ПОИСК И РАСПАРАЛЛЕЛИВАНИЕ ЛИНЕЙНЫХ УЧАСТКОВ ВНУТРИ ЦИКЛОВ НА РАЗНЫХ УРОВНЯХ ВЛОЖЕННОСТИ

Ткачев П.Ю., Борзов Д.Б. *(ЮЗГУ, г. Курск, Россия)*
Тел.: +7 (920) 2664651; E-mail: amdathlon64@yandex.ru, borzovdb@kursknet.ru

Abstract: *This article gives the concept of searching and parallelization linear segments within loops on different levels of nesting. Proposed method of solving this problem.*
Key words: *method, parallelization, loop, linear segment, program, multi-processor system, operator.*

Распараллеливание программ — это процесс адаптации алгоритмов, записанных в виде программ, для их эффективного исполнения на вычислительной системе параллельной архитектуры (в последнее время, как правило, на многопроцессорной вычислительной системе). В настоящее время, в связи с развитием многопроцессорных параллельных вычислительных систем, становится необходимым преобразование последовательных наследуемых участков программ в параллельные [1].

В мультипроцессорных системах задача распараллеливания реализована, как правило, централизованно хост–процессором. Известно, что в число его задач кроме компиляции входит также маршрутизация, назначение и перераспределение заданий, реконфигурация системы в случае сбоев и другие функции, обеспечивающие управление системой.

Компиляция является одним из этапов обеспечения распараллеливания сложных последовательных программ, которая отличается вычислительной сложностью, поэтому требуется привлечение дополнительных технических средств.

Использование специализированного вычислительного устройства для параллельного выполнения компиляции и других функций хост-процессора в современных мультипроцессорных системах, повышает эффективность работы вычислительной системы за счет более оптимальной загрузки процессоров. Это приводит к уменьшению объема внутренней памяти, за счет динамического поступления больших фрагментов программ, что положительно сказывается на выполнении других функций. Однако, для повышения скоростных характеристик систем недостаточно выполнять распараллеливание только на уровне команд, необходимо выявлять параллелизм и на уровне данных.

Преобразование последовательных наследуемых участков программ в параллельные, в связи с развитием мультипроцессорных систем, становится очень актуальной задачей [2].

Любую программу можно представить в виде последовательности операторов. Основной задачей при выявлении параллелизма между операторами (то есть могут ли два последовательно идущих оператора выполниться параллельно) является выявление информационной независимости, т.е. не должно одновременно происходить инициирование более чем одной операции записи в ячейку памяти, а также операции чтения и записи в одну ячейку [3].

Математически это можно проверить, вычислив функцию:

$$F(i,k) = (I_i \wedge O_k) \vee (I_k \wedge O_i) \vee (O_i \wedge O_k), \quad (1)$$

где *Ii, Ok* - строки матриц входных/выходных переменных соответственно. В ячейках этих матриц ставится 1, если переменная является входной/выходной для оператора *i. F* – результат проверки на возможность распараллеливания. Если *F=0*, то операторы могут выполняться параллельно, т.к. обрабатывают разные данные.

Данная работа является продолжением исследований начатых в [3].

Известен метод поиска и определения информационно независимых циклов, основанный на этой методике. В этом методе находились циклы, которые можно выполнить параллельно на разных процессорах, так как они обрабатывают разные данные. Однако вопрос распараллеливания линейных участков внутри циклов в этом методе не рассматривался.

Линейным участком считается последовательность операторов без условных/безусловных переходов, вложенных циклов, вызовов подпрограмм, возвратов из подпрограмм и т.д. Рассмотрим небольшой пример.

Исходный фрагмент кода имеет следующий вид:

```
While (K<N) do
{
        A = A+3;
        B = C+5;
        D = A+F;
        While (E>C) do
        {
                M = M+1;
                E = E-2;
        }
        C = C+1;
}
```

В данном примере есть 3 линейных участка, которые могут быть выполнены параллельно на нескольких процессорах при выявлении внутри этих блоков параллелизма на уровне данных. Такие линейные участки назовем «Условно-разделимыми блоками».

На рисунке 1 представлен граф, соответствующий исходному фрагменту кода.

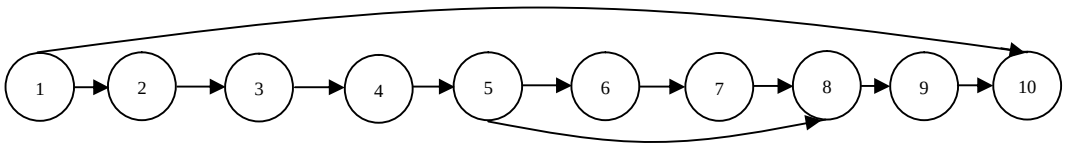

Рис. 1. Граф исходного алгоритма

По графу составим матрицу следования. Множество операторов программы задаётся как $R = \{R_1, R_2, ..., \mathrm{R}_{N_{op}}\}$, где Nop – количество операторов программы, $Nop=N$.

Связи по управлению между операторами программы описываются матрицей следования $Ms = \left\| Ms \right\|_{q \times q}$:

$$Ms_{ik} = \begin{cases} 1, & \text{если } R_i \ \varphi \ R_k \\ 0, & \text{в противном случае.} \end{cases}, \qquad (2)$$

где $k = \overline{1, N}$, символ “φ” обозначает наличие перехода между операторами. Для алгоритма (рис. 1) матрица следования Ms представлена в таблице 1. В ячейке на

пересечении строки i и столбца k матрицы следования ставится 1, если из i-того оператора следует переход в k-тый.

В нашем случае матрица следования имеет следующий вид:

Таблица 1 - Матрица следования *Ms*

№ оператора	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0
5	0	0	0	1	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	1	0	1	0	0	0
9	0	0	0	0	0	0	0	1	0	0
10	1	0	0	0	0	0	0	0	1	0

Алгоритм поиска линейных участков по матрице следования можно представить следующим образом:

1. Ввести счетчик столбцов *i,* и счетчик строк *j* с начальным значением 1;
2. Ввести вспомогательные переменные *s1* и *s2* для хранения границ анализируемого блока;
3. Если *Ms[i][j+1]=1* тогда п.4, иначе п.8;
4. Ищем 1 в строке *j+1* кроме позиции *Ms[i][j+1]*; Если не нашли, то п.5, иначе п.8;
5. Ищем 1 в столбце *i* кроме позиции *Ms[i][j+1]*; Если не нашли, то п.6, иначе п.8;
6. *s2=i*;
7. Если *s1<>s2,* тогда увеличить количество найденных линейных участков, сохранить значения границ *s1, s2* в массив линейных участков.
8. *i=i+1*;

В нашем примере существуют 2 условно-разделимых линейных участка с границами: {2;4}, {6;7}:

1. A = A+3;
   B = C+5;
   D = A+F;
2. M = M+1;
   E = E-2;

Далее для каждого полученного блока составляются частные матрицы следования, матрицы достижимости, матрицы входных и выходных переменных. После этого, составляется матрица неполного параллелизма и производится распараллеливание по методу, описанному в начале статьи с использованием формулы (1). После распараллеливания эти 2 блока примут следующий вид:

1. A = A+3; B = C+5;
   D = A+F;
2. M = M+1; E = E-2;

Операторы, представленные в первом столбце, выполняются на первом процессоре, операторы, представленные во втором столбце, выполняются на втором процессоре.

Для моделирования данного алгоритма поиска условно-разделимых блоков была разработана программа в среде Borland C++ Builder 6. В программу вводится матрица следования, на выходе получаем границы условно-разделимых блоков. Скриншот работающей программы приведен на рисунке 2.

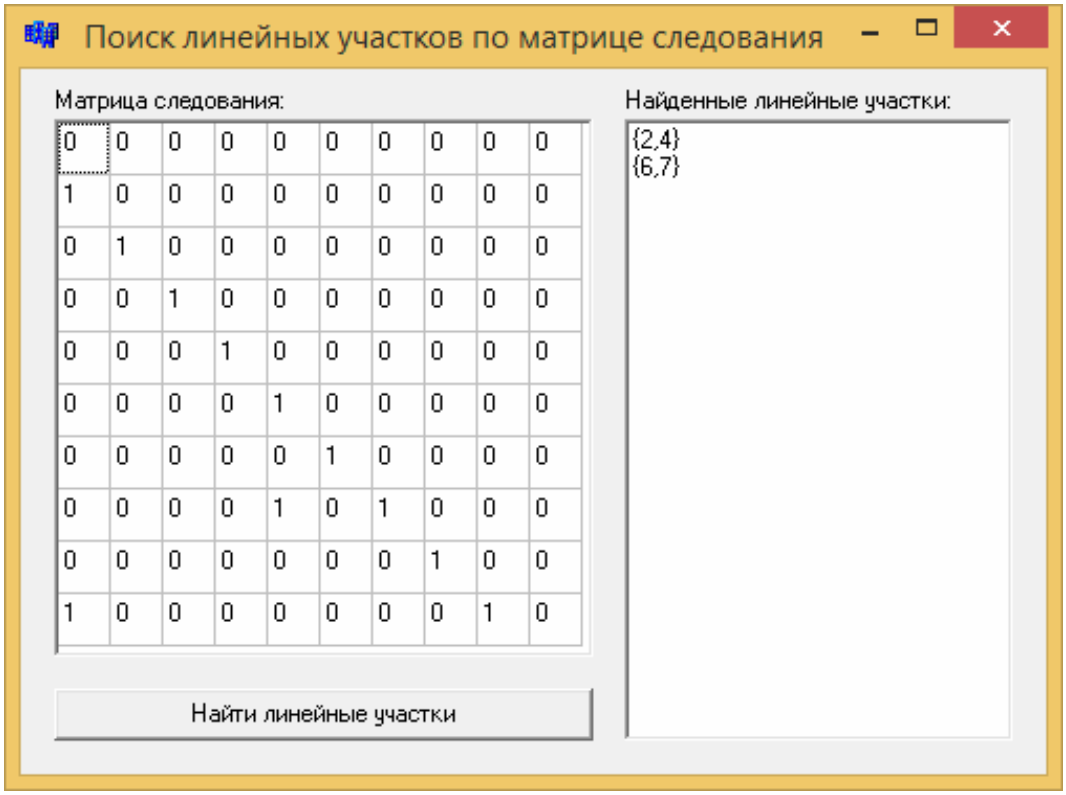

Рис. 2. Программа для моделирования предложенного алгоритма

Применение данной методики позволяет находить линейные участки внутри циклов на разных уровнях вложенности, причем наибольший эффект может быть достигнут при ее использовании применительно к телу циклов с большим числом итераций, и с большим числом операторов.

Ввиду сложности программной реализации данного алгоритма на хост-процессоре, для решения задачи загрузки процессоров в мультипроцессорной системе в дальнейшем планируется разработка специализированного вычислительного устройства для аппаратной реализации данного алгоритма.

Список литературы: **1.** Воеводин, В.В. Параллельные вычисления / Вл.В. Воеводин – СПб.: БХВ – Петербург, 2002.– 608 с. **2.** Цилькер, Б.Я. Организация ЭВМ и систем: учебник для вузов / Б.Я. Цилькер, С.А. Орлов. СПб.: Питер, 2004. 668 с. **3.** Ткачев, П.Ю. Проблема распараллеливания циклов с известным количеством итераций // «Инновация-2014»: Сборник материалов II регионального научно-технического семинара. Курск:ЮЗГУ,2014.