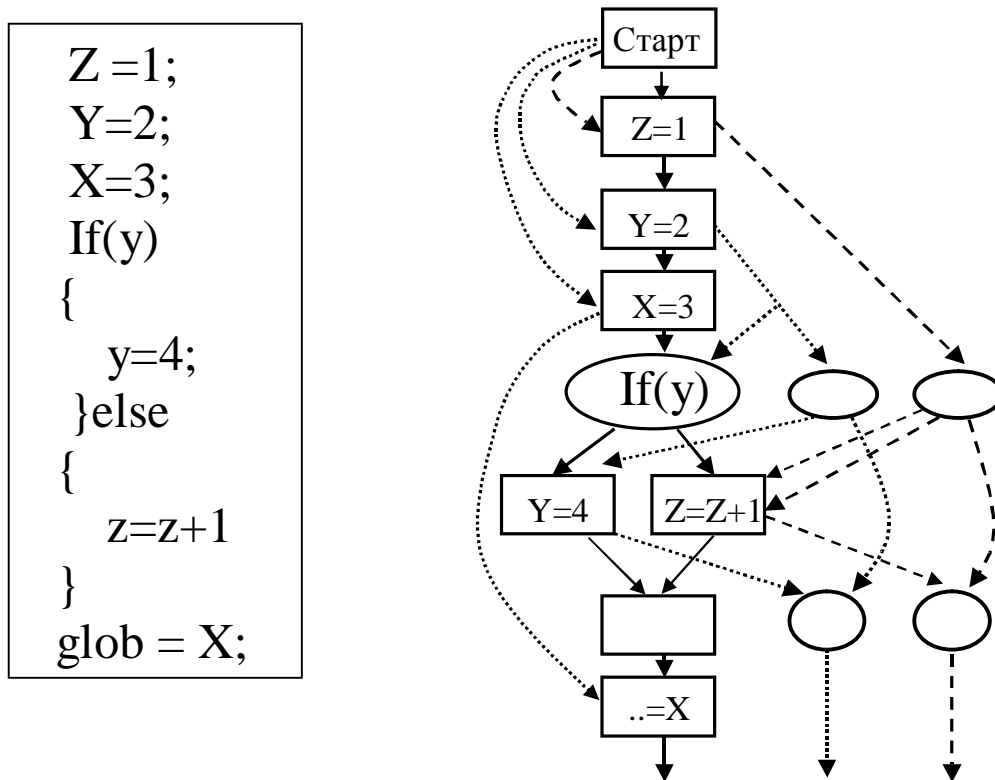
**Определение 2.** Дуга DFG, отображающая зависимость типа “Запись-Чтение” ставится в соответствие паре дуг ( $e_1, e_2$ ) управляющего графа программы, если:

- 1) Существует узел управляющего графа типа Запись – запись в  $x$ , из которого дуга  $e_1$  достижима.
- 2) Существует узел управляющего графа типа Чтение – чтение из  $x$ , достижимый от дуги  $e_2$ .
- 3) Не существует узла управляющего графа типа Запись для  $x$  на любом пути от  $e_1$  до  $e_2$ .
- 4) Пара дуг ( $e_1, e_2$ ) образует SESE-регион.

Если в некотором SESE-регионе не содержится значимой для потокового анализа информации, связанной с объектом  $x$ , информации (в нем отсутствуют операции чтения и записи для  $x$ ), то и в DFG, построенном для объекта  $x$  можно не строить дуг и узлов, соответствующих данному региону. На рис. 1 представлены примеры управляющего графа, и графа потока зависимостей, построенные для простой программы. В графе потока зависимостей отображены только зависимости типа “Запись-запись” и “Запись-чтение”. Каждый узел управляющего графа, изображенного на рисунке, содержащий присваивание, выделяется как SESE-регион. Кроме того, можно выделить SESE-регион, состоящий из 4 узлов УГ, который соответствует конструкции if-else в исходном тексте программы. Этот регион не содержит обращений к переменной  $x$ , а потому в графе потока зависимостей дуга, соответствующая зависимости “Запись-запись” для  $x$  просто его обходит.

## 2. Память, требуемая для построения DFG

Теоретически, объем памяти, требуемой для построения графа потока зависимостей, оценивается как  $O(EV)$ , где  $E$  – количество дуг в управляющем графе, а  $V$  – количество объектов, для которых построен DFG. Если перейти к абсолютным значениям, то объем памяти, требуемый для построения DFG, может достигать значения  $M=(N*n+E*e)*V$ , где  $n$  и  $e$  – память, требуемая компилятору для отображения дуги и узла DFG соответственно, а  $N$  – общее число узлов типа Схождение и Расхождение в управляющем графе программы.



Условные обозначения.

- — Дуга управляющего графа.
- — Узел схождения зависимостей “Merge”.
- — Узел расхождения зависимостей “Switch”.
- - - - -→ — Дуга графа зависимостей.

Рис. 1. Пример построения управляющего графа и DFG для простой программы.

При выводе формулы для  $M$  предполагалось, что для операций промежуточного представления не строятся отдельные DFG узлы (как это и показано на рис. 1). Разумеется, в реальных программах значение требуемой для построения DFG памяти практически никогда не достигает  $M$ . Тем не менее, при построении DFG для реальных программ объем требуемой памяти зачастую достигает значений, совершенно недопустимых в промышленных компиляторах. На рис. 2 представлена таблица, в которой отображены экспериментальные данные, полученные для избранных процедур пакета тестов Specint92, транслируемых оптимизирующим компилятором проекта “Эльбрус 3М”. Показана выборка из пяти процедур, построение DFG для которых потребовало наибольшего количества памяти. Оказалось, что все такие процедуры принадлежат тестовой задаче gcc (Gnu C Compiler). Для каждой процедуры приведено общее количество дуг в управляющем графе и общее количество объектов, для которых строился DFG.

Табл. 1.

Информация о 5 процедурах, DFG которых занимает наибольший объем памяти

Имя процедуры	Число дуг в УГ	Число объектов	Занимаемая память Мб
085.gcc/recog_6	2843	736	135
085.gcc/find_reloads	2417	534	57
085.gcc/expand_expr	2118	447	43
085.gcc/memory_address	2129	228	32
085.gcc/yylex	1538	256	16

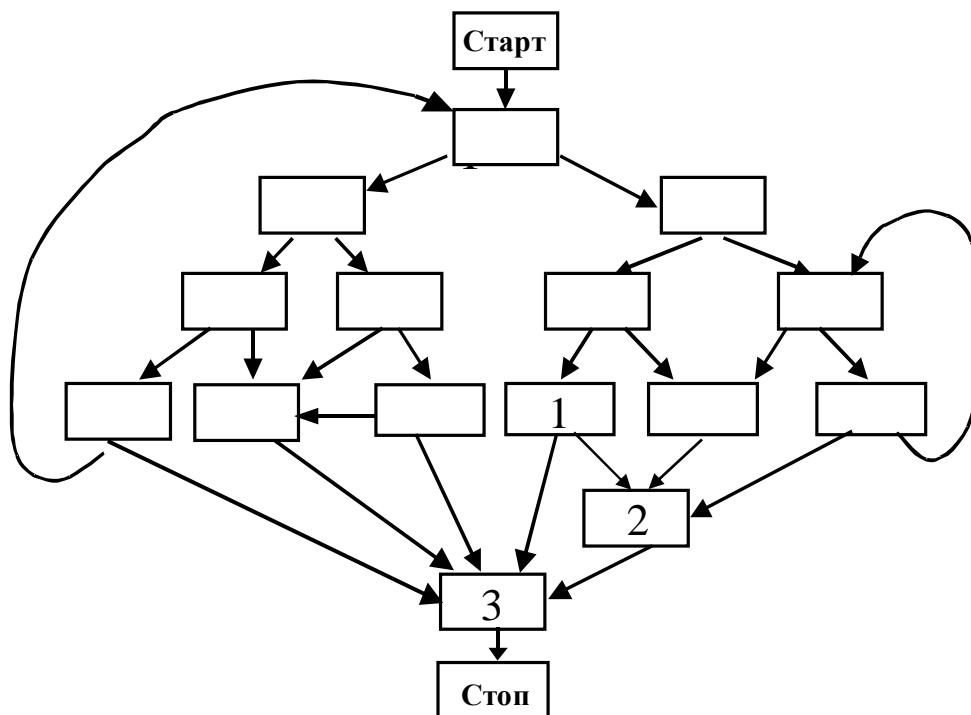


Рис. 2. Пример управляющего графа, который не может быть разбит на SESE-регионы.

Следует отметить, что DFG строился не только для объектов, заведенных в пользовательской программе, но и для объектов, созданных компилятором (в частности соответствующих виртуальным регистрам), дуга и узел графа потока зависимостей занимали 58 и 62 байта соответственно. Абсолютное количество памяти, необходимое для построения DFG для процедуры, сильно зависит от формы управляющего графа и типа уже проведенных компилятором оптимизаций (такие оптимизации как, например, inline могут сильно влиять на размер процедуры и количество объектов, для которых строится DFG).

Детальный анализ управляющих графов приведенных выше процедур позволил найти основную причину необходимости относительно большого количества памяти для

построения DFG. Оказалось, что каждая из них содержит один или несколько SESE участков, содержащих большое количество узлов, которые не принадлежат никаким вложенным SESE-регионам. На рис. 3 изображен пример SESE-участка, который без специальных преобразований УГ нельзя разбить на вложенные SESE-регионы. (Стоит отметить, что путем вставки пустого узла перед узлом 3 с перенаправлением на него дуг от узлов 1 и 2, один нетривиальный SESE регион, выделить можно; преобразования же, заключающиеся в выделении пустых узлов Схождение и Расхождение приводят к образованию лишь тривиальных SESE регионов.) Допустим, мы производим построение DFG для некоторого объекта  $x$  в таком регионе. В каком бы узле управляющего графа не находилась операция записи в этот объект, для всех узлов единственного SESE региона будут построены узлы Расхождение и/или Схождение DFG. Эти узлы будут построены и для всех других нитей DFG, в объекты которых происходит запись внутри региона. Понятно, что информация о путях управления в программе просто дублируется для всех объектов, что приводит к большим затратам памяти. В таблице 2 для каждой из рассматриваемых процедур приведены данные по трем SESE участкам, при построении DFG для которых (включая вложенные регионы) потребовалось максимальное количество памяти. Если сопоставить данные таблиц 1 и 2, то можно заметить, что основное количество памяти расходуется именно на построение DFG в регионах, которые плохо разделяются на вложенные регионы и содержат большое количество объектов для построения DFG. Так, например, функции `recog_6` и `memory_address` вообще состоят из одного большого SESE участка, который разбивается только на тривиальные вложенные SESE регионы. Для тривиальных SESE регионов узлы DFG Схождение и Расхождение не строятся, и память, соответственно практически не тратится.

Табл. 2.

Информация об SESE-регионах, построение DFG для которых требует наибольших затрат памяти

Условные обозначения:

A – общее число узлов УГ, принадлежащих SESE-региону.

B – число узлов УГ, принадлежащих региону, но не принадлежащих вложенным регионам.

V – число объектов, для которых строится DFG в регионе.

M – память в мегабайтах, требуемая для построения DFG для региона.

SESE регион	A	B	V	M
recog_6/SESE1	2173	1050	736	135
recog_6/SESE2	1	1	-	~0
recog_6/SESE3	1	1	-	~0
find_reloads/SESE1	1846	580	534	56
find_reloads/SESE2	29	8	46	~0
find_reloads/SESE3	28	14	16	~0
expand_expr/SESE1	1619	586	429	43
expand_expr/SESE2	20	7	10	~0
expand_expr/SESE3	10	7	5	~0
memory_address/SESE1	1577	757	228	32
memory_address/SESE2	1	1	-	~0
memory_address/SESE3	1	1	-	~0
yylex/SESE1	1119	324	245	14
yylex/SESE2	33	9	15	~0

### 3. Новая реализация графа потока зависимостей

Анализ результатов, изложенных в разделе 2, выявил необходимость создания новой реализации графа потока зависимостей, которая бы позволила строить его для более широкого круга процедур, учитывая ограниченность ресурсов, предоставляемых компилятору. Основным недостатком классической реализации – ненужное дублирование информации, связанной с путями управления в УГ. Чтобы этого избежать, был предложен следующий подход. Узлы типа Схождение, соответствующие разным объектам, но созданные для одного и того же узла управляющего графа объединяются в один узел, то же самое происходит и с узлами Расхождение. Когда один и тот же узел соответствует многим объектам, появляется возможность объединять и дуги DFG. То есть если две дуги DFG имеют одинаковые узлы исхода и назначения, то они объединяются в одну. Данный подход показан на рисунке 5. Будем называть дугу DFG, соответствующую нескольким переменным – мультидугой. Мультидуга не может входить в узел (или выходить из узла) DFG, соответствующего операции промежуточного представления. Мультидуги соединяют всегда лишь узлы типов Схождение/Расхождение.

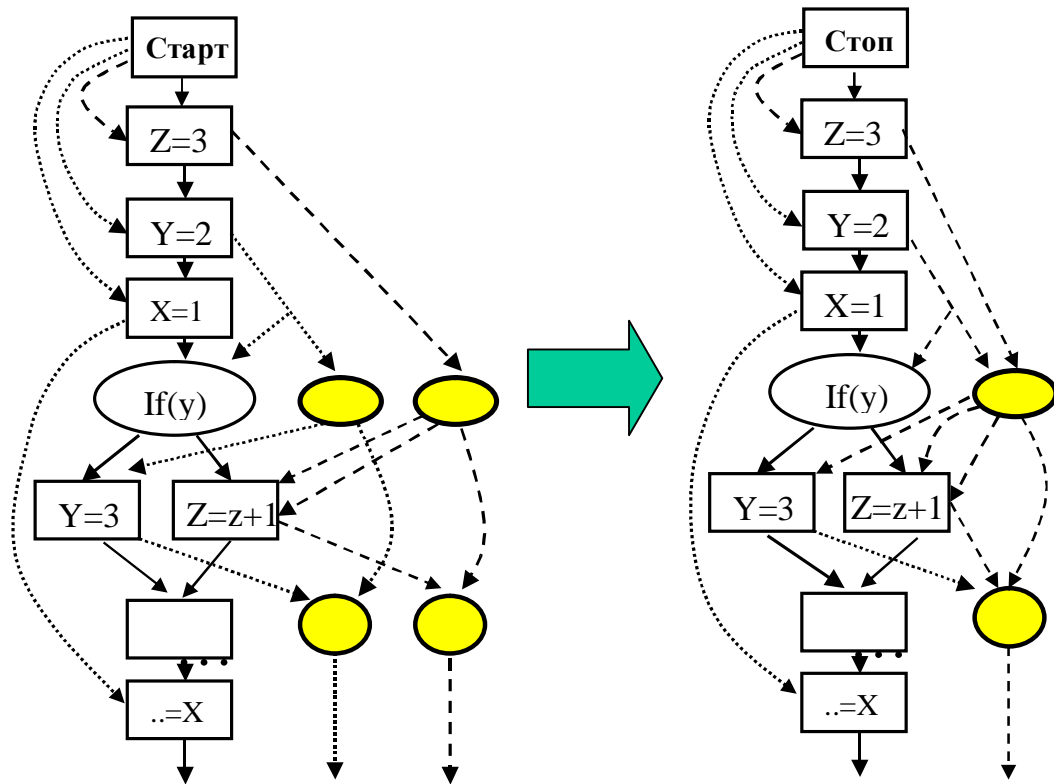


Рис. 3. Реализация DFG с совмещенными узлами Схождение и Расхождение.

Легко заметить, что введение мультидуг сильно осложняет обход DFG. Интерфейс, который должен быть предоставлен пользователю, в обязательном порядке включает в себя

получение дуги DFG по узлу DFG и направлению дуги. Это тривиальным образом реализуется, когда множество узлов и дуг, соответствующее заданному объекту, объединяется в отдельный граф (классическая реализация). В нашей реализации одна и та же дуга (или узел) может соответствовать многим объектам, поэтому целесообразно объединить все дуги и узлы в единый граф. Интерфейс обхода теперь должен измениться, так как чтобы получить дугу по узлу, нужно еще и задать номер объекта. Чтобы найти дугу, идущую в данном направлении, нужно перебрать **все** дуги для данного направления и выбрать из них те, которые соответствуют задаваемому номеру объекта. Такой перебор, очевидно, очень дорог. Для того, чтобы избежать этого, следует каждому узлу DFG поставить в соответствие некоторую структуру данных, с помощью которой по номеру объекта можно было бы получить ссылку на список дуг, соответствующих этому номеру. В качестве такой структуры данных можно использовать массив или список с быстрым поиском. Данную схему обхода дуг DFG графа иллюстрирует рис. 4 и реализует следующий ниже алгоритм.

**Алгоритм 1: получение дуг DFG по узлу DFG, направлению дуги, номеру объекта (реализация на списках с быстрым поиском).**

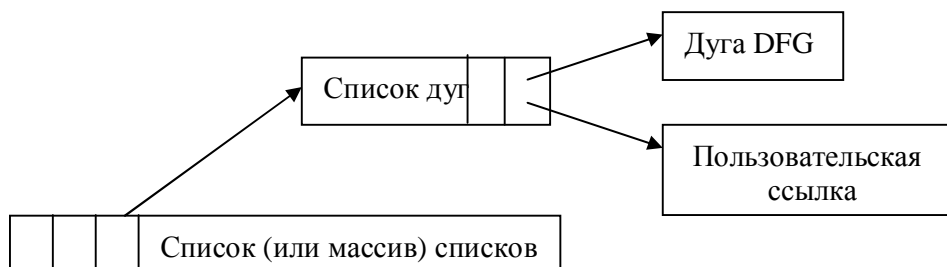
N – узел DFG;  
 Dir – направление обхода DFG  
 ObjNum – номер объекта  
 DFGEEdge – искомая дуга DFG  
 DFGEEdgeList – список дуг DFG  
 Router – список (или массив) списков дуг DFG  
 ListUnit – элемент списка  
 GetDFGNodeRouter() – получение от узла DFG списка (или массива) списков дуг DFG, соответствующего заданному направлению.  
 GetFastSearchListUnitInfo() - получение пользовательской ссылки от списка с быстрым поиском  
 GetFirstListUnit() – получение первого элемента списка  
 GetNextListUnit() – получение следующего элемента списка  
 GetListUnitInfo() – получение пользовательской ссылки от элемента списка

```

GetFirstDFGEEdge( N, Dir, ObjNum)
{
  1: DFGEEdge = Null
  1: Router = GetDFGNodeRouter( N, Dir)
  2: DFGEEdgesList = GetFastSearchListUnitInfo( Router, ObjNum)
  3: ListUnit = GetFirstListUnit( DFGEEdgesList)
  4: If ( ListUnit != Null ) DFGEEdge = GetListUnitInfo( ListUnit)
}
  
```

```

GetNextDFGEEdge( ListUnit)
{
  1: ListUnit = GetNextListUnit( ListUnit);
  2: if ( ListUnit != Null ) DFGEEdge = GetListUnitInfo( ListUnit)
}
  
```



Номер объекта

Рис. 4 . Схема поиска DFG дуги с использованием двоичного дерева.

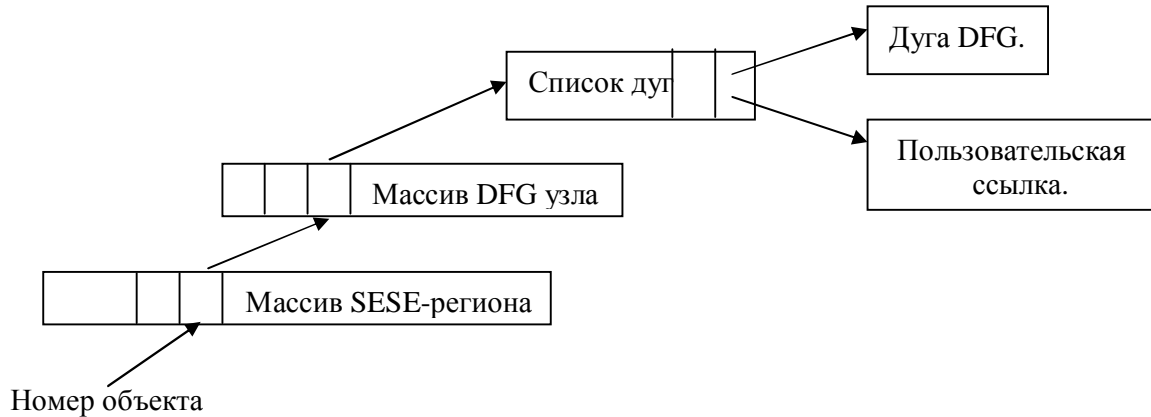


Рис. 5. Схема поиска DFG дуги с использованием массивов.

Использование списка с быстрым поиском, который реализован на сбалансированных двоичных деревьях, ухудшает асимптотическую оценку сложности построения и обхода DFG до  $O(E \log(V))$ , где  $V$  – число объектов,  $E$  – число дуг в управляющем графе. Использование массивов не оптимально, так как массив заводится с длиной, равной общему количеству объектов построения, хотя в данном SESE-участке может находиться ссылка только на часть из них. Здесь может быть предложена следующая реализация. Каждому узлу управляющего графа, для которого создается узел DFG типа Схождение или Расхождение ставится в соответствие массив. Его длина равна количеству объектов, на которые есть ссылки в SESE регионе, включающему узел. Каждому SESE-участку может быть поставлен в соответствие массив с длиной равной общему количеству объектов – массив переадресации. Поиск дуги в этом случае осуществляется следующим образом. По номеру объекта в массиве SESE региона находится локальный номер объекта для региона. По этому номеру в массиве узла находится ссылка на список, в котором находятся интересующие нас дуги. Эту реализацию обхода дуг DFG иллюстрирует рис. 5 и реализует следующий ниже алгоритм.

**Алгоритм 2: получение дуг DFG по узлу DFG, направлению дуги, номеру объекта (реализация с использованием массива переадресации).**

В дополнение к обозначениям, введенным при описании алгоритма 1, добавим еще некоторые обозначения.

SESE – SESE регион

SESEArray – массив переадресации SESE региона

LocalObjNum – локальный номер объекта

DFGNodeArray – массив, связанный с DFG узлом

GetDFGNodeSESE() – Получение SESE региона, которому принадлежит узел

GetSESEArray() – Получение массива SESE региона

GetDFGNodeArray – Получение массива DFG узла

GetArrayUnitInfo – Получение пользовательской ссылки элемента массива

```

GetFirstDFGEdge( N, Dir, ObjNum)
{
  1: DFGEdge = Null
  2: SESER = GetDFGNodeSESER( N)
  3: SESERArray = GetSESERArray(SESER)
  4: LocalObjNum = GetArrayUnitInfo( SESERArray, ObjNum)
  5: DFGNodeArray = GetDFGNodeArray(N)
  6: DFGEdgesList = GetArrayUnitInfo( DFGNodeArray, LocalObjNum)
  7: ListUnit = GetFirstListUnit( DFGEdgesList)
  8: If ( List_Unit != Null ) DFGEdge = GetListUnitInfo( ListUnit)
}

```

```

GetNextDFGEdge( ListUnit)
{
  1: ListUnit = GetNextListUnit( ListUnit);
  2: if ( List_Unit != Null ) DFG_Edge = GetListUnitInfo( ListUnit)
}

```

При классической реализации графа потока зависимостей нет проблемы хранения результатов потокового анализа. Каждой дуге DFG можно поставить в соответствие объект, в котором и будет храниться соответствующая информация (например, значение константы в оптимизации подстановки констант). В усовершенствованной реализации мы уже не можем так поступить, поскольку дуга может быть одна, а информация, в общем случае, различна для разных объектов. Можно предложить следующий выход из ситуации. Хранить ссылку на пользовательский объект можно в списке дуг (рис. 4 и 5). Одна мультидуга может принадлежать  $V$  объектам, но всегда будет существовать  $V$  уникальных ссылок на нее и столько же ячеек памяти для хранения информации потокового анализа

#### 4. Результаты экспериментов

В табл. 3 приведены экспериментальные данные, полученные для избранных (тех же, что и в табл. 1) процедур пакета тестов Specint92, транслируемых оптимизирующим компилятором проекта “Эльбрус 3М”. В таблице приведены значения памяти, которая потребовалась для построения графа потока зависимостей предложенного нами вида. Результаты приведены для реализации DFG на списках с быстрым поиском [3].

Табл. 3.

Сводная таблица экспериментальных данных.

Условные обозначения:

M1 – память, занимаемая DFG старой реализации

M2 – память, занимаемая DFG предложенной нами реализации

Имя процедуры	M1	M2	M1/M2
085.gcc/recog_6	135	51	2.6
085.gcc/find_reloads	57	23	2.5
085.gcc/expand_expr	43	17	2.5
085.gcc/memory_address	32	13	2.4
085.gcc/yylex	16	7	2.3

## Заключение

В работе был исследован вопрос применимости Графа Потока Зависимостей в промышленных компиляторах. Была предложена новая реализация DFG, позволившая уменьшить количество занимаемой представлением памяти более чем в два раза по сравнению с оригинальной реализацией для набора тестов пакета Specint92.

Результаты данной работы позволяют использовать граф потока зависимостей для анализа более широкого круга программ, учитывая ограниченность предоставляемых компилятору ресурсов.

## Список литературы

1. **Muchnick, Steven S.** Advanced Compiler Design and Implementation – Morgan Kauffman, San Francisco, 1997. Chapter 7.2.
2. **Johnson, Richard Craig.** Efficient Program Analysis Using Dependence Flow Graphs – Ph.D. dissertation, Cornell University.
3. **Кормен Т., Лейзерсон Ч., Ривест Р.** Алгоритмы: построение и анализ – М.: МЦНМО, 1999.