УДК 004.021

Сравнительная оценка вариантов алгоритмов построения множеств взаимно независимых операторов

Семёнкин П.С., студент

*Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра Компьютерные системы и сети»*

Научный руководитель Руденко Ю.М., к.т.н., профессор

Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана

v.suzev@bmstu.ru

На практике часто возникает задача выполнения работы за ограниченное время, и это превышает возможности одного исполнителя, мы стремимся разделить эту работу на ряд взаимосвязанных действий и распределить их между одновременно занятыми исполнителями. Такое разделение тем более эффективно, если допускает одновременные и независимые действия всех исполнителей.

Однако прежде чем исследовать конкретные возможности параллельного выполнения алгоритма, необходимо обратить внимание на его сложную логическую структуру. Она определяет зависимость состава порядка выполнения операторов заданных и вырабатываемых значений логических переменных.

Теперь поставим вопрос: можно ли для каждого оператора построить множество тех операторов (в которые входит и он сам), внутри которого имеет смысл решать задачу планирования одновременного независимого параллельного выполнения операторов? Для решения этого вопроса необходимо объединить информацию о логической несовместимости операторов и их информационно-логической связи.

Откажемся от свойства ориентированности графа G , считая что если существует связь $\alpha \rightarrow \beta$ (задающая или транзитивная), то операторы α и β никогда не могут быть выполнены одновременно, параллельно. Достаточно считать что в графе G существует неориентированная дуга, связывающая вершины α и β . В матрице следования S' , соответствующей этому графу, положим $(\alpha, \beta) = (\beta, \alpha) = 1$. Иными словами, матрица S' получается из матрицы S , дополненной транзитивными связями, симметричным отображением ее элементов относительно главной диагонали или сложением матрицы S с ее транспонированной матрицей. На сформированную таким

образом матрицу S' наложим матрицу L , дополненную транзитивными связями, сформировав значение каждого элемента по правилу дизъюнкции. Получим матрицу M .

Симметричную матрицу M , отражающую информационно-логические связи между операторами без учёта их ориентации, а также связи логической несовместимости операторов с учётом всех транзитивных связей, будем называть матрицей независимости.

Операторы a и b будем называть взаимно независимыми операторами (ВНО), если в матрице независимости $M(a,b)=(b,a)=0$.

Операторы $\{a_i\}$, $i=1,\dots,s$, образуют полное множество ВНО, если для любого оператора j , не принадлежащего множеству $\{a_i\}$ существует пара элементов матрицы независимости $M(a_i, j) = (j, a_i) = 1$, i принадлежит множеству $\{1,\dots,s\}$.

В литературе можно найти два алгоритма поиска полных множеств взаимно независимых операторов. Ниже приведены тексты этих алгоритмов:

Алгоритм 1. Нахождение полных множеств ВНО[1].

1. Пусть W – массив полных множеств ВНО. Максимальное полное множество ВНО обозначим через A , а l – число элементов в нём. Очередное формируемое множество ВНО обозначим через D , d – количество элементов в нём. Номер очередного найденного нулевого элемента в строке обозначим через k . Изначально полагаем, что стек пуст, $W=\emptyset$, $A=\emptyset$, $l=0$, $D=\emptyset$, $d=0$, $k=0$.

2. Загружаем очередную i -ю строку в стек, $i=1, \dots, n$, где n – размер матрицы M . Полагаем $D = \{i\}$, $d = 1$, $k = i$. Если все строки обработаны, то выполнение алгоритма заканчивается. Найдены все полные множества ВНО (W) и определено максимальное (A).

3. В строке-вершине стека находим очередной нуль, занимающий позицию $j > k$. Если нуль найден, то переходим к выполнению шага 6, иначе выполняется следующий шаг.

4. Если такого нуля нет или все нули найдены, выполняем проверку на полноту найденного множества D . Если в строке-вершине стека все нули соответствуют всем операторам из D , то найденное множество полное. Производим сохранение $W_m = D$ и переходим к шагу 7. Если в строке-вершине есть хотя бы один нуль, не соответствующий операторам из D , то найденное множество не является полным. Переходим к следующему шагу.

5. Исключаем из стека строку-вершину (не будем забывать, что, исключая строку, мы уничтожаем и текущее значение k и D и возвращаемся к их предыдущим значениям) Если после этого стек исчерпан, выполняем шаг 2. В противном случае выполняем шаг 3.

6. В текущей вершине стека присваиваем $k = j$. Складываем логически (поэлементная дизъюнкция) строку, исключая поля k и D (см. рисунок 5.3), из вершины стека со строкой с номером j – формируем новую вершину стека. В новой вершине стека формируем множество $D = D \cup \{j\}$, $d = d + 1, k = j$. Переходим к шагу 3.

7. Сравниваем значения d и l . Если $d > l$, то $A=D, l=d$. Независимо от результата сравнения переходим к шагу 5.

Конец алгоритма.

Алгоритм 2. Поиска полных множеств ВНО [2].

1. Пусть A – искомое полное множество ВНО, d – число элементов в нем. Первоначально полагаем $A = \emptyset, d = 0$.
2. Пусть B – множество анализируемых операторов, на основе которых формируется очередное полное множество ВНО. Первоначально полагаем $B = \{1\}, C = \{(1, 1), \dots, (1, m)\}, m$ – размер матрицы S .
3. Полагаем $i = 2$, где i – номер очередного оператора, принадлежность которому формируется полному множеству ВНО мы определяем.
4. Формируется множество $D = B \cup i$ и находим по правилу дизъюнкции сумму строк $E = C \vee \{(i, 1), \dots, (i, m)\}$.
5. Анализируем содержит ли строка E нулевые элементы во всех позициях, соответствующих операторам из D ? При данном анализе попутно устанавливаем факт наличия нулевых элементов, не соответствующих множеству операторов из D . Если в строке E нет нулевых элементов, соответствующим всем операторам из D , переходим к шагу 8; в противном случае выполняем следующий шаг.
6. Формируем новые значения $B = D, C = E$.
7. Если в строке E кроме нулевых элементов, соответствующих операторам из D , существуют другие нулевые элементы, выполним шаг 8. В противном случае оказывается найденным полное множество ВНО $A = B$. Здесь может быть произведен выход из алгоритма (конец алгоритма), если необходимо найти очередное полное множество ВНО. Повторных вход в данный алгоритм для

поиска следующего полного множества производится в шаге 10. Если в результате работы данного алгоритма необходимо найти максимальное полное множество ВНО, переходим к шагу 9.

8. Если $i \neq m$, присваиваем значение $i := i + 1$ и выполняем алгоритм с шага 4; в противном случае выполняем следующий шаг.
9. Анализируем превышает ли число r элементов множества B значение d ? Если превышает, полагаем $A = B, d = r$. Независимо от результата анализа выполняем следующий шаг.
10. Сравниваем в обратном порядке элементы множества $B = \{i_1, \dots, i_r\}$ и множества всех операторов $\{1, \dots, m\}$, т.е. выполняем сравнения $i_r = m, i_{r-1} = m - 1$ и т.д. Если при переборе всех элементов B отсутствует несовпадение, выполнение алгоритма заканчивается множеством A является максимальным полным множеством ВНО.
11. Если найдено первое из несовпадений $i_{i-p} \neq m - p$, исключаем из B операторы $i_{i-p}, \dots, i_r$.
12. Формируем значение C , равное сумме (по правилу дизъюнкции) строк матрицы M , соответствующих операторам из множества B :

$$C = \bigvee_{v=1}^{r-p-1} \{(i_v, 1), \dots, (i_v, m)\}.$$
13. Полагаем $i = i_{r-p} + 1$. Продолжаем выполнение алгоритма с шага 4.

Конец алгоритма.

Далее произведем сравнительную оценку эти алгоритмов.

Сравнение алгоритмов можно производить по количеству шагов N , требуемых для решения поставленной задачи. Число шагов N определяется описанием процесса решения задачи некоторой алгоритмической моделью. Это величина математическая, не зависящая от физической реализации алгоритма.

Величину N можно считать «математическим временем» выполнения алгоритма, определяющим физическое время с точностью до константы $t(t_0)$, зависящей от реализации. Такую оценку будем называть вычислительной сложностью алгоритма на некотором наборе входных данных. Для первого алгоритма количество шагов $N=7$, а для второго $N=13$.

Чаще всего для характеристики сложности алгоритма в качестве основных вычислительных ресурсов рассматривают время, затраченное алгоритмом на

вычисление, и использованную память. Понятно, что потребление этих ресурсов может зависеть от размера входных данных.

Сложность однородного алгоритма — это целочисленная функция целого аргумента $t(n)$, равная времени его работы для входных данных размерности n .

К сожалению, эффективность и однородность — требования, зачастую плохо совместимые между собой, поэтому нам необходимо иметь меру сложности, пригодную также для неоднородных алгоритмов. Естественная попытка определить сложность неоднородного алгоритма как функцию $t(R_1, \dots, R_m)$ от всего массива входных данных при ближайшем рассмотрении оказывается неудовлетворительной: эта функция несет в себе огромное количество избыточной информации, с трудом поддается вы-

числению или хотя бы оцениванию в явном виде и позволяет сравнивать между собой различные алгоритмы лишь в исключительных случаях. Поэтому, чтобы получить на самом деле работающее на практике определение сложности, нам необходимо каким-нибудь образом свести эту функцию к функции одного (или в крайнем случае небольшого числа) целого аргумента (как в случае однородных алгоритмов). Имеются два принципиально различных подхода к решению этой задачи. Самый распространенный подход — это рассмотрение сложности в наихудшем случае [3], когда входные данные являются худшими из возможных. Мы полагаем $t(n)$ равным

$$\max \left\{ t(R_1; \dots; R_m) \mid \sum_{i=1}^m [\log_2(|R_i| + 1)] \leq n \right\}$$

Иными словами, $t(n)$ — это то время работы, которое данный алгоритм может гарантировать заведомо, если известно, что суммарная битовая длина входных данных не превышает n .

Широко используется асимптотическая оценка вычислительной сложности алгоритма типа $O(n)$ или $o(n)$. Эта оценка с точностью до константы — множителя и членов более низкого порядка совпадает с точной. Основание для использования таких оценок — большая размерность большинства практических задач.

Рассмотрим трудоемкость этих алгоритмов в худшем случае.

Первый алгоритм содержит проверку на нулевой элемент, путем прохода строк в стеке, сложностью $O(n)$. Если полные множества ВНО не были найдены, то дальше следует поэлементная дизъюнкция строк сложностью $O(n)$. В тоге сложность алгоритма в худшем случае равна — $O(n) \cdot O(n) = O(n^2)$.

Второй алгоритм содержит формирование строки методом поэлементной дизъюнкции, сложностью $O(n)$. Следующим шагом производится поиск нулевого элемента в строке, сложность $O(n)$. Если на этом шаге не найдено полное множество ВНО то, производится сравнение в обратном порядке элементов множества всех операторов и анализируемых операторов, сложностью $O(n)$. В итоге сложность алгоритма в худшем случае равна — $O(n) \cdot O(n) \cdot O(n) = O(n^3)$.

После сравнения алгоритмов видно, что первый алгоритм обладает меньшим числом шагов $N=7$ и меньшей асимптотической сложностью $O(n^2)$.

Список литературы

1. Руденко Ю. М., Волкова Е. А. Вычислительные системы. Архитектура и составляющие ее компоненты. Представление параллельных алгоритмов и оптимизация времени их решения : учеб. пособие МГТУ им. Н. Э. Баумана. - М., 2010. - 211 с. : ил.
2. Барский А.Б. Параллельные процессы в вычислительных системах: планирование и организация. М.: Радио и связь, 1990. – 256 с.: ил.
3. Кузюрин Н.Н., Фомин С.А. Эффективные алгоритмы и сложность вычислений. Авторское электронное издание, 2008. – 345с.: ил.