УДК 681.32

В. А. Кирьянчиков
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Методика построения операционных графовых моделей программ

Рассматриваются назначение, способы представления и методика построения операционной графовой модели программы (ОГМП). Указываются основные этапы построения ОГМП и отмечаются проблемы, возникающие при их выполнении. Описывается программное средство (ПС) поддержки построения ОГМП, разработанное и используемое в лабораторном практикуме кафедры МОЭВМ при расчете характеристик эффективности выполнения программ. Отмечаются особенности реализации основных компонентов ПС поддержки построения ОГМП, таких как модуль расстановки контрольных точек (КТ), обеспечивающий внедрение вызовов функции КТ в текст исследуемой программы для ее профилирования и формирование управляющего графа программы (УГП), служащего каркасом для построения ОГМП, а также модуля нагружения УГП, формирующего с помощью программного монитора требуемую ОГМП в виде графа с нагруженными дугами. В заключение предлагается методика проверки адекватности созданной ОГМП, подтверждающая соответствие ее поведения работе исследуемой программы.

Эффективность, операционная графовая модель программы, управляющий граф программы, параметр потребления ресурсов, вероятность выбора маршрута выполнения программы, контрольная точка

Одной из интересных и важных характеристик качества программного обеспечения (ПО), имеющей количественные значения, получаемые в результате расчетов по соответствующим моделям, является эффективность [1]. Эффективность характеризует количество ресурсов, потребляемых при выполнении программного средства (ПС). В данной статье в качестве оцениваемого ресурса рассматривается время выполнения программы или ее отдельных функциональных участков. Поэтому аналогом эффективности может служить также такая широко используемая характеристика, как производительность.

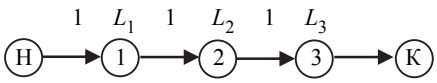
При расчете характеристик эффективности программ в качестве модели программы используется операционная графовая модель программы (ОГМП), представляющая собой ориентированный граф с конечным числом вершин и дуг, которым сопоставляются параметры потребления ресурсов при выполнении программы и параметры, определяющие порядок и вероятность выбора путей выполнения программы [2].

При изображении ОГМП обычно используют два основных варианта представления: графы с нагруженными вершинами (ГНВ) и графы с нагруженными дугами (ГНД). В ГНВ вершинам графа сопоставляются процессы, потребляющие ресурсы, и для каждой вершины задается параметр, характеризующий количество потребляемого ресурса, поэтому часто говорят, что вершина нагружается параметром потребления ресурса. В качестве процесса может рассматриваться команда процессора, оператор языка высокого уровня (ЯВУ), фрагмент программы или подпрограмма. Параметром, определяющим количество ресурса, может служить: количество команд, количество тактов процессора, время выполнения команды, оператора или фрагмента программы, задаваемые либо как детерминированные значения, либо в виде статистических характеристик (среднего значения и дисперсии). Дуги в ГНВ задают порядок выполнения процессов и, если выбор процессов имеет статистический характер, то дугам сопоставляется вероятность выбора процесса для очередного выполнения или выбора маршрута выполнения программы. Для типовых управляющих конструкций программ их модели в виде ГНВ будут иметь вид, показанный на рис. 1, где используются обозначения: Н, К – соответственно, начальная и конечная вершины; L_i – количество ресурса, сопоставляемое i-й вершине; P_{ij}, Q_{ij} или число ≤ 1 – вероятность выбора дуги, соединяющей вершины i и j; N – число повторений цикла.

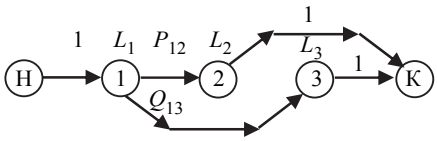
В ГНД выполняемые процессы сопоставляются дугам, поэтому для дуг задаются и параметры, определяющие потребление ресурсов, и вероятности выбора альтернативных процессов. Вершины графа служат только разделителями выполняемых процессов.

Модели типовых управляющих конструкций программ, представленные в виде ГНД, будут иметь вид, показанный на рис. 2, где используются те же обозначения, что и на рис. 1.

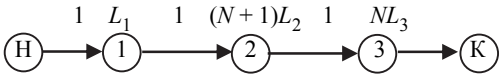
1. Следование



2. Ветвление



3. Цикл с фиксированным числом повторений (ЦФЧП)



Здесь L_1 – ресурсы, потребляемые при инициализации цикла; L_2 – ресурсы, потребляемые при проверке условия входа в цикл и выполнении перехода; L_3 – ресурсы, потребляемые при выполнении тела цикла.

4. Цикл с неизвестным числом повторений (ЦНЧП)

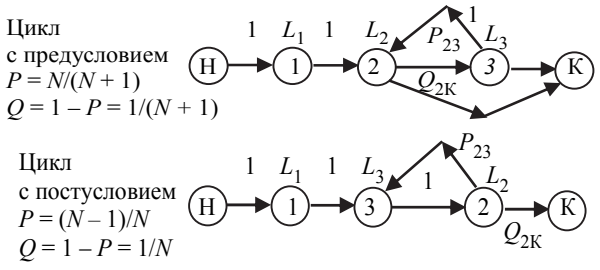


Рис. 1

Методика построения операционных графовых моделей программ. Для построения ОГМП надо выполнить следующие три этапа:

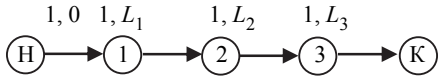
1) построить управляющий граф программы (УГП), который станет каркасом модели, требующей нагружения своих элементов;

2) оценить параметры потребления ресурсов фрагментами программы, сопоставляемыми вершинам или дугам графа;

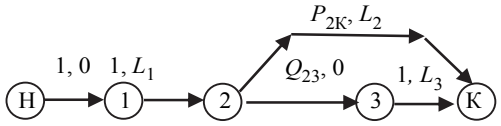
3) оценить вероятности выбора маршрутов выполнения программы.

Построение УГП осуществляется выделением в программе функциональных участков (ФУ) и сопоставлением им элементов управляющего графа. Возможна автоматизация построения УГП по

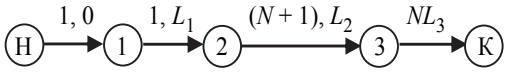
1. Следование



2. Ветвление



3. Цикл с фиксированным числом повторений (ЦФЧП)



4. Цикл с неизвестным числом повторений (ЦНЧП)

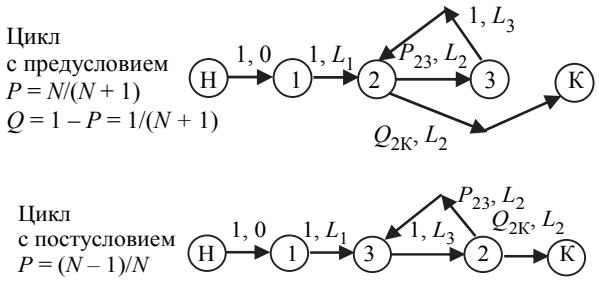


Рис. 2

тексту программы (на рынке ПО есть продукты, позволяющие выполнить построение моделей программ, написанных на процедурных языках, в виде структурного графа), но необходимо еще и нагрузить элементы полученного УГП, что является более сложной задачей построения модели и в известных продуктах отсутствует.

Время выполнения ФУ может оцениваться одним из следующих способов:

– прямым подсчетом по тексту программы в ассемблерном представлении с использованием времен выполнения команд, задаваемых в тактах процессора;

– с использованием программного измерительного монитора (ПИМ) типа профилировщика, в качестве которого может использоваться серийный монитор (*VTune* или аналогичный ему) или разработанный на кафедре МОЭВМ монитор *Sampler*.

При оценке потребления ресурсов посредством прямого подсчета по ассемблерному коду программы возникают трудности из-за оптимизации программы современными компиляторами, затрудняющей сопоставление оператора ЯВУ и ассемблерного кода, в который он транслируется. При отключении оптимизации такое сопоставление провести можно, однако получаемые в этом случае оценки будут не совсем корректны.

При получении оценок потребления ресурса времени функциональным участком с использованием ПИМ таких проблем не возникает. Однако при работе с профилировщиком в результаты измерений вносятся искажения из-за потребления им ресурсов ЭВМ на собственные нужды. В некоторых ПИМ (например, в ПИМ *Sampler*) удается снизить влияние этого негативного фактора за счет выполнения коррекции результатов измерений. Поэтому в целом оценка потребления ресурса с помощью ПИМ является предпочтительной.

Оценка вероятностей для каждого ветвления в УГП является наиболее трудной задачей и не всегда может быть выполнена аналитически, так как выбор ветвей может сложным образом зависеть от структуры и параметров программы. Тогда применяется имитационное моделирование – значения параметров, влияющих на события, определяющие выбор ветвей, задаются с помощью датчика случайных чисел с требуемым законом распределения и методом статистических испытаний определяются вероятности наступления событий.

Рассмотрим пример проведения испытаний и обработки их результатов. Допустим, в качестве модели фрагмента программы используется показанный на рис. 3 граф с нагруженными вершинами, соответствующий циклу *While*, имеющему вид

```
While (cond ())
{
body ();
// это тело цикла
}
```

и требуется получить оценку вероятностей P – выбора ветви входа в цикл и Q – выхода из цикла. С помощью профилировщика можно измерить параметры:

1) число выполнений данного фрагмента программы (параметры типа N_i);

2) количество передач управления на данный фрагмент и из него (параметры типа N_{ij}).

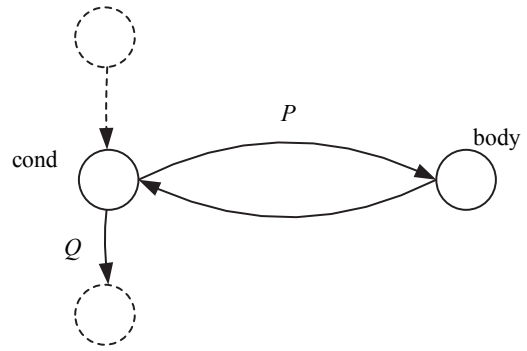


Рис. 3

Пусть в исходный текст для контроля прохождения путей внедряются вызовы гипотетической функции *CtrPoint* (*intNum*), реализующей счетчик числа проходов (т. е. фиксирующей, сколько раз произошел вызов функции *CtrPoint* с конкретным номером *Num*), и пусть эти счетчики расставлены в тексте программы следующим образом:

```
CtrPoint (1);
while (cond ())
{
CtrPoint (2);
body ();
// это тело цикла
CtrPoint (3);
}
CtrPoint (4);
```

Пусть при прогоне программы на разных наборах входных данных были получены данные, представленные в таблице.

№ *CtrPoint*	Среднее число выполнений
1	5
2	30
3	30
4	5

Из этого можно заключить, что при проверке условия выполнения цикла значения true и false получены, соответственно, 30 и 5 раз ($N_{\text{true}} = N_2 = N_3 = 30$, $N_{\text{false}} = N_1 = N_4 = 5$), а в совокупности условие проверялось $N = N_{\text{true}} + N_{\text{false}} = 35$ раз. Тогда вероятности входа в цикл и выхода из него будут, соответственно, равны $P = N_{\text{true}} / N = 30/35 = 6/7$; $Q = N_{\text{false}} / N = 5/35 = 1/7$.

Итак, чтобы получить вероятности ветвлений, необходимо измерить значения «параметров типа N_i».

Расчет вероятностей ветвлений также зависит от подхода к описанию поведения программ. Обычно используют марковский подход к поведению программы, при котором не учитывается предыстория выполнения программы до попадания в некоторое состояние, задаваемое вершиной графа. Соответственно, при этом делаются существенные допущения, связанные с тем, что события, используемые в условиях (предикатах), рассматриваемых в разных блоках принятия решений, считаются независимыми.

Другим подходом к поведению программ может служить подход, учитывающий семантику выполнения программы и реальные значения обрабатываемых программой данных. Этот подход,

который в плане автоматизации находится в стадии разработки, можно назвать семантическим.

Рассмотрим пример, показывающий, что допущение о независимости событий, используемых в разных блоках принятия решений, приводит к ошибкам в оценке вероятностей выполнения маршрутов.

Пусть программа представлена блок-схемой (рис. 4), где приняты обозначения: *Ai* – номер блока; *Pi* – вероятности выбора прямой ветви в блоке принятия решений; *Mi* – номер маршрута выполнения программы. Считается, что входная переменная *X* – случайная величина, равномерно распределенная в интервале [0; 10], а значения проверяемых границ *C*1 и *C*2 равны: *C*1 = 4, *C*2 = 8.

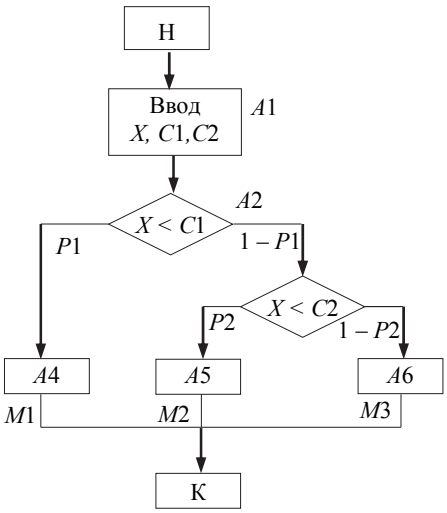


Рис. 4

При использовании марковского подхода вероятности выполнения маршрутов можно рассчитать следующим образом:

$$Prob\ (M1{:}A1{-}A2{-}A4) = P1 = 0.4;$$
$$Prob\ (M2{:}A1{-}A2{-}A3{-}A5) =$$
$$= (1{-}P1)P2 = 0.6{\cdot}0.8 = 0.48;$$
$$Prob(M3{:}A1{-}A2{-}A3{-}A6) =$$
$$= (1{-}P1)(1{-}P2) = 0.6{\cdot}0.2 = 0.12.$$

В случае семантического подхода для расчета вероятностей выполнения маршрутов следует использовать соотношения

$$Prob\ (M1) = P1 = 0.4;$$
$$Prob\ (M2) = (1{-}P1){\cdot}Prob\ (x<C2/x>= C1) =$$
$$= 0.6{\cdot}4/6 = 0.4;$$
$$Prob\ (M3) = (1{-}P1){\cdot}Prob\ (x>= C2/x>= C1) =$$
$$= 0.6{\cdot}2/6 = 0.2.$$

Из приведенного примера видно, что использование марковского подхода в данном случае дает некорректные результаты.

Реализация программной поддержки построения ОГМП. На кафедре МОЭВМ разработано программное средство (ПС) поддержки построения ОГМП, выполненное в виде программы, написанной на объектно-ориентированном языке *Ruby*, которая взаимодействует с пользователем посредством графического интерфейса, обрабатывает входной текст целевой программы, представленный на языке C/C++ и в результате получает ОГМП целевой программы.

Основным компонентом ПС поддержки построения ОГМП является модуль «Расстановка КТ» (рис. 5), который принимает на входе исходный текст целевой программы и на выходе формирует модифицированный текст программы с внедренными вызовами функции контрольных точек (КТ) и управляющий граф программы, нагруженный данными о переходах между КТ, по которым позднее будут вычисляться нагрузочные параметры ОГМП, представляемой в виде графа с нагруженными дугами.

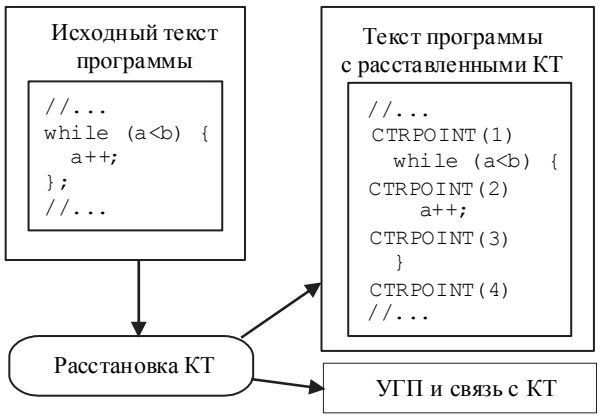


Рис. 5

По сути, данный компонент является транслятором исходного текста программы в другие формы ее представления, который в процессе перевода осуществляет анализ целевой программы на следующих уровнях представления:

1) уровень исходного текста на языке C/C++;

2) уровень токенов (токен определяет тип лексемы и ее позицию в исходном тексте);

3) уровень синтаксического дерева;

4) уровень ОГМП.

При переходе от первого уровня ко второму выполняется лексический анализ, в результате которого формируется поток токенов. При переходе от второго уровня к третьему осуществляется синтаксический анализ потока токенов и формируется синтаксическое дерево программы. Это дерево отражает структуру программы как сово-

купности вложенных друг в друга конструкций языка. При этом каждая конструкция (узел дерева) «знает» о своем местоположении в исходном коде, которое задается токенами, ограничивающими эту конструкцию и ее тело.

При переходе на уровень модели в ходе анализа структуры дерева происходит:

– внедрение в исходный текст вызовов функции КТ, необходимых для расчета параметров нагружения подграфа текущей конструкции (рис. 6);

– построение УГП, дуги которого для каждой конструкции нагружаются данными о переходах между КТ.

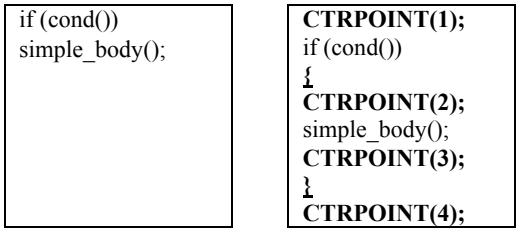


Рис. 6

Способы задания связи УГП и графа переходов. Обозначим через $T[i, j]$ переход между КТ с номерами i и j. Параметры дуги линейного участка, вершина-источник которой имеет только одну исходящую дугу, определяются одним переходом

$$P_{ij} = 1;\ L = L\,(T[i, j]) = L_{ij}.$$

Параметры дуги, входящей в блок принятия решения, зависят от нескольких переходов, которые могут принадлежать одному из множеств:

– *основному множеству UP*, включающему переходы, соответствующие выбору данной дуги;

– *дополняющему множеству DOWN*, включающему переходы, соответствующие выбору дуги, альтернативной по отношению к данной дуге.

Через параметры переходов основного множества вычисляется потребление ресурсов данной дугой

$$L = \left(\sum N_{ij} L_{ij}\right) \Big/ \left(\sum N_{ij}\right),\ \text{где } T[i, j] \in UP.$$

Параметры переходов из дополнительного множества (совместно с переходами основного множества) необходимы для вычисления вероятности

$$P = \left(\sum N_{ij}\right) \Big/ \left(\sum N_{ij} + \sum N_{mn}\right),$$

где $T[i, j] \in UP;\ T[m, n] \in DOWN$.

Заметим, что для дуги линейного участка, связывающей КТ с номерами i и j: $UP = T[i, j]$; $DOWN = \varnothing$.

Таким образом, для задания нагрузки дуг УГП достаточно указать переходы, входящие во множества *UP* и *DOWN* данной дуги.

После расстановки КТ модифицированный текст исследуемого фрагмента вставляется в исходный текст программы. В результате исполнения скомпилированной программы под управлением монитора *Sampler* формируется отчет, содержащий данные о выполненных переходах между КТ (граф переходов).

Далее выполняется модуль «Нагружение УГП» (рис. 7), который принимает на вход созданный *Sampler* граф переходов и построенное ранее описание УГП, на основании которых формирует описание операционной графовой модели программы в виде ГНД.

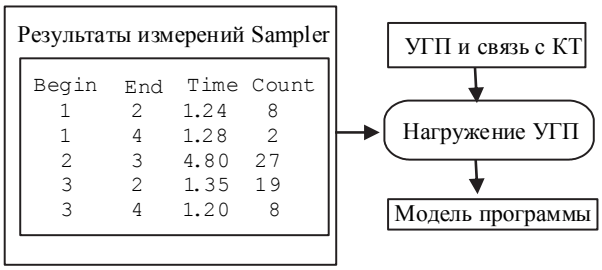


Рис. 7

Далее полученная ОГМП может использоваться в качестве исходной для выбранного метода расчета характеристик эффективности выполнения программы. В лабораторном практикуме на кафедре МОЭВМ расчет характеристик эффективности целевой программы осуществляется методом эквивалентных преобразований с помощью разработанного на кафедре пакета *CSA* (*ComputerSystemAnalysis*).

Проверка адекватности построения ОГМП. Методика проверки адекватности поведения модели выполнению исследуемой программы состоит в сравнении оценок количества потребляемого программой ресурса (времени выполнения), полученных:

1) в результате непосредственного измерения времени выполнения программы;

2) методами расчета оценок времени выполнения по построенной ОГМП.

В первом случае на границах программы (перед входным и после завершающего операторов) внедряются 2 вызова функции КТ и осуществляется измерительный эксперимент – выполнение программы на нескольких различных наборах входных данных под управлением монитора *Sampler*. В качестве результата измерения будет выступать $L_{\text{изм}}$ – среднее время выполнения программы.

Во втором случае применяется аналитический расчет времени выполнения по модели программы, сформированной с использованием средства поддержки построения ОГМП. Полученная оцен-

ка среднего времени выполнения $L_{\text{теор}}$ сравнивается со значением $L_{\text{изм}}$. Близкие значения $L_{\text{теор}}$ и $L_{\text{изм}}$ свидетельствуют об адекватном представлении моделью динамики выполнения программы, а значит и о корректности разработанной методики построения ОГМП в целом.

СПИСОК ЛИТЕРАТУРЫ

1. ГОСТ Р ИСО/МЭК 9126–93. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению. М.: Изд-во стандартов, 1993.

2. Кирьянчиков В. А., Чистяков П. А. Мобильный программный комплекс для моделирования и анализа производительности программ на основе иерархических графовых моделей // Изв. СПбГЭТУ «ЛЭТИ». 2004. № 2. С. 46–50.

V. A. Kirianchikov
Saint Petersburg Electrotechnical University «LETI»

THE TECHNIQUE OF CONSTRUCTING OF OPERATIONAL GRAPH MODEL OF THE PROGRAM

Thepurpose, ways of presentingand technique of constructing of operational graph model of the program (OGMP) is considered. Main stages of the operational graph model constructing are pointed and problems in their implementation are noted. A software tool for support of OGMP constructing is described. The tool was developed and used in the laboratory workshop of the MOEVM department for the calculation of program performance characteristics. The features of the implementation of this software tool main components are noted such as the module of the arrangement of control points (CT), which ensures the introduction of the CT function calls into the text of the program and the creation of the program control graph (PCG) and theprogram loading module forming the required OGMP in the form of a graph with loaded arcs. In conclusion, we propose a procedure for verifying the adequacy of the established OGMP, confirming the correspondence of its behavior to the work of the program under study.

Effectiveness,operational graph model of the program, program control graph,resource consumption parameter, probability of program execution route selection, check point

УДК 621.391

Д. М. Клионский, В. В. Геппенер
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Применение частотно-временного распределения Гильберта–Гуанга для анализа осциллирующих сигналов

Статья посвящена рассмотрению частотно-временного спектра Гильберта–Гуанга для решения задачи адаптивной обработки сигналов. Частотно-временное распределение строится на основе компонент, извлеченных из сигнала с использованием декомпозиции на эмпирические моды (ДЭМ). ДЭМ позволяет получить компоненты разложения непосредственно из исходного сигнала, в связи с чем такого рода разложение является уникальным и не повторяющимся для других сигналов. Эмпирические моды используются для расчета аналитического сигнала, на основе которого далее рассчитывается частотно-временной спектр. При этом вычисляется полная фаза сигнала, которая в дальнейшем применяется для расчета мгновенной частоты. Описан способ нормировки эмпирических мод для вычисления мгновенной частоты, а также необходимая предварительная обработка сигналов. Спектр Гильберта–Гуанга может быть использован для выявления амплитудных и частотных модуляций в сигналах, классификации сигналов, а также выявления областей, в которых сосредоточена энергия. Данный спектр также может применяться для расчета мгновенной плотности энергии как функции времени и маргинального спектра, который зависит только от частоты и является аналогом спектра Фурье для нестационарных сигналов.

Частотно-временное распределение, спектр Гильберта–Гуанга, декомпозиция на эмпирические моды, мгновенная частота, эмпирическая мода, осциллирующий сигнал

Распределение Гильберта–Гуанга относится к классу частотно-временных распределений (УВР), позволяющих добиться лучшего, чем преобразование Фурье и вейвлет-преобразование, частотно-