МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

УДК 591.68

АССОЦИАТИВНЫЙ ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ДЛЯ ДИНАМИЧЕСКОЙ ОБРАБОТКИ ДЕРЕВА КРАТЧАЙШИХ ПУТЕЙ ПОСЛЕ ДОБАВЛЕНИЯ НОВОЙ ДУГИ

А. Ш. Непомнящая, Т. В. Снытникова

Институт вычислительной математики и математической геофизики СО РАН, г. Новосибирск, Россия

Строится ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после добавления новой дуги к ориентированному взвешенному графу. Кратко описывается STAR-машина, которая моделирует работу ассоциативных (контекстно-адресуемых) параллельных систем с простейшими процессорными элементами и вертикальной обработкой информации. Описывается используемая структура данных и её свойства. Ассоциативный параллельный алгоритм представляется в виде процедуры InsertArcSPT, корректность которой доказывается. Показано, что на STAR-машине эта процедура выполняется за время $O(hk)$, где h — число битов, которое требуется для кодирования длины максимального кратчайшего пути; k — число вершин, для которых вычисляются новые кратчайшие пути после добавления новой дуги к исходному графу. Приводятся результаты тестирования реализации алгоритма на графическом ускорителе.

Ключевые слова: *ориентированный взвешенный граф, матрица смежности, инкрементальный алгоритм, аффектная вершина, вертикальная обработка данных, ассоциативный параллельный процессор.*

DOI 10.17223/20710410/46/5

ASSOCIATIVE PARALLEL ALGORITHM FOR DYNAMIC UPDATE OF SHORTEST PATHS TREE AFTER INSERTING AN ARC

A. Sh. Nepomniaschaya, T. V. Snytnikova

The Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia

E-mail: anep@ssd.sscc.ru, snytnikovat@ssd.sscc.ru

The paper proposes an associative parallel algorithm for dynamic update of the shortest paths tree after inserting an arc into a directed weighted graph. First we shortly recall the STAR-machine that simulates the functioning of associative parallel processors. The associative parallel algorithm is given as a procedure InsertArcSPT, correctness of which is proved. Now we give a short description of the associative incremental algorithm. Let the arc (i, j) be added to the graph G. Then we check whether the shortest distance from the root to the vertex j decreases if the shortest

path to the vertex j includes the arc (i, j). If this is true, then the vertex j becomes affected and the new shortest distance to this vertex is written in the matrix of the shortest distances. Then we replace the previous arc in the tree with the arc (i, j). After that, we calculate the distance to those vertexes v for which there is an arc (j, v). Then the all vertexes the distances to which decrease become affected. After that, the new distances are written in the corresponding rows of the shortest distance matrix. Moreover the corresponding arcs are included in the shortest paths tree. We have shown that the time complexity of this procedure is $O(hk)$, while the time complexity of a static associative version of Dijkstra algorithm is $O(hn)$. Here h is the number of bits for coding the maximum of the shortest paths length, n is the number of all graph vertexes and k is the number of affected vertexes for which the new distances are computed. This procedure was tested on the NVIDIA GEFORCE 920M using the implementation of STAR-machine on the GPU. Performance was evaluated on R-MAT graphs which simulate real graphs from social networks and the Internet. Graphs were generated by the GraphHPC-1.0 package with the following parameters: the number of vertexes is defined by a power of two (from 11 to 13), the average degree of vertexes connectivity is 32. We use two modes: zero weighting arcs are added (pessimistic mode) and arcs with random weight are added (realistic mode). In the experiments, we take into account as the runtime of the procedure and the number of affected vertexes. For each test, $\approx 10\% n$ runs were performed. After that, the runs were distributed by the number of affected vertexes. The shortest paths and distances do not change in most runs (more than 50 % in the case of the pessimistic mode and more than 88 % in the case of the realistic mode). In 99 % cases, the number of affected vertexes does not exceed 5 in the first mode and 2 in the second mode. Note that the associative algorithm traverses the graph along the vertexes (outgoing arcs are processed in parallel). While in the sequential version, the graph is traversed along arcs, and the number of arcs that need to be checked can go up to 2000 and 100 accordingly. Also we note that in average the dynamic version runs about 500 times faster in the pessimistic mode and about 1000 times faster in the realistic mode than the static parallel version of Dijkstra's algorithm.

Keywords: *oriented weighted graph, adjacency matrix, incremental algorithm, affected vertex, vertical data processing, associative parallel processor.*

Введение

Ассоциативные (контекстно-адресуемые) параллельные процессоры типа SIMD принадлежат к классу мелкозернистых параллельных систем с последовательно-поразрядной (вертикальной) обработкой информации и простейшими процессорными элементами. В работе [1] такие процессоры называют «системами вертикальной обработки». Перечислим основные достоинства этой архитектуры: параллелизм по данным на базовом уровне, использование двумерных таблиц в качестве простой и естественной структуры данных, параллельный поиск по содержимому памяти и выполнение базовых операций поиска за время, которое пропорционально числу битовых столбцов заданной матрицы, но не числу её строк [2].

В работах [3, 4] построена абстрактная модель типа SIMD (STAR-машина), которая моделирует работу систем вертикальной обработки. Эта модель использует группу элементарных операций, которые позволяют обрабатывать таблицы по содержимому памяти. Для представления ассоциативных параллельных алгоритмов построен язык высокого уровня STAR. На STAR-машине ассоциативные параллельные алгоритмы представляются в виде соответствующих процедур, корректность которых доказывает-

ся. В [4] построены базовые ассоциативные параллельные алгоритмы, которые используются для проектирования ассоциативных алгоритмов для различных приложений. Следуя Фостеру [5], мы полагаем, что каждая элементарная операция STAR-машины выполняется за единицу времени. Отметим, что системы вертикальной обработки наилучшим образом приспособлены для решения задач на графах. Для STAR-машины построены как новые ассоциативные параллельные алгоритмы на графах, так и ассоциативные версии хорошо известных последовательных алгоритмов на графах. В первую очередь были построены ассоциативные версии группы классических алгоритмов на графах. Эта группа, в частности, включает ассоциативные версии алгоритма Дейкстры для нахождения кратчайших путей из заданной вершины [6], алгоритма Флойда для нахождения кратчайших путей между любыми парами вершин [7], алгоритмов Краскала и Прима — Дейкстры для нахождения минимального остовного дерева [8].

Особый интерес представляет реализация на STAR-машине динамических алгоритмов на графах. Целью динамического алгоритма является обработка решения проблемы после динамических изменений в графе быстрее, чем перевычисление графа целиком после каждого локального изменения в нём самым быстрым статическим алгоритмом. На STAR-машине построена группа динамических алгоритмов на графах, которая, в частности, включает ряд ассоциативных алгоритмов для динамической обработки кратчайших путей.

Задача нахождения кратчайших путей возникает в различных приложениях. Известны следующие версии этой проблемы: нахождение кратчайших путей из одной вершины (the single-source shortest paths problem), нахождение подграфа кратчайших путей ориентированного взвешенного графа с единственным стоком (the single-sink shortest paths problem) и нахождение кратчайших путей между каждой парой вершин (the all-pairs shortest paths problem). Типичными операциями для преобразования кратчайших путей являются добавление или удаление одной дуги либо изменение веса одной дуги. Если граф представляет коммуникационную или транспортную сеть, то добавление или удаление дуги отражает такие реальные изменения в сети, как добавление или удаление связей во время существования сети. Алгоритм называется *полностью динамическим* (fully dynamic), если он позволяет выполнять любую последовательность упомянутых операций. Алгоритм называется *инкрементальным*, если он допускает только добавление дуги или уменьшение веса дуги. Если алгоритм допускает только удаление дуги или увеличение веса дуги, то он называется *декрементальным*.

Перечислим группу алгоритмов для динамической обработки кратчайших путей, которые были представлены на STAR-машине. В работах [9, 10] построены ассоциативные версии алгоритмов Рамалингама для динамической обработки подграфа кратчайших путей ориентированного взвешенного графа с одним стоком после добавления к графу новой дуги и после удаления из него одной дуги. На STAR-машине эти версии представлены соответственно в виде процедур InsertArc и DeleteArc, корректность которых доказана. В работах [11, 12] построены ассоциативные версии алгоритмов Рамалингама для динамической обработки кратчайших путей между любыми парами вершин после удаления одной дуги и после добавления новой дуги к заданному графу. Эти версии используют модификацию соответствующих алгоритмов для динамической обработки подграфа кратчайших путей с выделенным стоком после удаления одной дуги и после добавления новой дуги к заданному графу. Ассоциативные версии построены в виде процедур DeleteEdge и InsertEdge, корректность которых доказана. В [13] построен ассоциативный параллельный алгоритм для динамической обработки

дерева кратчайших путей ориентированного графа с неотрицательными весами дуг после удаления из графа одной дуги. Ассоциативный параллельный алгоритм представлен в виде процедуры DeleteArcSPT, корректность которой доказана. Показано, что эта процедура выполняется на STAR-машине за время $O(hk)$, где h — число битов, которое требуется для кодирования максимума кратчайших расстояний от корня, а k — число вершин, для которых строятся новые кратчайшие пути и вычисляются новые кратчайшие расстояния. Показано также, что выполнение процедуры DeleteArcSPT проще, чем процедуры DeleteArc.

В данной работе строится ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей ориентированного графа с неотрицательными весами дуг после добавления к графу новой дуги. Алгоритм представлен на STAR-машине в виде процедуры InsertArcSPT, корректность которой доказана. Показано, что эта процедура выполняется на STAR-машине за время $O(hk)$. В работе показано, как построить структуру данных, которая используется в ассоциативном алгоритме, и что выполнение процедуры InsertArcSPT проще, чем процедуры InsertArc. Приводятся результаты тестирования инкрементального алгоритма InsertArcSPT на графическом ускорителе и сравнивается время выполнения этой процедуры с временем работы статического алгоритма DistSPT [13].

1. Модель ассоциативного параллельного процессора

Приведём краткое описание STAR-машины, которая моделирует работу систем с вертикальной обработкой данных. Модель использует некоторые свойства машины Staran [5] и отечественного ассоциативного процессора [14]. В работе [15] приводится сравнение нашей модели с другими моделями ассоциативной обработки.

STAR-машина определяется как абстрактная модель типа SIMD с вертикальной обработкой информации, основными компонентами которой являются: последовательное устройство управления, в котором записаны программа и скалярные константы; устройство ассоциативной обработки, состоящее из p одноразрядных процессорных элементов; матричная память.

Чтобы моделировать обработку информации в матричной памяти, STAR-машина использует типы данных **slice**, **word** и **table**. С помощью переменной типа **slice** моделируется доступ к таблице по столбцам, а типа **word** — по строкам. С каждой переменной типа **table** ассоциируется матрица из n строк и k столбцов, где $n \leqslant p$. С каждой переменной типа **slice** ассоциируется последовательность из p компонентов, принадлежащих множеству $\{0, 1\}$. Для простоты переменную типа **slice** условимся называть *слайсом*. Более подробно STAR-машина и операции языка STAR описаны в [16].

Приведём группу базовых процедур [4, 6], которые будем использовать в данной работе. С помощью глобального слайса X будем указывать *позиции* анализируемых строк в соответствующей процедуре. В [4, 6] показано, что каждая базовая процедура выполняется за время $O(k)$.

Процедура TCOPY1(T, j, h, F) записывает h столбцов из матрицы T, начиная с $(1 + (j - 1)h)$-го столбца, в результирующую матрицу F, $j \geqslant 1$.

Процедура TMERGE(T, X, F) записывает строки матрицы T, отмеченные «1» в слайсе X, в соответствующие строки матрицы F. Остальные строки матрицы F не меняются.

Процедура SETMIN(T, F, X, Z) определяет позиции строк матрицы T, которые меньше соответствующих строк матрицы F. Она возвращает слайс Z, в котором $Z(j) = 1$ тогда и только тогда, когда $\mathrm{ROW}(j, T) < \mathrm{ROW}(j, F)$ и $X(j) = 1$.

Процедура ADDC(T, X, v, F) одновременно добавляет двоичное слово v к тем строкам матрицы T, которые отмечены «1» в слайсе X, и записывает результат в соответствующие строки матрицы F. Остальные строки матрицы F состоят из нулей.

2. Основные понятия

Пусть $G = (V, E, wt)$ — *ориентированный взвешенный граф*, в котором $V = \{1, 2, \ldots, n\}$ — множество вершин, $E \subseteq V \times V$ — множество ориентированных рёбер (*дуг*) и $wt(e)$ — функция веса. Полагаем, что $|V| = n$ и $|E| = m$. Будем рассматривать графы, дуги которых имеют неотрицательный вес. Будем считать, что $wt(i, j) = \infty$, если $(i, j) \notin E$.

Значение бесконечности выбирается как $\sum_{i=1}^{n} c_i$, где c_i — максимальный вес дуг, выходящих из вершины i в G. Обозначим через h число битов, используемых для кодирования этой величины.

Пусть $e = (i, j)$ — дуга, которая ориентирована от вершины i к вершине j. При этом вершина j называется *головой* дуги e, а вершина i — её *хвостом*.

Кратчайшим путём из вершины v_1 до вершины v_k в графе G назовём последовательность вершин $v_1, v_2, \ldots, v_k$, в которой $(v_i, v_{i+1}) \in E$ для $1 \leqslant i < k - 1$ и сумма весов дуг минимальна. Обозначим через $\mathrm{dist}(l)$ *длину* кратчайшего пути из вершины v_1 до вершины l.

Деревом кратчайших путей T_s с корнем s назовём связный подграф без циклов, который содержит все вершины графа G и в котором путь от корня до любой вершины является *минимальным*.

Пусть дуга (i, j) добавлена к графу G. Вершина y называется *аффектной* в графе G, если кратчайший путь в дереве T_s до вершины y включает дугу (i, j) и длина этого пути меньше, чем длина кратчайшего пути до вершины y, не включающего дугу (i, j).

3. Структура данных

Будем использовать следующую структуру данных:

- матрица смежности G размера $n \times n$, i-й столбец которой, $i = 1, \ldots, n$, хранит с помощью «1» головы дуг, выходящих из вершины i;
- матрица SPT размера $n \times n$, i-й столбец которой хранит головы дуг, выходящих из вершины i и принадлежащих дереву кратчайших путей;
- матрица $Weight$ размера $n \times hn$, элементами которой являются веса дуг. Она состоит из n полей, каждое поле — из h битовых столбцов. Вес дуги (i, j) записывается в j-ю строку i-го поля;
- матрица $Dist$ размера $n \times h$, i-я строка которой хранит кратчайшее расстояние от корня до вершины i;
- слайс $AffectedV$, который хранит с помощью «1» позиции аффектных вершин.

Поясним, как построить приведённую структуру данных. Исходный граф задаём в виде матрицы весов $Weight$ и матрицы смежности G, которая получается из матрицы весов с помощью вспомогательной процедуры Adj [7]. Слайс $AffectedV$ получаем с помощью вспомогательной процедуры UpdateOutgoingEdges, которая приведена в работе далее. Матрицу кратчайших расстояний $Dist$ и матрицу SPT для дерева кратчайших путей будем получать с помощью вспомогательной процедуры DistSPT [13], которая

является специальной реализацией на STAR-машине классического алгоритма Дейкстры [6] для нахождения кратчайших расстояний.

4. Инкрементальный ассоциативный алгоритм для обработки дерева кратчайших путей

Приведём содержательное описание ассоциативного инкрементального алгоритма для обработки дерева кратчайших путей T_s ориентированного графа. Пусть дуга (i, j) добавляется к графу G. Проверяем, уменьшится ли кратчайшее расстояние от корня до вершины j, если кратчайший путь до вершины j будет включать дугу $(i.j)$. Если это верно, то вершина j становится аффектной и новое кратчайшее расстояние до вершины j записывается в матрицу кратчайших расстояний. Поскольку в каждую вершину дерева входит единственная дуга, то в дереве кратчайших путей удаляем дугу, которая первоначально заходила в вершину j, и добавляем туда дугу (i, j). Понятно, что необходимо вычислить расстояние до тех вершин, в которые заходят дуги из вершины j. Если для каких-то вершин расстояние уменьшилось, то эти вершины необходимо пометить как аффектные, расстояния до них записать в соответствующие строки матрицы кратчайших расстояний, а соответствующие дуги внести в дерево кратчайших путей.

Ассоциативный параллельный алгоритм для обработки дуг, выходящих из текущей аффектной вершины, реализован на STAR-машине в виде процедуры UpdateOutgoingEdges (алгоритм 1).

Алгоритм 1. Ассоциативный параллельный алгоритм UpdateOutgoingEdges

Вход: $k, h, G, Weight, Dist, SPT$.
Выход: $Dist, SPT, Y$.

1: С помощью слайса X сохранить позиции дуг, выходящих из k в G.
2: С помощью матрицы $R1$ сохранить веса дуг, выходящих из k в $Weight$.
3: С помощью матрицы $R1$ сохранить новые расстояния от корня до тех вершин графа G, позиции которых отмечены «1» в слайсе X.
4: С помощью слайса Y сохранить номера тех вершин из слайса X, для которых новые кратчайшие расстояния от корня уменьшились *// слайс Y будет хранить новые аффектные вершины.*
5: Записать в матрицу $Dist$ новые кратчайшие расстояния от корня до тех аффектных вершин, позиции которых хранятся в слайсе Y.
6: Перестроить дерево кратчайших путей следующим образом. Пусть слово $w0$ содержит единственную «1» в k-м бите. Тогда в дереве кратчайших путей SPT записать слово $w0$ в каждую строку, отмеченную «1» в слайсе Y.

Ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после добавления к графу G дуги (i, j) весом $v1$ реализован на STAR-машине в виде процедуры InsertArcSPT (алгоритм 2).

На рис. 1 показан пример изменения кратчайших расстояний и дерева кратчайших расстояний после добавления дуги $(3,6)$ с весом 2. Метками у вершин показаны кратчайшие расстояния, а сплошными стрелками обозначены дуги, принадлежащие дереву кратчайших расстояний. Аффектными будут вершины *6*, *5* и *8*. При этом дуга $(6,8)$ принадлежала дереву кратчайших расстояний и до добавления дуги, в то время как дуга $(6,5)$ заменила в дереве дугу $(4,5)$.

Алгоритм 2. Ассоциативный параллельный алгоритм InsertArcSPT

Вход: $(i,j), v1, G, Weight, Dist, SPT$.
Выход: $Dist, SPT$.

1: Включить дугу (i,j) в матрицу смежности G, а её вес $v1$ добавить в матрицу $Weight$.
2: С помощью слова $v2$ сохранить кратчайшее расстояние до вершины i, с помощью слова $v3$ — длину нового пути до вершины j.
3: С помощью слова $v4$ сохранить старое кратчайшее расстояние до вершины j.
4: **Если** $v3 < v4$, **то**
5: включить вершину j в слайс $AffectedV$ и записать новое кратчайшее расстояние $v3$ до вершины j в матрицу $Dist$;
6: удалить из матрицы SPT дугу, которая заходила в вершину j перед добавлением к графу G дуги (i,j), и включить в матрицу SPT дугу (i,j).
7: **Пока** $AffectedV \neq \varnothing$:
8: в слайсе $AffectedV$ выделить верхнюю вершину k с помощью операции STEP;
9: выполнить процедуру UpdateOutgoingEdges$(k, h, SPT, Weight, Dist, Y)$;
10: добавить в слайс $AffectedV$ новые аффектные вершины, позиции которых хранятся в слайсе Y.

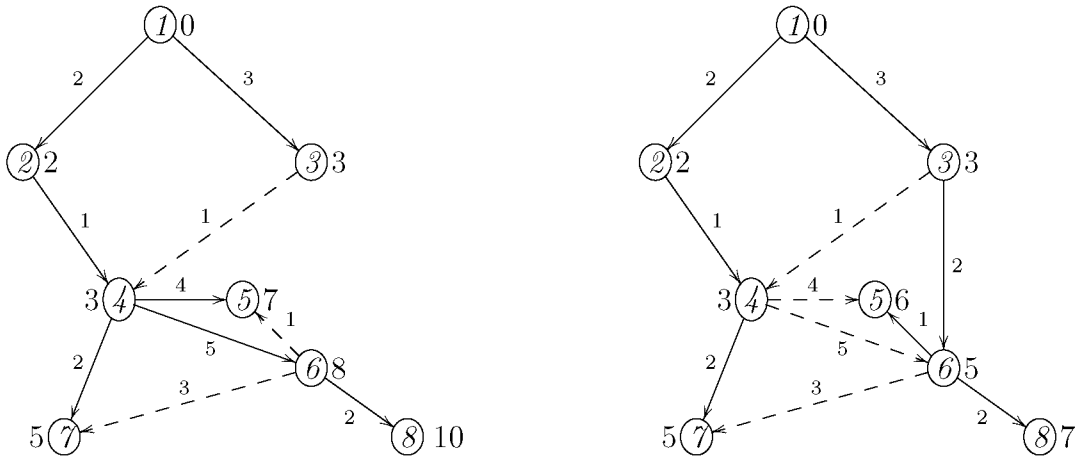

Рис. 1. Дерево кратчайших расстояний до (*а*) и после (*б*) добавления дуги (*3,6*) с весом 2

5. Выполнение инкрементального ассоциативного алгоритма на STAR-машине

Приведём вспомогательную процедуру UpdateOutgoingEdges. Зная аффектную вершину k и текущие матрицы SPT, $Weight$ и $Dist$, она возвращает слайс Y, который хранит аффектные вершины, смежные с вершиной k, и матрицу $Dist$, которая хранит новые кратчайшие расстояния от корня до этих вершин.

```
procedure UpdateOutgoingEdges(k,h:integer; G:table; Weight:table;
  var Dist:table; var SPT:table; var Y:slice(G));
/* Здесь k - это аффектная вершина.*/
var v: word(Dist); w:word(SPT);
```

```
 X:\,slice(SPT);
R1,R2: table;
i:integer;
 1. Begin X:=COL(k,G);
 2.    TCOPY1(Weight,k,h,R1);
/* Матрица R1 хранит веса дуг, выходящих из вершины k. */
 3.    v:=ROW(k,Dist);
/* Слово v хранит новое кратчайшее расстояние до вершины k. */
 4.    ADDC(R1,X,v,R2);
/* Матрица R1 хранит новые расстояния от корня до вершин,
выходящих из вершины k в матрице SPT. */
 5.    SETMIN(R2,Dist,X,Y);
/* Слайс Y хранит новые аффектные вершины.\,*/
 6.    TMERGE(R2,Y,Dist);
/* Новые расстояния до аффектных вершин смежных с вершиной k
записываются в соответсвующие строки матрицы Dist. */
 7.    CLR(w); w(k):='1'; X:=Y;
/* Вершина k помечается как вероятный отец для этих вершин */
 8.    While SOME(X) do
 9.    begin
10.       i:=FND(X);
11.       ROW(i,SPT):=w;
12.    end;
13. End;
```

Утверждение 1. Пусть граф G задаётся матрицами G, $Weight$, SPT и $Dist$. Пусть также заданы добавляемая дуга (i, j) и аффектная вершина k. Тогда после выполнения процедуры UpdateOutgoingArcs слайс Y будет хранить аффектные вершины, смежные с вершиной k, матрица $Dist$ — новые кратчайшие расстояния от корня до этих вершин, а матрица SPT — обновлённое дерево кратчайших путей.

Доказательство. Пусть вершина r смежна с вершиной k в матрице G и является аффектной, однако после выполнения процедуры UpdateOutgoingArcs r-й бит слайса Y равен нулю. Докажем, что это противоречит выполнению процедуры UpdateOutgoingArcs.

Действительно, после выполнения строк 1–2 слайс X хранит позиции дуг, выходящих в матрице G из вершины k, а матрица $R1$ — веса этих дуг. По предположению вершина r смежна с вершиной k в матрице G. Поэтому $X(r) = 1$, а в r-й строке матрицы $R1$ записан вес дуги (k, r). После выполнения строк 3–6 в r-й строке матрицы $R2$ будет записано новое кратчайшее расстояние от корня до вершины r. Все вершины, для которых это расстояние меньше $\mathrm{ROW}(r, Dist)$, отмечаются «1» в слайсе Y. Поэтому после выполнения строк 5–6 получаем, что $Y(r) = 1$ и новое кратчайшее расстояние от корня до вершины r записано в r-ю строку матрицы $Dist$. После выполнения строки 7 слово w содержит единственную «1» в k-м бите, а в слайсе X отмечены все вершины, смежные k, для которых расстояние уменьшилось. Тогда в дереве кратчайших путей SPT записываем слово w в каждую строку i, отмеченную «1» в слайсе X, таким образом вставляя дуги (k, i) в дерево кратчайших расстояний. Это противоречит нашему допущению. ■

Приведём основную процедуру InsertArcSPT. Зная параметр h, добавляемую в граф дугу (i, j) весом $v1$, текущие матрицы G, $Weight$, $Dist$ и SPT, она возвращает изменённые матрицы G, SPT, $Weight$ и $Dist$.

```
procedure InsertArcSPT(i,j,h:integer; v1:word(Dist);
    var Weight:table; var G,SPT:table; var Dist:table);
/* Дуга (i,j) весом v1 добавляется в граф G.*/
var k,r,l,l1,l2:integer;
   AffectedV,X,Y:slice(G);
   v2,v3,v4:word(Dist);  v5:word(G);
   v6:word(Weight);
 1. Begin X:=COL(i,G); X(j):='1'; COL(i,G):=X;
 /* Дуга (i,j) добавлена в граф G. */
 2.   l1:=1+(i-1)h; l2:=ih;
 3.   v6:=ROW(j,Weight); Rep(l1,l2,v1,v6);
 4.   ROW(j,Weight):=v6;
 /* В матрицу Weight записывается вес новой дуги (i,j). */
 5.   CLR(AffectedV); v2:=ROW(i,Dist);
 /* Строка v2 хранит кратчайшее расстояние от корня до вершины i. */
 6.   v3:=ADD(v1,v2);
 /* Строка v3 хранит новое кратчайшее расстояние от корня до вершины j.*/
 7.   v4:=ROW(j,Dist);
 /* Строка v4 хранит старое кратчайшее расстояние от корня до вершины j. */
 8.   if LESS(v3,v4) then
 9.   begin AffectedV(j):='1';
10.       ROW(j,Dist):=v3;
 /* Новое кратчайшее расстояние от корня до вершины j
 записывается в матрицу Dist. */
11.       CLR(v5); v5(i):='1'; ROW(j,SPT):=v5;
 /* Дуга (i,j) добавлена в матрицу SPT. */
12.       while SOME(AffectedV) do
13.       begin k:=STEP(AffectedV);
14.          UpdateOutgoingEdges(k,h,G,Weight,Dist,SPT,Y);
 /* Слайс Y хранит новые аффектные вершины, которые будут добавляться
 в слайс AffectedV. */
15.          AffectedV:=AffectedV or Y;
16.       end;
17.   end;
18. End;
```

Теорема 1. Пусть заданы параметр h и дуга (i, j) веса $v1$, которая добавляется к графу G. Тогда после выполнения процедуры InsertArcSPT матрицы G, $Weight$, SPT и $Dist$ будут задавать текущее состояние изменённого графа.

Доказательство. Индукция по числу аффектных вершин $r \geqslant 0$, которые образуются после добавления дуги (i, j) к графу G.

Б а з и с докажем для случая, когда $r \leqslant 1$. После выполнения строк 1–4 дуга (i, j) будет добавлена к графу G, а её вес — в матрицу $Weight$. После выполнения строк 5–6 слайс $AffectedV$ состоит из нулей, переменная $v3$ хранит новое кратчайшее расстояние от корня до вершины j, а $v4$ — старое кратчайшее расстояние до вершины j. Если

$v3 \geqslant v4$, то переходим на конец процедуры (строка 18). В противном случае переходим на строку 9. После выполнения строк 9–10 позиция вершины j отмечается «1» в слайсе $AffectedV$ и в матрицу $Dist$ записывается новое кратчайшее расстояние до вершины j. Осталось внести изменения в матрицу SPT — включить новую дугу (i, j). После выполнения строки 11 в j-ю строку матрицы SPT записывается слово, содержащее единственную «1» в i-й позиции. Так как $AffectedV \neq \varnothing$, выполняем цикл `while SOME(AffectedV) do` (строки 12–16). После выполнения строки 13 получаем, что $k = j$ и $AffectedV = \varnothing$. По предположению индукции имеется не более одной аффектной вершины. Поэтому в результате выполнения строки 14 получаем, что $Y = \varnothing$. Тогда после выполнения строки 15 переходим на конец процедуры.

Ш а г и н д у к ц и и. Пусть утверждение теоремы выполняется для случая $r \geqslant 1$. Докажем справедливость теоремы для $r + 1$ аффектных вершин. По индуктивному предположению после обработки первых r аффектных вершин дуга (i, j) будет добавлена в матрицу G, её вес — в матрицу $Weight$, дуга (l, j) удалена из матрицы SPT, а дуга (i, j) туда добавлена. Кроме того, в матрицу $Dist$ будут записаны новые кратчайшие расстояния до первых r аффектных вершин. Поскольку после обработки первых r аффектных вершин в слайсе $AffectedV$ останется единственная аффектная вершина, то после выполнения строки 14 по утверждению 1 для этой вершины будут обновлены матрицы $Dist$ и SPT. ■

6. Результаты тестирования на GPU

Для использования ассоциативных параллельных алгоритмов на практике разработана реализация STAR-машины на графических ускорителях:

1) в виде библиотеки на CUDA реализованы типы данных для языка STAR и простейшие операции над ними [16, 17];
2) библиотека стандартных процедур языка STAR эффективно реализована на GPU [18].

С помощью реализации STAR-машины проведено тестирование алгоритма InsertArcSPT на графическом ускорителе NVIDIA GEFORCE 920M.

Оценка производительности проводилась на R-MAT-графах. Они хорошо моделируют реальные графы из социальных сетей и интернета, а также являются достаточно сложными для анализа. Генерация графов производилась пакетом GraphHPC-1.0 [19] со следующими параметрами: количество вершин задаётся степенью двойки, средняя степень связности вершины равна 32. В таком R-MAT-графе имеется одна большая связная компонента и некоторое количество небольших связных компонент или висящих вершин.

Напомним, что порядок сложности ассоциативного параллельного алгоритма и его реализации на CUDA отличаются на $O(\log_{64} n)$, если в алгоритме используются операции, критичные к синхронизации($STEP(X)$, $FND(X)$, $SOME(X)$). Таким образом, сложность реализации алгоритма InsertArcSPT равна $O(hk \log_{64}(n))$, а реализации DistSPT — $O(hn \log_{64}(n))$. Здесь n — число вершин в графе; k — число аффектных вершин; h — число битов, которое требуется для кодирования длины максимального кратчайшего пути (по умолчанию 32). В экспериментах использовались два режима:

— **R-MAT*** — добавлялись дуги с весом 0 (пессимистичный сценарий);
— **R-MAT** — добавлялись дуги со случайным весом (реалистичный сценарий).

Учитывалось не только время работы процедуры, но и количество аффектных вершин (вершин, для которых длина кратчайшего пути изменилась). Для каждого теста про-

водилось $\approx n/10$ запусков с добавлением дуги. После этого запуски распределялись по количеству аффектных вершин.

На рис. 2 показано распределение запусков по количеству аффектных вершин при добавлении дуги с весом 0, а на рис. 3 — дуги со случайным весом. Видно, что для R-MAT-графов в большинстве случаев (более 50 % при добавлении дуги с нулевым весом и более 88 % — со случайным весом) кратчайшие пути и расстояния не изменяются. В 99 % случаев число аффектных вершин не превышает 5 в первом режиме и 2 — во втором. Отметим, что в отличие от ассоциативного алгоритма, ведущего обход графа по вершинам (исходящие дуги обрабатываются параллельно), в последовательном варианте обход графа совершается по дугам, и число дуг, которые необходимо проверить, может доходить до 2000 в первом режиме и до 100 — во втором.

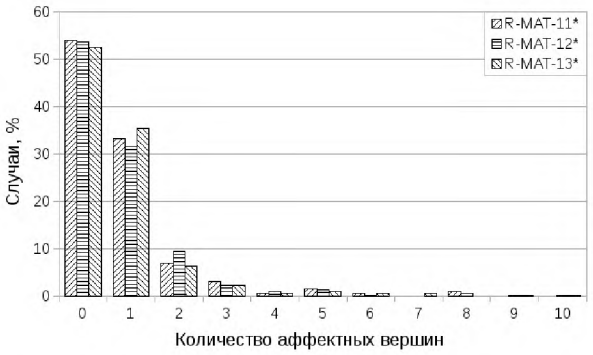

Рис. 2. Распределение запусков при добавлении дуги с весом 0

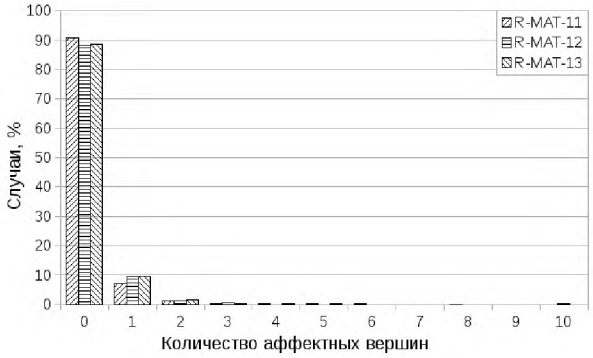

Рис. 3. Распределение запусков при добавлении дуги со случайным весом

В таблице приводятся среднее время выполнения процедуры DistSPT и максимальное время выполнения процедуры InsertArcSPT в двух режимах добавления дуг. Здесь n — число вершин в графе, k — число аффектных вершин в худшем случае (число аффектных вершин при добавлении дуги с максимальным временем обработки/максимальное число аффектных вершин). Время и количество аффектных вершин для режима R-MAT* отмечены «*».

Сравнение времени работы статического и динамического ассоциативных алгоритмов на R-MAT-графах

Граф	n	DistSPT	InsertArcSPT*	k*	InsertArcSPT	k
R-MAT-11	2048	6,171	0,012	6/8	0,009	1/8
R-MAT-12	4096	9,643	0,017	5/10	0,012	3/10
R-MAT-13	8192	29,475	0,038	7/15	0,020	4/13

Таким образом, при использовании динамического алгоритма время определения кратчайших расстояний уменьшается на несколько порядков, так как InsertArcSPT в большей степени зависит от числа аффектных вершин, чем от общего числа вершин в графе.

Заключение

Построен ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после добавления в ориентированный граф одной дуги. Этот алгоритм использует простую структуру данных, которая позволяет выполнять доступ к данным по содержимому памяти. Ассоциативный алгоритм представлен на STAR-

машине в виде процедуры InsertArcSPT, корректность которой доказана. Процедура выполняется на STAR-машине за время $O(hk)$, где h — число битов для кодирования бесконечности, а k — число аффектных вершин, для которых строятся новые кратчайшие пути. Полученная оценка является оптимальной, поскольку параметр h не меняется с ростом k, а обрабатывать надо все аффектные вершины.

Представлены результаты тестирования реализации данного алгоритма на графических ускорителях. При тестировании на R-MAT-графах фиксировалось не только время выполнения алгоритма, но и количество аффектных вершин. Получено, что для этого класса графов в более чем 50 % случаев при добавлении дуги кратчайшие расстояния не изменяются. Количество аффектных вершин не превышает 5 в 99 % случаев.

Приведено сравнение времени работы динамического алгоритма InsertArcSPT с временем работы статического алгоритма DistSPT.

ЛИТЕРАТУРА

1. *Fet Y. I.* Vertical processing systems: a survey // IEEE Micro. 1995. V. 15. Iss. 1. P. 65–75.
2. *Potter J. L.* Associative Computing: a Programming Paradigm for Massively Parallel Computers. Boston: Perseus Publishing, 1991. 304 p.
3. *Nepomniaschaya A. Sh.* Language STAR for associative and parallel computation with vertical data processing // Parallel Computing Technologies. Singapore: World Scientific, 1991. P. 258–265.
4. *Nepomniaschaya A. Sh.* Basic associative parallel algorithms for vertical processing systems // Bulletin of the Novosibirsk Computing Center. 2009. Ser. Comp. Sci. No. 9. P. 63–77.
5. *Foster C. C.* Content Addressable Parallel Processors. N.Y.: John Wiley & Sons, 1976. 233 p.
6. *Nepomniaschaya A. Sh. and Dvoskina M. A.* A simple implementation of Dijkstra's shortest path algorithm on associative parallel processors // Fundamenta Informaticae. IOS Press, 2000. V. 43. P. 227–243.
7. *Nepomniaschaya A. S.* Solution of path problems using associative parallel processors // Proc. ICPADS'97. Seoul: IEEE Computer Society Press,1997. P. 610–617.
8. *Непомнящая А. Ш.* Сравнение алгоритмов Прима — Дейкстры и Краскала с помощью ассоциативного параллельного процессора // Кибернетика и системный анализ. 2000. № 2. С. 19–27.
9. *Непомнящая А. Ш.* Ассоциативная версия алгоритма Рамалингама для динамической обработки подграфа кратчайших путей после добавления к графу новой дуги // Кибернетика и системный анализ. 2012. № 3. С. 45–57.
10. *Nepomniaschaya A. S.* Efficient parallel implementation of the Ramalingam decremental algorithm for updating the shortest paths subgraph // Computing and Informatics. 2013. V. 32. P. 331–354.
11. *Nepomniaschaya A. S.* Associative version of the Ramalingam decremental algorithm for the dynamic all-pairs shortest-path problem // Bulletin of the Novosibirsk Computing Center. 2016. Ser. Comp. Sci. No. 39. P. 37–50.
12. *Nepomniaschaya A. S.* Associative version of the Ramalingam incremental algorithm for the dynamic all-pairs shortest-path problem // Bulletin of the Novosibirsk Computing Center. 2016. Ser. Comp. Sci. No. 40. P. 75–86.
13. *Непомнящая А. Ш.* Ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей // Моделирование и анализ информационных систем. 2013. Т. 20. № 2. С. 5–22.

14. *Mirenkov N. N.* The Siberian approach for an open-system high-performance computing architecture // Computing and Control Engineering J. 1992. V. 3. No. 3. P. 137–142.
15. *Непомнящая А. Ш., Владыко М. А.* Сравнение моделей ассоциативного вычисления // Программирование. 1997. № 6. С. 41–50.
16. *Снытникова Т. В., Непомнящая А. Ш.* Решение задач на графах с помощью STAR-машины, реализуемой на графических ускорителях // Прикладная дискретная математика. 2016. № 3(33). С. 98–115.
17. *Snytnikova T. V. and Snytnikov A. V.* Implementation of the STAR-machine on GPU // Bulletin of the Novosibirsk Computing Center. 2016. Ser. Comp. Sci. No. 39. P. 51–60.
18. *Снытникова Т. В.* Реализация модели ассоциативных вычислений на GPU: библиотека базовых процедур языка STAR // Вычислительные методы и программирование. Новые вычислительные технологии. 2018. № 19(1). С. 85–95.
19. `http://www.dislab.org/GraphHPC-2018/contest/GraphHPC-1.0.tgz` — GraphHPC-1.0. 2018.

REFERENCES

1. *Fet Y. I.* Vertical processing systems: a survey. IEEE Micro, 1995, vol. 15, iss. 1, pp. 65–75.
2. *Potter J. L.* Associative Computing: a Programming Paradigm for Massively Parallel Computers. Boston, Perseus Publishing, 1991. 304 p.
3. *Nepomniaschaya A. Sh.* Language STAR for associative and parallel computation with vertical data processing. Parallel Computing Technologies, Singapore, World Scientific, 1991, pp. 258–265.
4. *Nepomniaschaya A. Sh.* Basic associative parallel algorithms for vertical processing systems. Bulletin of the Novosibirsk Computing Center, 2009, ser. Comp. Sci, no. 9. pp. 63–77.
5. *Foster C. C.* Content Addressable Parallel Processors. N.Y., John Wiley & Sons, 1976. 233 p.
6. *Nepomniaschaya A. Sh. and Dvoskina M. A.* A simple implementation of Dijkstra's shortest path algorithm on associative parallel processors. Fundamenta Informaticae, IOS Press, 2000, vol. 43, pp. 227–243.
7. *Nepomniaschaya A. S.* Solution of path problems using associative parallel processors. Proc. ICPADS'97, Seoul, IEEE Computer Society Press, 1997, pp. 610–617.
8. *Nepomniaschaya A. S.* Comparison of performing the Prim — Dijkstra algorithm and the Kruskal algorithm by means of associative parallel processors. Cybernetics and System Analysis, 2000, no. 2, pp. 19–27.
9. *Nepomniaschaya A. S.* Associative version of the Ramalingam algorithm for the dynamic update of the shortest paths subgraph after inserting a new edge. Cybernetics and System Analysis, 2012, no. 3, pp. 45–57.
10. *Nepomniaschaya A. S.* Efficient parallel implementation of the Ramalingam decremental algorithm for updating the shortest paths subgraph. Computing and Informatics, 2013, vol. 32, pp. 331–354.
11. *Nepomniaschaya A. S.* Associative version of the Ramalingam decremental algorithm for the dynamic all-pairs shortest-path problem. Bulletin of the Novosibirsk Computing Center, 2016, ser. Comp. Sci., no. 39, pp. 37–50.
12. *Nepomniaschaya A. S.* Associative version of the Ramalingam incremental algorithm for the dynamic all-pairs shortest-path problem // Bulletin of the Novosibirsk Computing Center, 2016, ser. Comp. Sci., no. 40, pp. 75–86.
13. *Nepomniaschaya A. S.* Assotsiativnyy parallel'nyy algoritm dlya dinamicheskoy obrabotki dereva kratchayshikh putey [Associative parallel algorithm for dynamic update shortest

paths tree]. Modeling and Analysis of Information Systems, 2013, vol. 20, no. 2, pp. 5–22. (in Russian)

14. *Mirenkov N. N.* The Siberian approach for an open-system high-performance computing architecture. Computing and Control Engineering J., 1992, vol. 3, no. 3, pp. 137–142.
15. *Nepomniaschaya A. S. and Vladyko M. A.* A comparison of associative computation models. Programming and Computer Software, 1997, no. 6, pp. 319–324.
16. *Snytnikova T. V. and Nepomniaschaya A. Sh.* Resheniye zadach na grafakh s pomoshch'yu STAR-mashiny, realizuyemoy na graficheskikh uskoritelyakh [Solution of graph problems by means of the STAR-machine being implemented on GPUs]. Prikladnaya Diskretnaya Matematika, 2016, no. 3(33), pp. 98–115. (in Russian)
17. *Snytnikova T. V. and Snytnikov A. V.* Implementation of the STAR-machine on GPU. Bulletin of the Novosibirsk Computing Center, 2016, ser. Comp. Sci., no. 39, pp. 51–60.
18. *Snytnikova T. V.* Realizatsiya modeli assotsiativnyh vychisleniy na GPU: biblioteka bazovyh protsedur yazyka STAR [Implementation of an associative-computing model on GPU: a basic procedure library of the STAR language]. Numerical Methods and Programming. Advanced Computing, 2018, no. 19(1), pp. 85–95.(in Russian)
19. `http://www.dislab.org/GraphHPC-2018/contest/GraphHPC-1.0.tgz` — GraphHPC-1.0, 2018.