Параллельное программирование

Графовые модели программ

6.04.2016

Графовые модели программ

Рассмотрим структуры: *информационная структура* алгоритма (программы) и *операционная структура* алгоритма.

Эти структуры можно рассматривать отдельно, можно связать между собой. Каждая из них может быть представлена в виде графа.

Под информационной структурой программы понимается совокупность сведений о том, как отдельные элементы программы связаны между собой.

Операционная структура показывает, каким образом связаны между собой действия в программе или выполняемые операции.

Если мы описываем в виде графа операционную структуру, то в этом случае мы имеем дело с графовой моделью программы, в которой вершины соответствуют действиям (отельным или группам действий), а дуги отражают отношения между этими действиями.

Уровни сложности графовых моделей могут быть различны в силу того, что отдельная вершина графовой модели может представлять и программу целиком, и какие-то её большие или малые фрагменты, и даже отдельные элементарные операции. И на всё это накладывается многообразие связей.

Между действиями программы устанавливаются как правило два типа отношений:

1. Первый определяется фактом выполнения одного действия непосредственно за другим, и если какие-либо два действия находятся в этом отношении, то будем говорить, что между ними установлена связь по управлению или операционная связь.
2. Второй тип отношений связан с использованием одним действием в качестве аргументов результатов других действий. Этот тип отношений определяет информационную связь.

Оба типа отношений вводят на множестве действий в общем случае частичный порядок.

Пример 1

```
1. y := b_1 / a_1
```

```
2. x_1 := y
3. do i := 2, n
4.     x_i := (b_i - c_i * x_{i-1}) / a_i
5.     if (x_i <= y) goto 7
6.     y_i = x_i
7. end do
```

Здесь предполагается, что решается следующая задача:

Есть система линейных алгебраических уравнений $A\,x=b$, где матрица A – это двухдиагональная матрица порядка n. Через $a_1,\dots,a_n$ обозначены диагональные элементы, через $c_2,\dots,c_n$ – поддиагональные элементы (то есть вторая диагональ), $b_1,\dots,b_n$ – элементы вектора b, $x_1,\dots,x_n$ – вектор решений.

Предполагается, что необходимо найти максимальную координату вектора решения.

Для этого примера можно нарисовать управляющий граф, который отражает операционную связь:

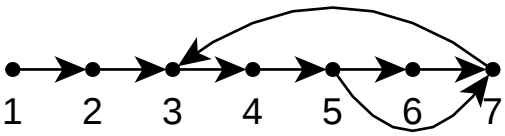


Рис. 1

Одно из основных свойств управляющего графа – независимость от входных данных программы. Граф представляет одну из моделей программы.

Предположим теперь, что исходные данные программы каким-либо образом определены, и все действия выполняются последовательно. Каждое срабатывание каждого оператора (оно не обязательно будет единственным) будем фиксировать отдельной вершиной.

В отличие от управляющего графа, порядок непосредственного срабатывания операторов здесь можно определить точно. Соединяя эти вершины дугами передач управления, получаем ориентированный граф, который называется операционно логической историей программы. Он представляет единственный путь от начальной вершины к конечной, и в операционно логической истории от входных данных зависит практически всё: общее число вершин, количество вершин, соответствующих одному оператору и др.

Для того, чтобы изобразить операционно логическую историю для примера 1, возьмём конкретные данные:

$n=3;\ b_1=0;\ b_2=b_3=a_1=a_2=a_3=c_2=c_3=1$

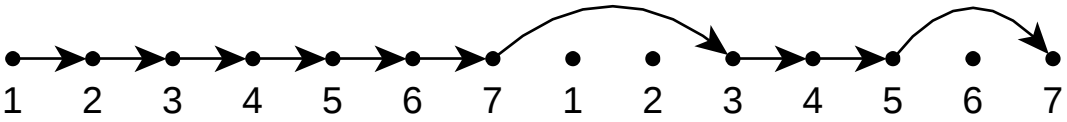


Рис. 2

$n=3;\ b_1=0;\ b_2=c_2=-1;\ b_3=a_1=a_2=a_3=c_3=1$

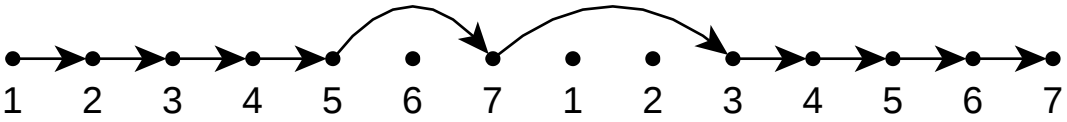
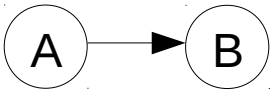


Рис. 3

Операционно логическую историю можно преобразовать следующим образом: все вершины, отвечающие одному оператору программы, объединим в одну с сохранением входных и выходных дуг и ликвидацией образующихся кратных дуг. Полученный в результате граф является подграфом графа управления программы (= управляющего графа), соответствующего конкретным входным данным.

Объединяя все операционно логические истории программы можно тем не менее не получить всего графа управления. Это говорит о том, что программа плохо спроектирована. В данном примере программа спроектирована хорошо =)

Объединяя графы на рисунках 2 и 3 так, как это написано, получим граф на рисунке 1.



Две вершины A и B в операционном отношении соединяются направленой дугой тогда и только тогда, когда оператор, соответствующий вершине B, может быть выполнен сразу после оператора, соответствующего вершине A.

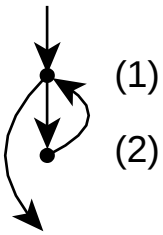
Итак, в графе управления программы (в управляющем графе) вершинами являются операторы, а дуги отражают операционное отношение.

13.04.2016

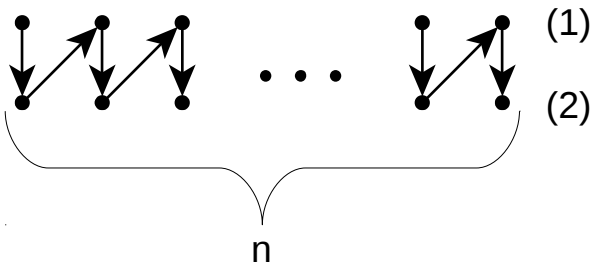
Пример 2

```
for (i = 0 ; i < n; ++i) {
     A[i] := A[i = 1] + 2      (1)
     B[i] := B[i] + A[i]       (2)
}
```

Граф управления выглядит следующим образом:



Операционная история строится на основе оператора ...



Операционное отношение.

Рассмотрим в примере 1 только те операторы, которые осуществляют переработку информации. Их называют преобразователями.

В качестве отношения между преобразователями будем брать отношение информационной зависимости.

Построим граф, вершинами которого будут операторы преобразователей, а дугами соединим такие из этих операторов, между которыми возможна информационная связь. Полученный граф называется информационным графом программы.

Для примера 1 он выглядит следующим образом:

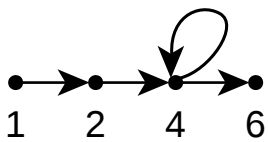


Рис. 4

Информационный граф независим от входных данных и представляет собой одну из моделей программы.

Если мы снова возьмём какие-либо исходные данные и будем наблюдать за выполнением

программы при последовательном вычислении, то каждое срабатывание каждого оператора-преобразователя можно представить отдельной вершиной, и, соединив вершины дугами передач информации, получим ориентированный граф, который называют историей реализации программы.

Для данных рисунка 2 история реализации выглядит следующим образом:

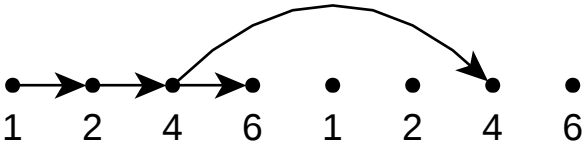


Рис. 5

Для данных рисунка 3 история реализации выглядит следующим образом:

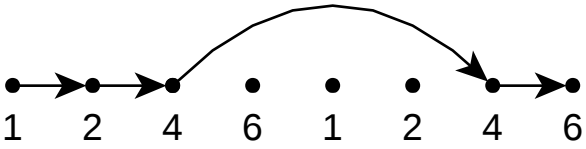


Рис. 6

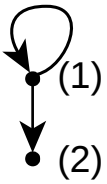
Если объединить истории реализации на рисунке 5 и рисунке 6, то мы получим информационный граф программы, изображённый на рисунке 4.

Определение понятия информационного отношения

Две вершины A и B соединяются направленной дугой тогда и только тогда, когда оператор, соответствующий вершине B, использует в качестве аргумента некоторое значение, вычисленное оператором, соответствующим вершине A.

В информационном графе вершины – это операторы-преобразователи, а дуги – информационное отношение.

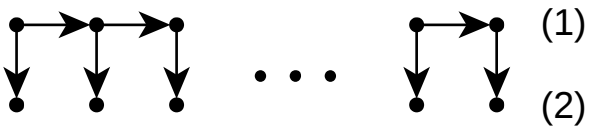
Информационный граф для примера 2:



Подобно операционной истории, можно рассматривать информационная историю.

Информационная история представляет собой ориентированный граф, в котором вершины – это срабатывание операторов, а дуги – это информационное отношение.

Информационная история для примера 2:



Таким образом, наиболее часто используют следующие четыре графовые модели программ:

1. Граф управления
2. Операционная история
3. Информационный граф
4. Информационную история (или история реализации)

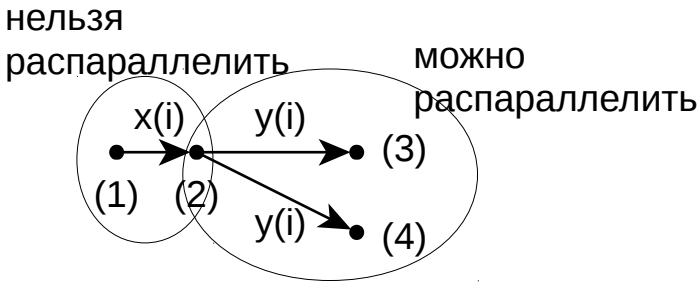
Эти модели существуют для всех программ, на их основе можно строить смешанные модели, в которых одни фрагменты представляются информационным графом или графом управления, а другие – той или иной историей. Существуют преобразования, переводящие одни модели в другие.

Из всех этих моделей наиболее интересной является история реализации программы, потому что, зная эту историю, можно, например, определить множество независимых друг от друга операций, найти подходящее распределение операций по процессорам вычислительной системы, обнаружить узкие места и так далее.

С точки зрения преобразования последовательного алгоритма в параллельный, наиболее важной является информационная структура программы.

Пример 3

```
x(i) = a + b(i)          (1)
y(i) = 2 * x(i) – 3      (2)
t1 = y(i) * y(i) + 1     (3)
t2 = b(i) – y(i) * a     (4)
```



Трудности в определении информационных связей между операциями по тексту программы привели к графовым моделям, называемым графами зависимости. В этих моделях отношение информационной связи заменяется более широким отношением зависимости.

Определение отношения зависимости

Два оператора или две операции называются зависимыми, если при их выполнении имеют место обращения к одной и той же переменной (ячейке памяти).

Так как информационная связь также реализуется через обращение к одной переменной различных операторов, то её можно рассматривать как частный случай зависимости.

Ясно, что устанавливать факт зависимости значительно проще, чем факт информационной связи. Для примера 1 граф зависимости может быть изображён следующим образом:

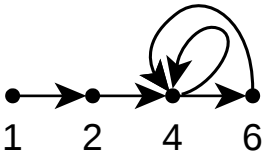


Рис. 7

Если сравнить рисунок 7 с рисунком 4, то видно, что информационный граф является подграфом графа зависимостей.

Заметим, что в примере 1 переменные x и y в разных операторах играют разную роль. Где-то они определяют аргументы, а где-то – результат. Но на графе зависимостей на рисунке 7 это разделение не видно.

Пример 4

Алгоритм суммирования для элементов массива.

```
s := 0                (1)
for i:= 1 to n do
s := s + a[i];        (2)
```

Граф зависимостей для этого примера выглядит следующим образом:

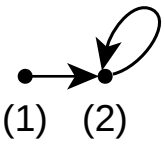


Рис. 8

Этот граф выражает зависимость между операторами.

Если же рассматривать отдельные срабатывания операторов, то для $n=4$ граф зависимостей будет выглядеть следующим образом:

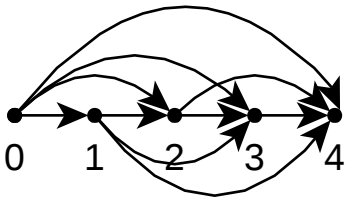


Рис. 9

Здесь видно, что в вершину с номером n входит n дуг. Большое число входящих дуг графа зависимостей в общем случае является платой за лёгкость проверки выполнения зависимостей.

Существует много разновидностей графов зависимостей, одна из них называется граф влияния.

Говорят, что одна операция влияет на другую, если изменением значения переменной, которую вычисляет первая операция, можно изменить значение переменной, которую вычисляет вторая операция.

Будем считать операции вершинами графа и соединим дугами каждую из пар вершин, в которых одна из соответствующих им операций влияет на другую. Пусть направление дуги совпадает с направлением влияния. Такой граф называется графом влияния. Для примера 2 граф влияния при $n=4$ совпадает с графом зависимостей на рисунке 9, но в общем случае эти два графа различны.

Отношения управления, информационной связи и зависимости не являются транзитивными, а отношение влияния транзитивно, за исключением вырожденных случаев. Историю реализации программ можно рассматривать как остовный подграф графа влияния. Более того, граф влияния оказывается транзитивным замыканием истории реализации. Сама же история реализации по отношению к графу влияния представляет его минимальный по числу дуг подграф, обладающий тем свойством, что в обоих графах любые две вершины либо связаны путём, либо не связаны одновременно.

История реализации программы из примера 2 представляет собой минимальный граф зависимостей, который выглядит следующим образом:

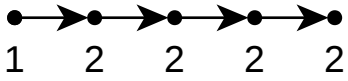


Рис. 10

История реализации программы должна обладать ещё одним важным свойством, а именно, в каждую её вершину входит конечное число дуг, не зависящее от общего числа вершин в истории. Это как правило связано с тем, что любая вычислительная операция имеет конечное число операндов.

Историю реализации также называют графом алгоритма.

Ярусно-параллельная форма графа алгоритма (ЯПФ)

Для определения ресурса параллелизма в программе необходимо представить граф алгоритма в ярусно-параллельной форме. Граф, не имеющий контуров, может быть представлен в ярусно-параллельной форме.

Определение

Ярусно-параллельная форма – это такой вид графа, у которого в верхний нулевой ярус помещены вершины, имеющие только исходящие дуги, а в нижний ярус помещены вершины, имеющие только входящие дуги. На k-ом ярусе ($0<k<n$) помещаются вершины, которые имеют входящие дуги из предыдущих ярусов, среди которых есть хотя бы одна дуга из (k-1)-го яруса.

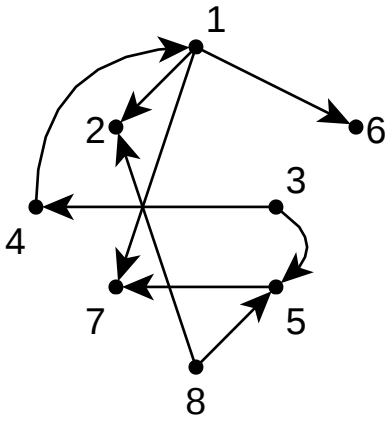
В ярусно-параллельной форме начальная вершина каждой дуги расположена на ярусе с номером меньшим, чем номер яруса конечной вершины.

Рассмотрим алгоритм приведения графа к ярусно-параллельной форме.

1. Составляется матрица смежности графа.
2. Находятся нулевые столбцы (столбцы, содержащие все нули), и вершины, которым эти столбцы соответствуют. Эти вершины помещают в нулевой ярус.
3. Из матрицы смежности удаляют столбцы и строки, соответствующие вершинам нулевого яруса.
4. Повторять шаги 2-3 до тех пор, пока не будут охвачены все вершины.
5. В полученном распределении вершин по ярусам на основании исходной матрицы смежности восстанавливаются дуги между вершинами.

Пример

Привести заданный граф G в ярусно-параллельную форму.



	1	2	3	4	5	6	7	8
1		1				1	1	
2								
3				1	1			
4	1							
5							1	
6								
7								
8		1			1			

В нулевой ярус помещаются вершины 3 и 8, их строки и столбцы вычёркиваем. Получаем таблицу:

	1	2	4	5	6	7
1		1			1	1
2						
4	1					
5						1
6						
7						

В первый ярус помещаются вершины 4 и 5. Получаем:

	1	2	6	7
1		1	1	1
2				
6				
7				

Во второй ярус помещается вершина 1, а в последний ярус – вершины 2, 6, 7.

Восстанавливаем дуги:

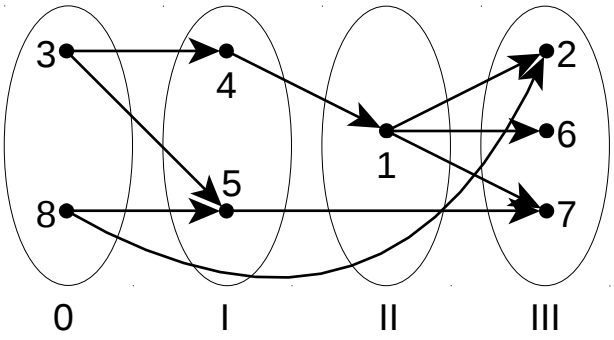


Рис. 12

20.04.2016

Определим следующие понятия:

Высота ярусно-параллельной формы – это число ярусов.

Ширина яруса – это число вершин, расположенных на ярусе.

Ширина ярусно-параллельной формы – максимальная ширина яруса ярусно-параллельной формы.

Высота ярусно-параллельной формы представляет собой сложность параллельной реализации алгоритма.

Ярусно-параллельную форму можно преобразовать к так называемой канонической форме.

Пример

Рассмотрим граф G следующего вида:

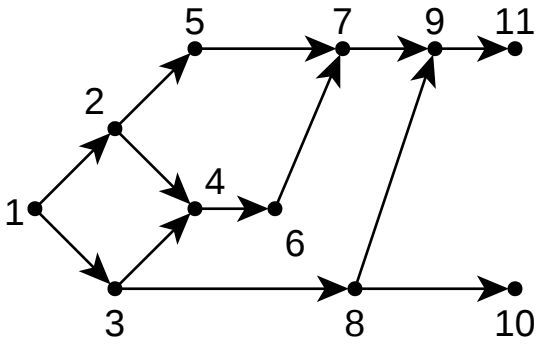


Рис. 13

Ярусно-параллельная форма:

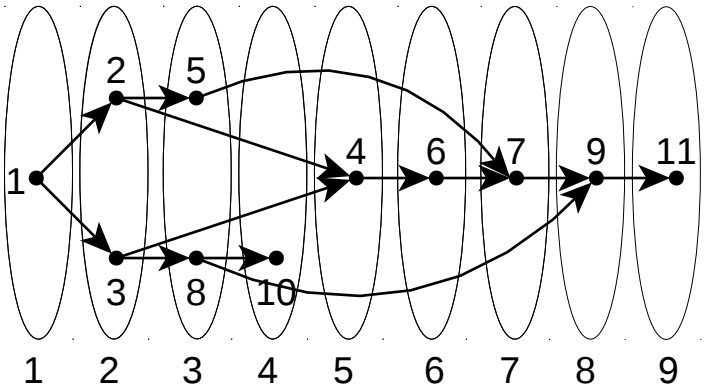


Рис. 14

Каноническая форма ярусно-параллельной формы:

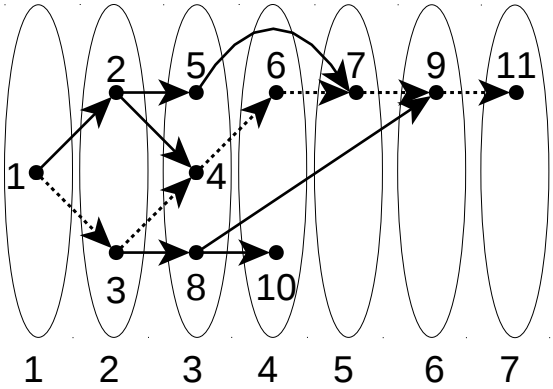


Рис. 15

Отмеченный путь (пунктирные стрелки) называют критическим путём. Высота канонической формы равна длине критического пути плюс один.