УДК 519.68
DOI: 10.15827/0236-235X.122.268-274
Дата подачи статьи: 01.02.18
2018. Т. 31. № 2. С. 268–274

МЕТОД ОПРЕДЕЛЕНИЯ ВОЗМОЖНОСТЕЙ ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ ФУНКЦИЙ АЛГОРИТМОВ АНАЛИЗА ДАННЫХ

И.И. Холод [1], *к.т.н., доцент, iiholod@mail.ru*

[1] *Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина), ул. Профессора Попова, 5, г. Санкт-Петербург, 197376, Россия*

В статье описывается метод определения возможности параллельного выполнения функций алгоритмов анализа данных.

Входными параметрами являются алгоритм анализа данных, представленный в виде композиции потокобезопасных функций, и модель знаний, представленная в виде массива деревьев унифицированных элементов, описывающих выявленные алгоритмом закономерности. При определении возможностей распараллеливания учитываются информационные зависимости между функциями, определяемые множеством используемых и множеством изменяемых элементов моделей знаний. Метод анализирует информационные связи для каждой пары функций алгоритма при проверке возможности распараллеливания по задачам, а также для вызовов функций в циклах на разных итерациях при проверке возможности распараллеливания по данным. В процессе анализа проверяются необходимые и достаточные условия параллельного выполнения, сформулированные для систем с общей и распределенной памятью. Они уточняют условия Бернштейна, используемые в теории компиляторов и являющиеся достаточными, но не необходимыми.

Метод определяет возможности параллельного выполнения функций для систем с общей памятью и для систем с распределенной памятью. При этом параллельное выполнение функций в системах с общей памятью более эффективно за счет отсутствия вызовов функции клонирования и объединения моделей знаний.

Результатом работы предлагаемого метода является параллельная форма исходного алгоритма анализа данных. Она содержит вставки специальных функций высшего порядка, обеспечивающих параллельное выполнение функций алгоритма, удовлетворяющих необходимым и достаточным условиям.

Для иллюстрации работы предложенного метода выполнено распараллеливание алгоритма классификации 1R. Определены функции алгоритма, которые могут быть распараллелены как по данным, так и по задачам. В качестве результата представлена параллельная форма алгоритма 1R с вставками функций распараллеливания для систем с общей памятью и для систем с распределенной памятью.

Ключевые слова*: параллельные алгоритмы, распределенные системы, data mining, зависимость по данным.*

Широкое внедрение информационных технологий приводит к появлению больших объемов хранящейся информации. Такая информация может размещаться на множестве распределенных узлов и в хранилищах (хранилищах данных [1], «облаках» [2] и т.п.), объединенных единой сетью. В связи с этим возрастает потребность в параллельной и распределенной обработке данных.

Одним из видов обработки данных является анализ. Алгоритмы анализа данных характеризуются следующими особенностями:

– неизменяемость анализируемых данных;

– создание моделей знаний малого объема при анализе больших объемов входных данных;

– извлечение из данных закономерностей сложными функциями, обрабатывающими несколько произвольных объектов из набора данных (не обладающими свойством списочного гомоморфизма);

– наличие ярко выраженной итерационной структуры с множеством циклов, обрабатывающих данные и элементы строящейся модели знаний.

Для параллельного и распределенного выполнения таких алгоритмов необходимы методы распараллеливания для систем как с распределенной, так и с общей памятью. Такие методы должны учитывать зависимости по данным между распараллеливаемыми функциями. В настоящее время известны условия Бернштейна для параллельного выполнения отдельных операций программ. Они являются достаточными, но не необходимыми.

В данной статье описывается метод определения возможности параллельного выполнения функций алгоритма анализа данных в системах с общей памятью и в распределенных системах.

Представление алгоритма анализа в виде композиции функций

Для определения возможностей параллельного выполнения алгоритм анализа данных должен быть представлен в виде композиции функций [3]:

$$dma = f_n \circ f_{n-1} \circ \ldots \circ f_i \circ \ldots \circ f_1 \circ f_0, \quad (1)$$

где функция $f_0 : D \to M$ принимает в качестве аргумента набор данных $d \in D$ (D – множество наборов данных) и возвращает созданную по нему модель знаний $m_0 \in M$ (M – множество моделей знаний); функции $f_i : M \to M$, $i = 1, ..., n$, принимают в качестве входного аргумента модель знаний $m_{i-1} \in M$, построенную предыдущей функцией f_{i-1}, и возвращают измененную модель знаний $m_i \in M$.

Набор данных содержит характеристики объектов (людей, элементов и др.), описываемых атрибутами (возраст, пол, вес и др.), и его часто представляют в виде 2-мерной матрицы для z объектов и их p атрибутов [4, 5]:

$$d = (x_{j.k})_{j=1,\,k=1}^{z,\,p}, \quad (2)$$

где $x_{j.k}$ – значение k-го атрибута для j-го объекта.

Набор возможных значений k-го атрибута обозначается как Def_k ($x_{j.k} \in Def_k$). Строка матрицы (2) является вектором.

Модель знаний $m \in M$ содержит закономерности, извлеченные из набора данных d алгоритмом анализа данных dma. Такие закономерности являются элементами модели знаний (классификационные и ассоциативные правила, центры кластеров, узлы деревьев решений и т.п.).

Существуют различные способы формального представления знаний: продукционные модели [6], семантические сети [7], фреймы [8] и др. Модель знаний, получаемая в результате выполнения алгоритмов анализа данных, может быть представлена в виде массива деревьев унифицированных элементов, для которых определены функции, позволяющие выполнять их параллельное построение [9]:

$$m : [E], \quad m = [e_1, e_2, \ldots, e_v, \ldots, e_w]. \quad (3)$$

Для индексации элементов модели знаний используем децимальную нотацию индексации элементов леса деревьев. Для хранения и обработки индексов в децимальной нотации – список целых чисел. Первое число индекса обозначает корневой элемент соответствующего дерева в модели знаний $m[v] \equiv e_v$, следующие числа в индексе – позиции элементов модели знаний среди узлов-потомков дерева. Например, элемент модели знаний $m[2, 1, 3]$ обозначает 3-й элемент модели знаний среди узлов-потомков 1-го узла среди узлов-потомков корневого узла 2-го дерева модели знаний m. Все множество индексов модели знаний обозначим символом I.

Каждый элемент модели знаний описывает некоторую закономерность с помощью набора параметров и содержит множество дочерних элементов: $e : E = <[P], [E]>$, $e = <[p_1, p_2, \ldots, p_d, \ldots, p_f], [e_1, e_2, \ldots, e_v, \ldots, e_w]>$.

При этом элементы модели знаний, являющиеся дочерними элементами одного и того же элемента, должны иметь одинаковый набор параметров.

Для представления цикла алгоритма анализа введена функция высшего порядка, описывающая итерационную обработку данных строящейся модели знаний в виде композиции функции, применяемой к разным элементам модели знаний:

$$loope : I \to I \to (M \to M) \to M \to M$$

$$loope\ i_s\ i_e\ f_t\ m = (f_t\ m[i_e]) \circ \ldots \circ (f_t\ m[i_s]),$$

где i_e и i_s – индексы первого и последнего элементов модели знаний, обрабатываемых в цикле.

Цикл, выполняющий итерационную обработку элементов набора данных d (2), также может быть представлен в виде композиции функции, применяемой к разным векторам набора данных:

$$loopd : I \to I \to (M \to M) \to D \to M,$$

$$loopd\ i_s\ i_e\ f_t\ d = loopd\ (f_t\ d[i_e]) \circ \ldots \circ (f_t\ d[i_s])\ f_t\ d,$$

где i_e и i_s – индексы первого и последнего векторов набора данных, обрабатываемых в цикле.

В процессе выполнения функция анализа данных f_i может использовать не все элементы модели знаний m_{i-1}, являющейся ее входным аргументом. Кроме того, результат ее работы может привести к изменению только некоторых элементов итоговой модели знаний m_i. Для анализа зависимостей по данным [10, 11] между функциями алгоритма анализа данных используем соответствующие обозначения:

– $In(f_i)$ – множество элементов входной модели знаний m_{i-1}, используемых функцией f_i в процессе выполнения;

– $Out(f_i)$ – множество элементов выходной модели знаний m_i, изменяемых функцией f_i в процессе выполнения.

Для распараллеливания последовательных алгоритмов и их выполнения в системах с общей и распределенной памятью введена функция высшего порядка *parallel* [3]:

$$parallel : Boolean \to [(M \to M)] \to M \to M$$

$$parallel\ s\ [f_1, \ldots, f_k]\ m = if\ (s == true)\ fork\ [f_1, \ldots, f_k]\ m$$
$$else\ join\ m\ (fork[f_1, \ldots, f_k]\ clone\ m).$$

Она использует функцию высшего порядка *fork*, которая принимает список функций и модель знаний, применяет каждую функцию из списка к модели знаний и полученную модель знаний добавляет в возвращаемый список:

$$fork : [M \to M] \to M \to [M]$$

$$fork\ [f_1, \ldots, f_k]\ m = [(run\ f_1\ clone\ m), \ldots, (run\ f_k\ clone\ m)] = [m_1, \ldots, m_k].$$

В функции *parallel* первый булевский аргумент s определяет способ выполнения: с общей или распределенной памятью.

Для систем с распределенной памятью, когда каждая параллельно выполняющаяся функция работает с аргументами, хранящимися в отдельной памяти, для создания отдельных копий модели знаний может использоваться функция клонирования *clone* [9]. Параллельно построенные модели знаний $m_1, \ldots, m_k$, также хранящиеся в отдельных областях памяти, должны объединяться в единую модель знаний m_r. Для этого может быть использована функция объединения *join* [9].

При параллельном выполнении в системах с общей памятью необходимо учитывать, что все изменения с моделью знаний, реализуемые параллельными функциями, осуществляются над одним и тем же экземпляром модели знаний, хранящейся в общей памяти. Это означает, что аргументами параллельных функций является один и тот же экземпляр модели знаний, что не требует клонирования модели знаний и ее объединения. За счет этого параллельное выполнение функций алгоритма анализа данных в системах с общей памятью более эффективно.

Частичное применение функции *parallel* с фиксацией первого аргумента позволяет получить

функция распараллеливания:

– для систем с общей памятью:

parallels : $[(M \to M)] \to M \to M$

parallels = *parallel true* $[f_r, \ldots, f_s]$ m = *fork*$[f_1, \ldots, f_k]$ m;

– для систем с распределенной памятью:

paralleld : $[(M \to M)] \to M \to M$

paralleld = *parallel false* $[f_r, \ldots, f_s]$ m = *join* m (*fork*$[f_1, \ldots, f_k]$ *clone* m).

Необходимые и достаточные условия параллельного выполнения функций алгоритмов анализа данных

Согласно условиям Бернстайна [12], достаточные условия для параллельного выполнения функций f_i и f_{i+1} следующие:

– отсутствие антизависимости (data anti-dependency): $In(f_i) \cap Out(f_{i+1}) = \varnothing$;

– отсутствие потоковой зависимости (data flow dependency): $Out(f_i) \cap In(f_{i+1}) = \varnothing$;

– отсутствие зависимости по выходу (output dependency): $Out(f_i) \cap Out(f_{i+1}) = \varnothing$.

Данные условия являются достаточными, но не необходимыми для параллельного выполнения функций f_i и f_{i+1}. Определим необходимые условия для параллельного выполнения пары функций алгоритма анализа данных для систем как с общей, так и с распределенной памятью.

При параллельном выполнении в системах с общей памятью функции f_i и f_{i+1} строят общую модель знаний и существует вероятность, что функция f_{i+1} будет выполнена до функции f_i: $f_{i+1} \circ f_i\, m_{i-1} = f_i \circ f_{i+1}\, m_{i-1}$.

Следовательно, для корректности параллельного выполнения функций f_i и f_{i+1} с общей моделью знаний они должны быть ассоциативными. Это возможно при следующих условиях:

– если функции f_i и f_{i+1} не используют и не изменяют общие элементы модели знаний (то есть выполняются условия Бернстайна);

– если функции f_i и f_{i+1} выполняют над общими для них элементами модели знаний только ассоциативные по отношению друг к другу операции o_i и o_{i+1} (то есть операции, которые могут выполняться в любом порядке):

$o_{i+1}(o_i(m_{i-1}[r])) = o_i(o_{i+1}(m_{i-1}[r]))$ для $m_{i-1}[r] \in (In(f_i) \cap Out(f_{i+1})) \cup (Out(f_i) \cap In(f_{i+1})) \cup (Out(f_i) \cap Out(f_{i+1}))$.

Ассоциативными являются операции, изменяющие свойства элементов модели знаний, например, суммирование, вычитание, инкрементация и т.п. Операции, выполняющие добавление или удаление элементов модели знаний, не являются ассоциативными.

При параллельном выполнении в системах с распределенной памятью функции f_i и f_{i+1} независимо строят разные экземпляры модели знаний (m_i и m_{i+1} соответственно), получая на вход модель знаний m_{i-1}, построенную предыдущей функцией. При этом в отличие от последовательного варианта на вход функции f_{i+1} будет поступать модель знаний m_{i-1}, построенная функцией f_{i-1}, а не функцией f_i. Итоговая модель знаний m_{i+1} должна строиться путем объединения моделей знаний, построенных функциями f_i и f_{i+1} в распределенной памяти функцией *join*: $f_{i+1} \circ f_i\, m_{i-1} = join(f_{i+1}\, m_{i-1})\, (f_i\, m_{i-1})$.

Таким образом, для корректности параллельного выполнения функций f_i и f_{i+1} с распределенной моделью знаний необходимо, чтобы независимое выполнение функций f_i и f_{i+1} не отличалось от последовательного выполнения, а функция *join* корректно объединяла модели знаний, построенные независимо.

Первое условие будет выполняться, если

– элементы модели знаний, используемые функцией f_{i+1}, не изменяются функцией f_i, то есть отсутствует потоковая зависимость $(Out(f_i) \cap \cap In(f_{i+1}) = \varnothing)$;

– для элементов, входящих в потоковую зависимость, операции o_i и o_{i+1}, выполняемые над такими элементами, были ассоциативными:

$o_{i+1}(o_i(m_{i-1}[r])) = o_i(o_{i+1}(m_{i-1}[r]))$ для $m_{i-1}[r] \in (Out(f_i) \cap In(f_{i+1}))$;

что в том числе позволяет объединять их в функции *join* после независимого выполнения функций f_i и f_{i+1}.

Второе условие выполняется, если

– функции f_i и f_{i+1} не изменяют общие элементы, то есть отсутствует зависимость по выходу, и в этом случае функция *join* объединяет множества элементов моделей знаний m_i и m_{i+1};

– для операций o_i и o_{i+1}, выполняемых над элементами моделей знаний, изменяемых функциями f_i и f_{i+1}, существует функция агрегирования $\oplus$ изменений, сделанных ими независимо в функциях f_i и f_{i+1}: $o_{i+1}(o_i(m_{i-1}[r]) = o_{i+1}(m_{i-1}[r]) \oplus o_i(m_{i-1}[r])$ для $m_{i-1}[r] \in (Out(f_i) \cap Out(f_{i+1}))$.

Наличие антизависимости между функциями f_i и f_{i+1}: $In(f_i) \cap Out(f_{i+1}) \neq \varnothing$ в системах с распределенной памятью невозможно, так как результаты параллельно выполненных функций физически разделены. Таким образом, оно не является необходимым.

Метод проверки условий параллельного выполнения функций алгоритма анализа данных

Верификация достаточных и необходимых условий для параллельного выполнения пары функций f_i и f_{i+1} включает:

1) проверку наличия потоковой зависимости $FD = Out(f_i) \cap In(f_{i+1})$ и при ее наличии проверку ассоциативности операций, выполняемых над элементами модели знаний, входящими в нее;

2) проверку наличия антизависимости $AD = In(f_i) \cap Out(f_{i+1})$ и при ее наличии проверку ассоциативности операций, выполняемых над элементами модели знаний, входящими в нее;

3) проверку наличия зависимости по выходу $OD = Out(f_i) \cap Out(f_{i+1})$ и при ее наличии проверку

а) ассоциативности операций, выполняемых над элементами модели знаний, входящими в нее;

б) существования операции агрегирования для элементов модели знаний, входящих в нее.

Порядок выполнения перечисленных проверок следующий:

1. выполняется проверка 1 и при ее выполнении
2. выполняется проверка 2

2.1. при ее выполнении

2.1.1. выполняется проверка 3а для проверки возможности параллельного выполнения с общей памятью функцией *parallels*;

3. независимо от результатов проверки 2 выполняется проверка 3б для проверки возможности параллельного выполнения с распределенной памятью функцией *paralleld*.

Представим псевдокод метода.

***verifyParallelize* (f_r, f_{r+1})** – функция проверки возможности параллельного выполнения функций алгоритма анализа данных

Вход: пара функций f_r и f_{r+1}, для которых выполняется распараллеливание

Выход: функция *parallel*, если параллельное выполнение возможно, и *null* иначе

verifyParallelize(f_i, f_{i+1})
$FD = Out(f_i) \cap In(f_{i+1})$; // множество элементов модели знаний, составляющих потоковую зависимость
$AD = In(f_i) \cap Out(f_{i+1})$; // множество элементов модели знаний, составляющих антизависимость
$OD = Out(f_i) \cap Out(f_{i+1})$; // множество элементов модели знаний, составляющих зависимость по выходу
if ($FD \neq \varnothing$) // проверка наличия потоковой зависимости
  for each e from FD // для каждого элемента модели знаний из потоковой зависимости
    if $o_{i+1}(o_i(e)) \neq o_i(o_{i+1}(e))$ // проверить ассоциативность операций, выполняемых над ним
      return null;
if ($AD \cup OD \neq \varnothing$) // проверка наличия антизависимости или зависимости по выходу
  for each e from AD $\cup$ *OD* // для каждого элемента из антизависимости или зависимости по выходу
    if $o_{i+1}(o_i(e)) \neq o_i(o_{i+1}(e))$ // проверить ассоциативность операций, выполняемых над ним
      return null
  return parallels;
if ($OD \neq \varnothing$) // проверка наличия зависимости по выходу
  for each e from OD // для каждого элемента модели знаний из зависимости по выходу
    if $\nexists \oplus$ // проверить наличие функции агрегирования
      return null
  return paralleld;

Если допустимо параллельное выполнение и с общей, и с распределенной памятью, должна использоваться функция *parallels* как наиболее эффективная (не использующая функции клонирования и объединения моделей знаний).

Проверка возможности параллельного выполнения функций выполняется в два этапа:

– проверка возможности распараллеливания по задачам – выполняется проверка необходимых и достаточных условий для каждой пары функций, находящихся на одном уровне композиции;

– проверка возможности распараллеливания по данным – выполняется проверка необходимых и достаточных условий для функций, вызывающихся итерационно в каждом цикле на разных итерациях.

Пример распараллеливания алгоритма анализа данных 1R

Рассмотрим распараллеливание с помощью описанного метода алгоритма классификации 1R [13]. Он формирует модель знаний m в виде простых правил классификации. Такие правила строятся по значению одного атрибута, поэтому в литературе алгоритм часто называют «1-правило» (1-rule). В соответствии с предложенным подходом модель знаний, представленная в виде леса деревьев (3), будет включать в себя следующие четыре дерева [3]:

- $m[0]$ – дерево метаданных набора данных d, где $m[0, k]$ – элемент модели знаний, содержащий информацию о k-м атрибуте a_k и его значениях.
- $m[1]$ – дерево классификационных правил, построенных алгоритмом 1R, в котором:

– $m[1].n$ – параметр, определяющий количество входящих в массив векторов, удовлетворяющих правилам (корректных векторов);

– $m[1, p]$ – классификационное правило, простейший вид которого может быть определен как набор параметров: $m[1, p] = <a_k, v_{k.q}, v_{t.p}, n>$, где a_k – атрибут $a_k \in A$, значение которого проверяется правилом; $v_{k.q}$ – значение атрибута a_k, с которым сравнивается значение вектора; $v_{t.p}$ – предсказываемое правилом значение зависимого атрибута a_t, $v_{t.p} \in Def(a_t)$; n – параметр, определяющий количество корректных векторов – векторов, для которых данное правило является верным: $n = |\{x_j : x_j \in X, (a_k(x_j) = v_{k.q} \text{ и } a_t(x_j) = v_{t.p})\}|$.

- $m[2]$ – дерево простых правил кандидатов.
- $m[3]$ – дерево, хранящее информацию о количестве векторов для каждой комбинации: значения независимого атрибута a_k, значения зависимого атрибута a_t:

– $m[3, k]$ содержит элементы, соответствующие значениям независимого атрибута a_k;

– $m[3, k, q]$ содержит элементы, соответствующие значению $v_{k.q}$ независимого атрибута a_k;

– $m[3, k, q].mi$ – индекс (mi: 1..u) значения зависимого атрибута a_t (класса), для которого имеется максимальное количество векторов (соответствует переменной mi);

– $m[3, k, q, p]$ – количество векторов, имеющих соответствующие значения независимого атрибута a_k и зависимого атрибута a_t: $m[3, k, q, p] = |\{x_j | a_k(x_j) = v_{k.q}, a_t(x_j) = v_{t.p}, x_j \in X, a_k, a_t \in A, v_{k.q} \in Def(a_k), v_{t.p} \in Def(a_t)\}|$.

Представим псевдокод реализации алгоритма 1R в библиотеке Weka.

```
// Для каждого атрибута
For k = 0 ... |A|
  // Для каждого атрибута подсчитать появление каждого класса
  For j = 1 ... |X|
      m[3, k, d[j, k], d[j, t]]++;
  For q = 1 ... |Def_k|
    // найти наиболее частый класс
    For p = 1 … |Def_t|
      if m[3, k, m[0, k, q], m[0, t, p]] > m[3, k, m[0, k, q],
                                            m[3, k, m[3, k, q]].mi]
        m[3, k, q].mi = p;
    // сделать правило для класса и значения атрибута
    m[2].add(<m[0, k], m[0, k, q], m[3, k, m[0, k, q]],
             m[3, k, m[0, k, q], m[3, k, m[3, k, q]].mi]})
    // вычислить ошибку класса
    m[2].n+= m[3, k, m[0, k, q], m[3, k, m[0, k, q]].mi]
  //Выбрать правило с минимальной ошибкой
  if(m[2].n > m[1].n)
    m[1].n = m[2].n;
    m[1] = m[2];
  m[2].cleare;
```

Сам алгоритм 1R в виде композиции функций (1) имеет вид [3]:

$$1R = f_1 \circ f_0 = \quad (4)$$
$$= (loope\ [0, 1]\ [0, p]\ f_{11} \circ f_{10} \circ f_4 \circ f_2) \circ f_0 =$$
$$= (loope\ [0, 1]\ [0, p]\ f_{11} \circ f_{10} \circ (loope\ [0, k, 1]\ [0, k, u]\ f_9 \circ f_8 \circ f_5) \circ (loopd\ [0]\ [z]\ f_3)) \circ f_0 =$$
$$= (loope\ [0, 1]\ [0, p]\ f_{11} \circ f_{10} \circ (loope\ [0, k, 1]\ [0, k, u]\ f_9 \circ f_8 \circ (loope\ [0, t, 1]\ [0, t, y]\ f_6)) \circ (loopd\ [0]\ [z]\ f_3)) \circ f_0,$$

где f_1 – цикл по метаданным с $m[0, 1]$ по $m[0, p]$: $f_1 = loope\ [0, 1]\ [0, p]\ (f_{13} \circ f_{10} \circ f_4 \circ f_2)$;

f_2 – цикл по векторам набора данных с $d[0]$ по $d[z]$: $f_2 = loopd\ [0]\ [z]\ f_3$;

f_3 – функция, выполняющая инкрементацию количества корректных векторов $m[3, k, q, p]$ для значений $v_{k.q}$ независимого атрибута a_k и значений $v_{t.p}$ зависимого атрибута a_t;

f_4 – цикл по значениям независимого атрибута a_k с $m[0, k, 1]$ по $m[0, k, u]$: $f_4 = loope\ [0, k, 1]\ [0, k, u]\ (f_9 \circ f_8 \circ f_5)$;

f_5 – цикл по значениям зависимого атрибута a_t с $m[0, t, 1]$ по $m[0, t, y]$: $f_5 = loope\ [0, t, 1]\ [0, t, y]\ f_6$;

f_6 – функция, выполняющая поиск элемента модели знаний с максимальным значением количества корректных векторов $m[3, k, q, p]$ среди элементов поддерева $m[3, k, q]$ и запоминающая его индекс в поле $m[3, k, q].mi$;

f_8 – функция, выполняющая добавление правила $m[2, p]$ с максимальным количеством корректных векторов во множество правил кандидатов $m[2]$;

f_9 – функция, выполняющая подсчет корректных векторов $m[2, p].n$ для одного атрибута a_k;

f_{10} – функция, выполняющая выбор лучшего набора правил ($m[1]$ или $m[2]$) для одного атрибута;

f_{11} – функция, выполняющая удаление набора правил кандидатов из $m[2]$ для одного атрибута.

Множества используемых и изменяемых элементов функций алгоритма 1R, определенные на основании псевдокода, представлены в таблице.

Для алгоритма 1R возможности распараллеливания по задачам для каждого уровня композиции будут следующими:

- функции f_0 и f_1 не могут выполняться параллельно, так как имеют потоковую зависимость по элементу модели знаний $m[0]$: $Out(f_0) \cap In(f_1) = \{m[0]\} \neq \varnothing$, над которым в них выполняются не ассоциативные по отношению друг другу операции добавления и извлечения;
- функции f_2 и f_4 не могут выполняться параллельно, так как имеют потоковую зависимость по элементу модели знаний $m[3, k]$: $Out(f_2) \cap In(f_4) = \{m[3, k]\} \neq \varnothing$, над которым в них выполняются не ассоциативные по отношению друг другу операции инкрементации и сравнения;
- функции f_4 и f_{10} не могут выполняться параллельно, так как имеют потоковую зависимость по элементу модели знаний $m[2]$: $Out(f_4) \cap In(f_{10}) = \{m[2]\} \neq \varnothing$, над которым в них выполняются не ассоциативные по отношению друг другу операции добавления и извлечения;

Множества используемых и изменяемых элементов модели знаний функциями алгоритма 1R

Sets of used and modifiable elements of a mining model by 1R algorithm functions

f_i	$In(f_i)$	$Out(f_i)$
f_0	d	m[0]
f_1	d, m[0], m[1], m[2], m[3]	m[1], m[2], m[3]
f_2	d[0,k], ..., d[z,k]; d[0,t], ..., d[z,t]; m[3,k]	m[3,k]
f_3	d[j, k], d[j, t], m[3,k,q,p]	m[3,k,q,p]
f_4	m[0,k], m[0,t], m[3,k], m[2].n	m[3,k,1].mi,…,m[3,k,u].mi, m[2], m[2].n
f_5	m[0,k,q], m[0,t], m[3,k,q’], m[3,k,q’,p”]	m[3,k,q’].mi
f_6	m[0,k,q], m[0,t,p], m[3,k,q’,p’], m[3,k,q’,p”]	m[3,k,q’].mi
f_8	m[0, k], m[0,k,q], m[3,k,q’], m[3,k,q’].mi, m[3,k,q’,p”]	m[2]
f_9	m[0,k,q], m[3, k, q’].mi, m[3, k, q’,p”], m[2].n	m[2].n
f_{10}	m[1].n, m[2].n, m[1], m[2]	m[1].n, m[1]
f_{11}	m[2]	m[2]

• функции f_{10} и f_{11} могут выполняться параллельно только с распределенной памятью, так как

– не имеют потоковой зависимости $Out(f_{10}) \cap \cap In(f_{11}) = \varnothing$ и зависимости по выходу $Out(f_{10}) \cap \cap Out(f_{11}) \neq \varnothing$;

– имеют антизависимость по элементу модели знаний $m[2]$: $In(f_{10}) \cap Out(f_{11}) = \{m[2]\}$, над которым в них выполняются не ассоциативные по отношению друг другу операции удаления и извлечения;

• функции f_5 и f_8 не могут выполняться параллельно, так как имеют потоковую зависимость по элементу модели знаний $m[3, k, q'].mi$: $Out(f_5) \cap \cap In(f_8) = \{m[3, k, q'].mi\} \neq \varnothing$, над которым в них выполняются не ассоциативные по отношению друг другу операции поиска (присваивания индекса максимального элемента) и чтения;

• функции f_8 и f_9 могут выполняться параллельно, так как

– не имеют потоковой зависимости: $Out(f_8) \cap \cap In(f_9) = \varnothing$;

– не имеют антизависимости: $In(f_8) \cap Out(f_9) = \varnothing$;

– не имеют зависимости по выходу: $Out(f_8) \cap \cap Out(f_9) = \varnothing$.

Возможности распараллеливания по данным для каждого цикла будут следующими:

• композиция функций $f_{11}{\circ}f_{10}{\circ}f_4{\circ}f_2$ для разных итераций k и $k + 1$ не может быть распараллелена по данным, так как имеется потоковая зависимость по элементам модели знаний $m[1]$ и $m[2]$: Out ($f_{11}{\circ}f_{10}{\circ}f_4{\circ}f_2$ для k) $\cap$ In ($f_{11}{\circ}f_{10}{\circ}f_4{\circ}f_2$ для $k + 1$) = = $\{m[1], m[2], m[3, k]\} \cap \{d, m[0, k + 1], m[1], m[2], m[3, k + 1]\} = \{m[1], m[2]\} \neq \varnothing$, над которыми в ней выполняются не ассоциативные по отношению друг другу операции добавления, удаления и извлечения;

• композиция функций $f_9{\circ}f_8{\circ}f_5$ для разных итераций q и $q + 1$ может быть распараллелена по данным, так как

– существуют потоковая зависимость и антизависимость по элементу $m[2].n$: Out ($f_9{\circ}f_8{\circ}f_5$ для q) $\cap$ In ($f_9{\circ}f_8{\circ}f_5$ для $q + 1$) = $\{m[3, k, q'].mi, m[2], m[2].n\} \cap \{m[0, k, q + 1], m[0, t], m[3, k, q + 1], m[2].n\} = \{m[2].n\} \neq \varnothing$;

– In ($f_9{\circ}f_8{\circ}f_5$ для q) $\cap$ Out ($f_9{\circ}f_8{\circ}f_5$ для $q + 1$) = = $\{m[0, k, q], m[0, t], m[3, k, q], m[2].n\} \cap \{m[3, k, q'].mi, m[2], m[2].n\} = \{m[2].n\} \neq \varnothing$, над которым выполняется ассоциативная операция инкрементации;

– имеется зависимость по выходу по элементам модели знаний $m[2]$ и $m[2].n$: Out ($f_9{\circ}f_8{\circ}f_5$ для q) $\cap$ Out ($f_9{\circ}f_8{\circ}f_5$ для $q + 1$) = $\{m[3, k, q'].mi, m[2], m[2].n\} \cap \{m[3, k, q'].mi, m[2], m[2].n\} = \{m[2], m[2].n\} \neq \varnothing$, над которым выполняется ассоциативная операция инкрементации и для которого существует операция агрегирования – суммирование;

• функция f_6 для разных итераций p и $p + 1$ не может быть распараллелена по данным, так как

– нет потоковой зависимости: Out (f_6 для p) $\cap$ $\cap$ In (f_6 для $p + 1$) = $\{m[3, k, q'].mi\} \cap \{m[0, k, q], m[0, t, p + 1], m[3, k, q', p'], m[3, k, q', p'']\} = \varnothing$;

– нет антизависимости: In(f_6 для p) $\cap$ Out(f_6 для $p + 1$) = $\{m[0, k, q], m[0, t, p], m[3, k, q', p'], m[3, k, q', p'']\} \cap \{m[3, k, q'].mi\} = \varnothing$;

– но имеется зависимость по выходу по элементу $m[3, k, q'].mi$: Out(f_6 для p) $\cap$ Out(f_6 для $p + 1$) = $\{m[3, k, q'].mi\} \cap \{m[3, k, q'].mi\} \neq \varnothing$, над которым выполняется неассоциативная операция поиска (присваивания индекса максимального элемента), а также отсутствует функция агрегирования;

• функция f_3 для разных итераций j и j+1 может быть распараллелена по данным, так как

– нет потоковой зависимости: Out (f_3 для j) $\cap$ $\cap$ In (f_3 для $j + 1$) = $\{m[3, k, q, p]\} \cap \{d[j + 1, k], d[j + 1, t]\} = \varnothing$;

– нет антизависимости: In (f_3 для j) $\cap$ Out (f_3 для $j + 1$) = $\{d[j, k], d[j, t]\} \cap \{m[3, k, q, p]\} = \varnothing$;

– имеется зависимость по выходу по элементу модели знаний $m[3, k, q, p]$: Out (f_3 для j) $\cap$ $\cap$ Out (f_3 для $j + 1$) = $\{m[3, k, q, p]\} \cap \{m[3, k, q, p]\} = \{m[3, k, q, p]\} \neq \varnothing$, над которым выполняется ассоциативная операция инкрементации и для которого существует операция агрегирования – суммирование.

Таким образом, распараллеливание алгоритма 1R, представленного выражением (4), может быть выполнено следующим образом:

$$1RPar = (loope\ [0, 1]\ [0, p]\ \underline{\textbf{paralleld}}\ [f_{10}, f_{11}]^{\circ} (\underline{\textbf{paralleld}}\ [loope\ [0, k, 1]\ [0, k, u] (\underline{\textbf{parallels}}\ [f_8\ f_9]) \circ (loope\ [0, t, 1]\ [0, t, y]\ f_6)]) {}^{\circ}(\underline{\textbf{parallels}}\ [loopd\ [0]\ [z]\ f_3)])^{\circ} f_0.$$

Программная реализация и эксперименты, подтверждающие эффективность такого распараллеливания, описаны в статье [3].

Заключение

В данной работе рассмотрены достаточные и необходимые условия параллельного выполнения функций алгоритмов анализа данных, учитывающие множества используемых и изменяемых элементов моделей знаний.

Для систем с общей памятью достаточными и необходимыми условиями являются:

– наличие свойства ассоциативности у операций, выполняемых над элементами модели знаний, входящих в потоковую зависимость, или отсутствие таких элементов;

– наличие свойства ассоциативности у операций, выполняемых над элементами модели знаний, входящих в антизависимость, или отсутствие таких элементов;

– наличие свойства ассоциативности у операций, выполняемых над элементами модели знаний,

входящих в зависимость по выходу, или отсутствие таких элементов.

Для систем с распределенной памятью достаточными и необходимыми условиями являются:

– наличие свойства ассоциативности у операций, выполняемых над элементами модели знаний, входящих в потоковую зависимость, или отсутствие таких элементов;

– существование операции агрегирования для элементов модели знаний, входящих в зависимость по выходу, или отсутствие таких элементов.

Для выполнения проверки возможностей параллельного выполнения функций алгоритмов анализа данных предложен метод, который осуществляет проверки в заданном порядке и определяет наиболее эффективную функцию для распараллеливания как по задачам, так и по данным для систем с общей и распределенной памятью.

Литература

1. Inmon W.H. Building the data warehouse. Wiley Publ., 2005, 523 p.

2. Antonopoulos N., Gillam Lee. Cloud computing: principles, systems and applications. Springer, 2010, 379 p.

3. Kholod I., Shorov A., and Gorlatch S. A Functional Approach to Parallelizing Data Mining Algorithms in Java. Springer, LNCS, 2017, vol. 10421, pp. 459–472. DOI: 10.1007/978-3-319-62932-2_44.

4. Hastie T., Tibshirani R., Friedman J. The elements of statistical learning: data mining, inference and prediction, Springer, NY, 2001, 533 p.

5. Han J., Kamber M. Data mining: concepts and techniques. SF, Morgan Kaufman Publ., 2001, 550 p.

6. Roberts D. The Existential Graphs of Charles S. Peirce. France, The Hague Mouton Publ., 1973, 168 p.

7. Lehmann F., Rodin E.Y. (eds.) Semantic networks in artificial intelligence. NY, Pergamon Press, 1992, vol. 24, 758 p.

8. Минский М. Фреймы для представления знаний; [пер. с англ.]. М.: Энергия, 1979. 151 с.

9. Kholod I.I., Shorov A.V. Unification of mining model for parallel processing. Proc. EIConRus, 2017, pp. 450–455.

10. Banerjee U. Data dependence in ordinary programs. Urbana, 1976, Technical Report, no. 76-837.

11. Banerjee U. An introduction to a formal theory of dependence analysis. The J. of Supercomputing. 1988, vol. 2, pp. 133–149.

12. Bernstein A.J. Program Analysis for Parallel Processing. IEEE Trans. on Electronic Computers, 1966, vol. 15, no. 5, pp. 757–762.

13. Holte R.C. Very simple classification rules perform well on most commonly used datasets. Machine Learning, 1993, vol. 11, pp. 63–90.

Software & Systems
DOI: 10.15827/0236-235X.122.268-274

Received 01.02.18
2018, vol. 31, no. 2, pp. 268–274

A METHOD FOR DETERMINING THE CAPABILITIES OF PARALLEL EXECUTION OF DATA MINING ALGORITHM FUNCTIONS

I.I. Kholod [1], *Ph.D. (Engineering), Associate Professor, iiholod@mail.ru*

[1] *St. Petersburg Electrotechnical University "LETI", Prof. Popov St. 5, St. Petersburg, 197376, Russian Federation*

Abstract. The article describes the method for determining the possibility of parallel execution of data mining algorithm functions.

The input parameters of the method are: a data mining algorithm represented as a composition of thread-safe functions, and a mining model represented as an array of unified element trees describing the patterns mined by the algorithm. When determining the possibilities of parallelization, the method takes into account information dependencies between functions, which are determined by a set of used and modifiable elements of a mining model. The method analyzes data connections for each pair of algorithm functions when testing the possibility of task parallelization, as well as for calling functions in loops for different iterations, while verifying the possibility of data parallelization. The analysis includes checking necessary and sufficient conditions for parallel execution for systems with shared and distributed memory. They extend the Bernstein conditions used in compiler theory and are sufficient, but not necessary.

The method determines the possibilities of parallel execution of functions for shared memory systems and distributed memory systems. In this case, parallel performance of functions in shared memory systems is more efficient due to the lack of calls for the cloning function and combining of mining models.

The result of the proposed method is the parallel form of the source data analysis algorithm. It contains insertions of special functions of higher order that provide parallel execution of algorithm functions that satisfy necessary and sufficient conditions.

To illustrate the proposed method, the paper shows parallelizing 1R classification algorithm. It defines algorithm functions that can be paralleled, both by data and by tasks. The result is a parallel form of the 1R algorithm with parallelization function inserts for shared memory systems and distributed memory systems.

Keywords: parallel algorithms, distributed systems, data mining, data dependency.

References

1. Inmon W.H. *Building the Data Warehouse*. 4nd ed., Canada, Indianapolis, Indiana, Wiley Publ., 2005, 523 p.

2. Antonopoulos N., Lee G. *Cloud Computing: Principles, Systems and Applications*. Springer Publ., 2010, 379 p.

3. Kholod I., Shorov A., Gorlatch S. A Functional Approach to Parallelizing Data Mining Algorithms in Java. *PaCT 2017, LNCS*. V. Malyshkin (Ed.), Springer Int. Publ., 2017, vol. 10421, pp. 459–472.

4. Hastie T., Tibshirani R., Friedman J. *The elements of statistical learning: data mining, inference and prediction*. NY, Springer Publ., 2001, 533 p.

5. Han J., Kamber M. *Data Mining: Concepts and Techniques*. Morgan Kaufman Publ., San Francisco, 2001.

6. Roberts D. *The Existential Graphs of Charles S. Peirce*. Mouton, The Hague Publ., 1973.

7. *Semantic networks in artificial intelligence*. Lehmann F., Rodin E.Y. (Eds.). Oxford, NY, Pergamon Press. 1992, 758 p.

8. Minsky M. *A Framework for Representing Knowledge*. MIT-AI Laboratory Memo 306 Publ., 1974 (Russ.ed.: Moscow, Energiya Publ., 1979).

9. Kholod I.I., Shorov A.V. Unification of mining model for parallel processing. *Proc. 2017 IEEE North West Russia Section Young Researchers in Electrical and Electronic Engineering Conf. (2017 ElConRusW)*. IEEE Xplore 2017, pp. 450–455.

10. Banerjee U. *Data dependence in ordinary programs*. Urbana, 1976, no. 76–837.

11. Banerjee U. An introduction to a formal theory of dependence analysis. *The J. of Supercomputing*. 1988, vol. 2, pp. 133–149.

12. Bernstein A.J. Program analysis for parallel processing. *IEEE Trans. on Electronic Computers*. 1966, vol. 15, no. 5, pp. 757–762.

13. Holte R.C. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*. 1993, vol. 11, pp. 63–90.