РОССИЙСКАЯ АКАДЕМИЯ НАУК

СИБИРСКОЕ ОТДЕЛЕНИЕ

А В Т О М Е Т Р И Я

2003, том 39, № 3

УДК: 519.688 → 519.172

Т. В. Борец

(Новосибирск)

АССОЦИАТИВНАЯ ВЕРСИЯ АЛГОРИТМА ЛЕНГАУЭРА – ТАРЬЯНА ДЛЯ ВЫЧИСЛЕНИЯ НЕПОСРЕДСТВЕННЫХ ДОМИНАТОРОВ В ГРАФЕ *

Представлена ассоциативная версия алгоритма Ленгауэра – Тарьяна на модели параллельного процессора с вертикальной обработкой данных. Алгоритм реализован в виде процедуры на языке STAR. Приведена оценка временной сложности и обоснование корректности этой процедуры.

Введение. Вычисление непосредственных доминаторов в управляющем графе (уграфе) – хорошо известная проблема, возникающая при решении задач потокового анализа и оптимизации программ. Решение этой проблемы описано Кнутом [1], Ахо и Ульманом [2], Тарьяном и Ленгауэром [3], Евстигнеевым и Касьяновым [4]. Для ее решения были предложены три основных алгоритма: явный алгоритм [2] с временной сложностью $O(mn)$, где m – число дуг и n – число вершин уграфа; алгоритм, работающий с битовыми векторами [5]; быстрый алгоритм Ленгауэра – Тарьяна [3] с временной сложностью $O(m\alpha(m,n))$, где α – функция, обратная функции Аккермана. Для этих алгоритмов предлагаются новые реализации и структуры данных, позволяющие уменьшить их время работы.

В данной работе представлена реализация алгоритма Ленгауэра – Тарьяна на модели ассоциативного параллельного процессора типа SIMD (Single Instruction Multiple Data). Такая архитектура была выбрана ввиду того, что она ориентирована на решение нечисловых задач. Интерес к ней растет, поскольку она поддерживает массовый параллельный поиск по содержимому ячеек и обработку неупорядоченных данных, представленных в виде двумерных таблиц [6]. Мы показали, что ассоциативная версия алгоритма Ленгауэра – Тарьяна выполняется на STAR-машине за время $O(m\log n)$.

1. **Модель STAR-машины.** STAR-машина состоит из последовательного устройства управления, в котором записаны программа и скалярные константы, матричной памяти, устройства ассоциативной обработки, состояще-

* Работа выполнена при поддержке Российского фонда фундаментальных исследований (грант № 03-01-00399).

го из p одноразрядных процессорных элементов ($p = 2^l$). Полное описание STAR-машины дано в [7].

Чтобы моделировать обработку информации в матричной памяти, STAR-машина использует новые типы данных: *slice*, *word* и *table*. Переменная типа *slice* (далее слайс) моделирует доступ к таблице по столбцам, а переменная типа *word* – доступ по строкам. С каждой переменной типа *table* ассоциируется бинарная матрица из n строк и k столбцов, где $n \le p$.

Пусть X и Y – слайсы, i – целочисленная переменная. Приведем необходимые операции и предикаты для слайсов.

SET(Y) записывает в слайс Y все единицы; CLR(Y) записывает в слайс Y все нули; $Y(i)$ выделяет i-ю компоненту в слайсе $Y (1 \le i \le p)$; FND(Y) выдает порядковый номер i позиции первой (или старшей) единицы в слайсе Y, где $i \ge 0$; STEP(Y) выдает такой же результат, как и операция FND(Y), и затем обнуляет старшую единицу. Общепринятым способом вводятся предикаты ZERO(Y) и SOME(Y), а также следующие побитовые логические операции: X and Y, X or Y, not X, X xor Y.

Для переменных типа *word* выполняются те же операции, что и для переменных типа *slice*. Для переменных типа *table* определены следующие операции: COL(i, T) и ROW(i, T), которые выделяют из матрицы T i-й столбец и i-ю строку соответственно.

Следуя Фостеру [8], допускаем, что каждая элементарная операция STAR-машины выполняется за единицу времени. Поэтому сложность алгоритма определяется числом элементарных операций, которые выполняются в наихудшем случае.

Для реализации алгоритма нам потребуются следующие базовые процедуры из [9, 10].

Процедура MATCH(T: table; X: slice; v: word; var Z: slice) определяет позиции тех строк матрицы T, которые совпадают со словом w. Процедура возвращает слайс Z, в котором $Z(i) =$ '1' тогда и только тогда, когда ROW(i, T) $= w$ и $X(i) =$ '1'.

Процедура MIN(T: table; X: slice; var Y: slice) определяет позиции тех строк матрицы T, в которых записано минимальное число. Она возвращает слайс Y, в котором $Y(i) =$ '1' тогда и только тогда, когда ROW(i, T) – минимальный элемент и $X(i) =$ '1'.

Процедура GREAT(T: table; X: slice; w: word; var Y: slice) определяет позиции тех строк матрицы T, которые больше, чем слово w. Процедура возвращает слайс Y, в котором $Y(i) =$ '1' тогда и только тогда, когда ROW(i, T) $> w$ и $X(i) =$ '1'.

Процедура TCOPY(T: table; n: integer; var R: table) копирует n столбцов матрицы T в матрицу R.

Процедура WMERGE(w: word; X: slice; var T: table) копирует слово w в те строки матрицы T, которые помечены '1' в слайсе X.

Как показано в [9, 10], каждая из базовых процедур выполняется на STAR-машине за время, пропорциональное числу битовых столбцов.

2. Алгоритм Ленгауэра – Тарьяна. Приведем понятия, которые будем использовать для описания алгоритма Ленгауэра – Тарьяна. Ориентированным графом (орграфом) G называется пара множеств (V, E), где $V = \{1, \ldots, n\}$ – множество вершин; E – множество дуг (ориентированных ребер), $E \subseteq V \times V$, $|E| = m$.

Путем p из вершины v в вершину w называется последовательность вершин $v = v_0, v_1, \ldots, v_k = w$, такая, что $(v_i, v_{i+1}) \in E$ для $0 \le i < k$.

Графом $F = (G, r)$ называется такой орграф $G = (V, E)$ с выделенной начальной вершиной r, что для любой вершины $v \in V$ существует путь из вершины r в вершину v. Тогда *корневое дерево* $T = (V, E, r)$ можно определить как граф, у которого $|E| = |V| - 1$, вершина r называется *корневой вершиной* или *корнем*. Если v и w – вершины в дереве T и существует путь из v в w, то вершина v называется предшественником вершины w ($v \xrightarrow{*} w$). Если при этом $v \neq w$, то $v \xrightarrow{+} w$.

Приведем определение поиска в глубину [11]. Поиск в глубину – метод систематического посещения вершин графа. Он использует следующую стратегию: идти в глубину, пока это возможно (т. е. имеются непройденные исходящие ребра), и искать другой путь, когда таких ребер нет. Так делается, пока не обнаружены все вершины, достижимые из исходной. Поиск в глубину нумерует вершины от 1 до n в том порядке, в котором они посещаются в первый раз. Номер, полученный вершиной в процессе поиска в глубину, называется M-номером. Будем считать, что $v < w$, если M-номер вершины v меньше, чем M-номер вершины w. В процессе поиска в глубину вычисляется остовное дерево, называемое деревом поиска в глубину или DFS-деревом.

Будем говорить, что вершина v доминирует над вершиной w (v – *доминатор* w) тогда и только тогда, когда в графе F любой путь из r в w проходит через вершину v. Вершина v называется непосредственным доминатором вершины w (обозначается как $idom(w)$), если любой другой доминатор u вершины w доминирует над вершиной v.

Вводим также понятие полудоминаторов, как они были определены Тарьяном.

Полудоминатор вершины v – это ее предшественник, определенный как

$$sdom(v) = \min\{u \mid \exists \text{ путь } u = w_0, w_1, \ldots, w_k, w_{k+1} = v, \text{где } w_i > v \text{ для } i = 1, \ldots, k\}.$$

Приведем те свойства доминаторов и полудоминаторов, которые понадобятся при описании алгоритма Ленгауэра – Тарьяна (обоснование этих свойств можно найти в [3]).

Свойство 1. Для любой вершины $v \neq r$ $idom(v) \xrightarrow{*} sdom(v)$.

Свойство 2. Для любой вершины $v \neq r$ $sdom(v) \xrightarrow{+} v$.

Свойство 3. Пусть вершина $v \neq r$, и пусть u – такая вершина, что

$$sdom(u) = \min\{sdom(w) \mid sdom(v) \xrightarrow{+} w \xrightarrow{*} v\}.$$

Тогда

$$idom(v) = \begin{cases} sdom(v), & \text{если } sdom(v) = sdom(u), \\ idom(u) & \text{иначе.} \end{cases}$$

Свойство 4. Для любой вершины $v \neq r$ $sdom(v) = \min(S_1 \cup S_2)$, где $S_1 = \{w \mid (w, v) \in E$ и $w < v\}$ и $S_2 = \{sdom(u) \mid u > v$ и $\exists (w, v)$: $u \xrightarrow{*} w\}$.

Последние два свойства дают алгоритм вычисления полудоминаторов и непосредственных доминаторов.

Теперь опишем сам алгоритм Ленгауэра – Тарьяна, который состоит из трех частей. На первом шаге выполняется поиск в глубину, который строит DFS-дерево, начиная с корневой вершины *r*. В порядке обхода графа каждой вершине приписывается *M*-номер. Далее, как промежуточный шаг для поиска доминаторов для каждой вершины $v \in V \setminus \{r\}$ вычисляется полудоминатор *sdom*(v). Вычисление полудоминаторов выполняется в порядке убывания *M*-номеров вершин. На этом шаге можно неявно вычислить значения непосредственных доминаторов для некоторых вершин. На последнем шаге согласно свойству 3 явно вычисляются значения непосредственных доминаторов.

Ленгауэр и Тарьян построили реализацию этого алгоритма в виде процедур, написанных на алголоподобном языке. Уграф задается списком смежности, когда для каждой вершины v хранится множество таких вершин *w*, что (v, *w*) – ребро уграфа. Дерево поиска в глубину хранится как массив *parent*, в котором *parent*(v) – предок вершины v в дереве. Для построения DFS-дерева используется известная рекурсивная процедура. На этом шаге также происходит инициирование переменных, используемых на последующих шагах.

Для поиска полудоминаторов используется динамический лес, который вначале состоит из всех вершин уграфа как вырожденных деревьев. Производится обход DFS-дерева *T* в порядке убывания *M*-номеров, пока сохраняется динамический лес *Fr*, который является подграфом DFS-дерева *T*. Поэтому дуга $(v, w) \in T$, включенная в *Fr*, связывает в *Fr* родителя v с сыном *w*.

Над лесом *Fr* выполняются следующие операции.

1. LINK(v, *w*) добавляет дугу $(v, w) \in T$ в *Fr*. Вершины v и *w* – корневые вершины деревьев в *F* (это получается исключительно в порядке обхода).

2. EVAL(v) возвращает вершину v, если v – корень дерева, и любую вершину *u* с минимальным полудоминатором, принадлежащую пути из корня в v, если v – не корень.

Чтобы вычислить значение полудоминатора для вершины v, нужно найти минимум значений *sdom*(EVAL(*w*)) по множеству вершин *w*, из которых выходят дуги в вершину v. Вершина, на которой достигается минимум, запоминается как *dom*(*w*), и дуга (*parent*(v), v) добавляется в лес *Fr*. На последнем этапе по свойству 4 корректируются значения доминаторов для тех вершин v, у которых *M*-номер вершины *dom*(v) не равен *sdom*(v).

Тарьян и Ленгауэр показали, что полудоминаторы и непосредственные доминаторы можно найти за время $O(m\alpha(m, n))$.

3. **Представление алгоритма Ленгауэра – Тарьяна на ассоциативном параллельном процессоре.** Чтобы реализовать алгоритм Ленгауэра – Тарьяна на ассоциативном параллельном процессоре, как и в последовательной реализации, выделяем три шага:

– построение дерева поиска в глубину;

– вычисление полудоминаторов;

– вычисление непосредственных доминаторов.

Каждый шаг реализован в виде процедуры на языке STAR. Далее опишем используемые структуры данных и реализацию каждого шага.

3.1. *Структуры данных.* Граф (V, E, r) представим в следующем виде:

1) пара матриц *left* и *right* задает множество дуг E, когда дуге (u, v) соответствуют строки, содержащие бинарную запись номеров вершин u и v;

2) *root* – переменная типа *word* хранит бинарный код начальной вершины r;

3) матрица *code* хранит двоичные коды вершин графа;

4) n – переменная типа *integer* хранит число вершин графа;

5) матрицы *left*1 и *right*1 используются для хранения множества дуг графа в M-нумерации;

6) матрица NV хранит M-номера всех вершин;

7) матрица R – матрица путей размера $m \times n$, в i-м столбце которой отмечаются '1' позиции только тех дуг, которые принадлежат пути по DFS-дереву от вершины *root* до вершины с M-номером, равным i. Таким образом, R однозначно определяет DFS-дерево;

8) матрица *idom* используется для хранения значений непосредственных доминаторов вершин.

Матрицы *left*1, *right*1, NV и R получаются в результате построения DFS-дерева.

3.2. *Построение DFS-дерева и вычисление полудоминаторов.* Процедура DFSMP* на вход получает граф в исходной нумерации и строит дерево поиска в глубину, одновременно переводит граф в M-нумерацию.

Кроме уже описанных переменных процедура использует следующие дополнительные переменные: слайс X, в котором '1' помечены позиции дуг, заходящих в еще непронумерованные вершины; ord – переменная типа *integer*, содержащая текущий M-номер; слайс T, в котором '1' отмечены позиции только тех дуг, которые принадлежат пути от корня до вершины с порядковым номером *ord*.

DFSMP* вызывает следующую вспомогательную процедуру OrdNode, которая присваивает вершине *node* M-номер *ord*, увеличивая его на единицу; удаляет из слайса X позиции дуг, заходящих в вершину *node*; отмечает в слайсе Y '1' позиции дуг, которые выходят из вершины *node*.

```
procedure OrdNode(left, right: table; code: table; node: word; var ord:
integer; Var X: slice; Var left1, right1: table; Var NV: table; Var Y: slice);
Var  U, V: slice{code};
     Z:    slice{left};
     w:    word;
     j:    integer;
1. Begin
2.    SET(Z);    SET(U);
3.    MATCH(code, U, node, V);    j := FND(V);
4.    w := ROW(ord, code);
5.    ROW(j, NV) := w;    ord := ord + 1;
6.    MATCH(right, Z, node, Y);    WMERGE(w, Y, right1);
7.    X := X  and  (not Y);
8.    MATCH(left, Z, node, Y);    WMERGE(w, Y, left1);
9.    Y := Y  and  X;
10. End;
```

Теперь приведем процедуру DFSMP*. Предполагаем, что обходится весь уграф, начиная с вершины *root*, поэтому слайс X состоит только из '1', слайс T – только из '0', переменная *ord* равна единице и *node* равна *root*.

```
procedure DFSMP* (left, right, code: table: node: word: var X: slice: T: slice:
var ord: integer: var left1, right1: table: var NV: table: var R: table):
Var  Y:   slice:
     j:   integer:
1. Begin
2.    COL (ord, R) := T:
3.    OrdNode(left, right, code, node, ord, X, left1, right1, NV, Y):
4.    While SOME (Y)  do
5.    begin
6.          j := STEP (Y):    node := ROW (j, right):
7.          T(j) := '1':
8.          DFSMP* (left, right, code, node, X, T, ord, left1, right1, NV, R):
9.          T(j) := '0':    Y := Y  and  X:
10.   end;
11. End:
```

Теорема 1. Процедура DFSMP* выполняет поиск в глубину и строит матрицу путей R за время $O(n\log n)$.

Доказательство. Проводится по N – числу вершин, достижимых из *root*.

Базис индукции. Для $N=1$ доказательство очевидно.

Шаг индукции. Пусть для $N-1$ теорема доказана, докажем для N.

После выполнения процедуры OrdNode слайс X не содержит позиции дуг, заходящих в вершину *root*, и слайс Y не пуст. В строке 6 из слайса Y выбирается дуга, по которой продолжается обход, и ее правая вершина запоминается в переменной *node*. Эта дуга помечается в слайсе T и составляет путь от корня до вершины *node*. Обозначим V_{node} множество вершин, достижимых из вершины *node* по дугам, отмеченным '1' в слайсе X. Заметим, что $|V_{\text{node}}| \le \le |V_{\text{root}}| - 1 = N - 1$.

1. $|V_{\text{node}}| = N - 1$. Тогда по предположению индукции после выполнения строки 8 все вершины пронумерованы, матрица R содержит все пути, слайс X пуст. После выполнения строки 9 процедура останавливается.

2. $0 \le |V_{\text{node}}| < N - 1$. Тогда после выполнения строки 8 вершины, достижимые из *node*, пронумерованы и пути до этих вершин записаны в соответствующие столбцы матрицы R. После выполнения строки 9 в слайсе T отмечены позиции дуг, принадлежащих пути до вершины *root*, из слайса Y удалены позиции дуг, заходящих в пронумерованные вершины. Слайсы X и Y не пусты. Переходим к графу с множеством вершин $V \setminus (\{\text{node}\} \cup V\text{node})$. Для него по предположению индукции теорема доказана. Дерево поиска в глубину для исходного графа получается с помощью объединения двух построенных поддеревьев.

Процедура выполняется за время $O(n\log n)$, так как каждая вершина посещается ровно 1 раз и на обработку вершины необходимо время $O(\log n)$.

Процедура SemiDom вычисляет значения полудоминаторов для вершин графа, заданного в M-нумерации. Результатом будет матрица *sdom* размера $m \times n$, в i-й строке которой записано значение полудоминатора для вершины,

чей номер записан в i-й строке матрицы *right*. Хотя одна вершина может храниться в нескольких строках матрицы *right*, такая запись удобна для работы с путями.

Напомним, что по свойству 4 для вычисления полудоминатора вершины v достаточно найти минимум среди номеров вершин, принадлежащих к следующим множествам: $S_1 = \{w \mid (w, \mathrm{v}) \in E$ и $w < \mathrm{v}\}$ и $S_2 = \{sdom(u) \mid u > \mathrm{v}$ и $\exists (w, \mathrm{v})$: $u \xrightarrow{*} w\}$. Для удобства разобьем второе множество на следующие два множества: $S_2' = \{sdom(w) \mid (w, \mathrm{v}) \in E$ и $w > \mathrm{v}\}$ и $S_2'' = \{sdom(u) \mid u > \mathrm{v}$ и $\exists (w, \mathrm{v})$: $u \xrightarrow{+} w\}$.

Таким образом, вычисление полудоминатора вершины v производится в порядке уменьшения M-номеров: в матрицу *sdom* копируется матрица *left*. Тогда в строке матрицы *sdom*, соответствующей дуге (w, v), будет записана вершина w, если $w < \mathrm{v}$, и полудоминатор вершины w иначе. Таким образом, имея позиции всех таких дуг, получаем множество $S_1 \cup S_2'$. Чтобы получить множество S_2'', нужно найти пути от корня до вершин множества S_2' и выделить из объединения этих путей дуги (u, w), такие, что $u > \mathrm{v}$. Вычисленное значение полудоминатора вершины v записываем во все те строки матрицы *sdom*, которым соответствует вершина v в матрице *right*.

Теперь приводим саму процедуру.

```
procedure SemiDom(left, right: table; code: table; R: table; n: integer; Var sdom: table);
Var   U, U1:          slice {code};
      X, Y, Z, A, D:  slice {left};
      i, j, k:        integer;
      v, w:           word;
1. Begin
2.    SET (U); SET (A);
3.    TCOPY (left, n, sdom);
4.    for i := n downto 2 do
5.    begin
6.          w := ROW (i, code);
7.          MATCH (right, A, w, Z);
(* В слайсе Z '1' отмечены позиции строк, которые задают множество S1 ∪ S2'. *)
8.          GREAT (left, Z, w, X);
(* В слайсе X '1' отмечены позиции таких дуг (u, w), у которых u > w. *)
9.          CLR (D);
10.         while SOME (X) do
11.         begin
12.               j := STEP (X);
13.               v := ROW (j, left);
14.               MATCH (code, U, v, U1); k := FND (U1);
15.               Y := COL (k, R);
16.               D := D or Y;
17.         end;
18.         GREAT (right, D, w, X);
(* В строках 10–18 вычисляем множество S2''. *)
19.         D := D or Z;
(* В слайсе D '1' отмечены позиции вершин, принадлежащих множеству S1 ∪ S2. *)
20.         MIN (sdom, D, X);
21.         k := FND (X);
```

```
22.          v := ROW (k, sdom);
23.          WMERGE (v, Z, sdom);
24.    end;
25. End;
```

Утверждение 1. Процедура SemiDom вычисляет полудоминаторы за время $O(m\log n)$.

Доказательство. Чтобы показать корректность процедуры, рассмотрим более подробно ее работу.

Вычисление полудоминаторов выполняется в цикле 4–24.

В строке 6 бинарный код i-й вершины запоминается в переменной w. После выполнения строк 7–19 в слайсе D '1' отмечены позиции строк матрицы *sdom*, в которых записаны вершины, принадлежащие к множествам S_1 и S_2. Минимальная строка в матрице *sdom* из отмеченных '1' в слайсе D дает $sdom(w)$.

Так как для каждой вершины w нужно просмотреть обратные дуги (u, w) (такие, что $u > w$), то процедура обрабатывает все обратные дуги графа и все вершины. На обработку одной дуги или вершины необходимо время $O(\log n)$. Обратных дуг не больше чем $m - n$. Поэтому время работы процедуры оценивается как $O(m\log n)$.

3.3. *Вычисление непосредственных доминаторов.* Процедура сначала строит дерево поиска в глубину и вычисляет значения полудоминаторов, вызывая уже описанные процедуры.

После этого по свойству 4 легко найти непосредственный доминатор для каждой вершины v ≠ root: необходимо просмотреть значения полудоминаторов для вершин, принадлежащих пути от $sdom$(v) до v. Если на этом пути встречается вершина u, такая, что $s\text{dom}(u) < s\text{dom}(\text{v})$, то $i\text{dom}(\text{v}) = i\text{dom}(u)$, иначе $i\text{dom}(\text{v}) = s\text{dom}(\text{v})$.

```
procedure IDominator (left, right: table; code: table; root: word; n: integer; var idom: table);
Var   sdom, left1, right1, NV, R: table;
      U, U1:                      slice {code};
      X, Y, Z:                    slice {left};
      v, w:                       word;
      k, i, j:                    integer;
1. Begin
2.    SET (U); SET (X); CLR (Y);  i := 1;
3.    DFSMP* (left, right, code, root, X, Y, i, left1, right1, NV, R);
(* Построили DFS-дерево. Матрицы left1 и right1 задают граф в M-нумерации. *)
4.    SemiDom (left1, right1, code, R, n, sdom);
(* Вычислили полудоминаторы. *)
5.    for k := 2 to n do
6.    begin
7.          v := ROW (k, code);
8.          Z := COL (k, R);
9.          MATCH (right1, Z, v, Y);  i := FND (Y);
10.         w := ROW (i, sdom);
(* В данной строке w = sdom (v). *)
11.         GREAT (right1, Z, w, X);
(* В слайсе X '1' отмечены позиции sdom (u) для вершин u, sdom (v) ⟶+ u ⟶* v. *)
```

```
12.          MIN(sdom, X, Y);
13.          if Y(i) = '0' then
(* Случай, когда sdom(v) > min{sdom(u) | sdom(v) -+-> u -*-> v}. *)
14.          begin
15.                  j := FND(Y);
16.                  w := ROW(j, right1);
17.                  MATCH(NV, U, w, U1);
18.                  j := FND(U1);
(* В j записан исходный номер вершины w. *)
19.                  w := ROW(j, idom);
20.          end else
(* Случай, когда sdom(v) = min{sdom(u) | sdom(v) -+-> u -*-> w}. *)
21.          begin
22.                  MATCH(NV, U, w, U1);
23.                  j := FND(U1);
(* В j записан исходный номер вершины sdom(v). *)
24.                  w := ROW(j, code);
25.          end;
(* В этой строке w = idom(v). *)
26.          MATCH(NV, U, v, U1);  j := FND(U1);
27.          ROW(j, idom) := w;
28.    end;
29. End;
```

Утверждение 2. Процедура IDominator вычисляет значение непосредственных доминаторов за время $O(m\log n)$.

Процедура моделирует алгоритм Ленгауэра – Тарьяна.

Для каждой вершины v в порядке увеличения M-номеров в строках 8–12 находим минимальное значение полудоминатора на отрезке пути из вершины сына $s\mathrm{dom}(\mathrm{v})$ до вершины v, позиции минимальных строк отмечены в слайсе Y. Далее разбираем два возможных случая в соответствии со свойством 4.

Проверка свойства 4 для уграфа выполняется за время $O(n\log n)$. Но так как поиск полудоминаторов выполняется за время $O(m\log n)$, то и вся процедура выполняется за время $O(m\log n)$.

Заключение. В данной работе представлена ассоциативная параллельная версия алгоритма Ленгауэра – Тарьяна. Этот алгоритм использует простое и естественное представление исходных данных и прозрачные вспомогательные процедуры.

Хотя оценка по времени сравнима с последовательной версией ($O(m\log n)$ против $O(m\alpha(m,n))$), но стоит отметить, что они считаются поразному. В ассоциативной версии считается число битовых операций, поэтому операции сравнения, сложения, вычитания выполняются за время $O(\log n)$, в то время как в последовательной версии предполагается, что эти операции выполняются за один такт машинного времени.

Можно ожидать, что другие алгоритмы, использующие динамические деревья Тарьяна, также будут естественно представляться на ассоциативном параллельном процессоре.

СПИСОК ЛИТЕРАТУРЫ

1. **Кнут Д.** Искусство программирования. М.: Вильямс, 2000. Т. 1.
2. **Ахо А., Ульман Дж.** Теория синтаксического анализа, перевода и компиляции. Т. 2. Компиляция. М.: Мир,1978.
3. **Lengauer T., Tarjan R. E.** A fast algorithm for finding dominators in a flowgraph // ACM Trans. on Program. Languages and Systems. 1979. **1**, N 1. P. 121.
4. **Евстигнеев В. А., Касьянов В. Н.** Сводимые графы и граф-модели в программировании. Новосибирск: Изд-во ИДМИ, 1999.
5. **Ахо А., Хопкрофт Дж., Ульман Дж.** Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.
6. **Potter J. L.** Associative Computing: A Programming Paradigm for Massively Parallel Computers. New York: London: Plenum Press, 1992.
7. **Непомнящая А. Ш., Владыко М. А.** Сравнение моделей ассоциативного вычисления // Программирование. 1997. № 6. С. 41.
8. **Фостер К.** Ассоциативные параллельные процессоры. М: Энергоиздат, 1981.
9. **Nepomniaschaya A. S., Dvoskina M. A.** A simple implementation of Dijkstra's shortest path algorithm on associative parallel processors // Fundamenta Informaticae. 2000. **43**.
10. **Nepomniaschaya A. S.** Investigation of associative search algorithms in vertical processing systems // Parallel Computing Technologies. M.: NT-Centre, 1993. V. 3. P. 631.
11. **Кормен Т., Лейзерсон Ч., Ривест Р.** Алгоритмы: построение и анализ. М.: МЦНМО, 2000.

Институт вычислительной математики
и математической геофизики СО РАН,
E-mail: borets@ssd.sscc.ru

Поступила в редакцию
25 марта 2003 г.