ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

УДК 519.68

DOI: 10.17223/19988605/54/11

А.Ш. Непомнящая, Т.В. Снытникова

АССОЦИАТИВНАЯ ВЕРСИЯ ИНКРЕМЕНТАЛЬНОГО АЛГОРИТМА РАМАЛИНГАМА ДЛЯ РЕШЕНИЯ ПРОБЛЕМЫ ДОСТИЖИМОСТИ В ПОТОКОВЫХ ГРАФАХ С ОДНИМ ИСТОЧНИКОМ

Работа выполнена в рамках государственного задания ИВМиМГ СО РАН (0251-2021-0005).

Построена ассоциативная версия последовательного динамического алгоритма Рамалингама. Эта версия представлена в виде процедуры на языке STAR, корректность которой доказана. Результаты тестирования на графических ускорителях показывают, что ассоциативный динамический алгоритм выполняется в несколько раз быстрее статического ассоциативного алгоритма и делает значительно меньше итераций, чем последовательный динамический алгоритм.

Ключевые слова: SIMD; графический ускоритель; контекстно-адресуемая память; потоковый граф; динамический алгоритм.

В данной работе строится ассоциативная версия алгоритма Рамалингама для решения динамической проблемы достижимости в потоковых графах с одним источником при добавлении новой дуги. Эта задача возникает в различных приложениях, таких как компиляторы, системы верификации [1, 2], а также анализ и синтез информации в геоинформационных системах (ГИС) [3] и обработка запросов в базах данных с транзитивными отношениями [4].

С ростом объемов обрабатываемых данных возрастает необходимость разработки динамических алгоритмов. Такие алгоритмы позволяют после локального изменения в графе получать решение быстрее, чем перевычисление графа целиком после каждого изменения в нем самым быстрым статическим алгоритмом.

Для решения динамической проблемы достижимости нами используется модель ассоциативных (контекстно-адресуемых) процессоров типа SIMD с вертикальной обработкой данных (STAR-машина). К основным достоинствам ассоциативных архитектур относятся параллелизм по данным на базовом уровне, использование двумерных таблиц в качестве структуры данных, параллельный поиск по содержимому памяти [5]. Такие архитектуры ориентированы на решение задач нечисловой обработки данных. В частности, сюда относятся реляционная алгебра, теория графов, обработка сейсмических данных.

Прогресс в компьютерной индустрии и полупроводниковых технологиях за последние годы сделал ассоциативную обработку привлекательной. Приведем несколько современных результатов, подтверждающих это утверждение. В работе [6] исследуется ассоциативный процессор (AP), базирующийся на резистентной контекстно-адресуемой памяти. В статье показывается, что технология резистентной памяти предположительно позволяет масштабировать AP с нескольких миллионов до несколько сотен миллионов процессорных элементов на отдельный силиконовый кристалл. Более

того, резистентный АР позволяет получать большие масштабируемость и производительность по сравнению с популярными сегодня графическими ускорителями, также относящимися к типу SIMD. В работе [7] описана крупнейшая электронная система ATLAS Fast Tracker с использованием ассоциативной архитектуры, которая проектировалась как часть детектора ATLAS Большого Адронного Коллайдера (Large Hadron Collider, LHC), предназначенного для изучения процессов в физике высоких энергий. Задача системы – распознать и сохранить полезные события при значительном подавлении фоновых процессов (соотношение оценивается как 1 к 10^9). Fast Tracker использует аппаратную технологию с огромным параллелизмом, объединяя интегральные микросхемы ассоциативной памяти, FPGA (ПЛИСы) и высокоскоростные каналы связи. Тем не менее в настоящее время не существует общедоступных ассоциативных архитектур.

В работах [8, 9] 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 был построен язык высокого уровня STAR. На STAR-машине ассоциативные параллельные алгоритмы представляются в виде соответствующих процедур, корректность которых доказывается. В [9] построена библиотека базовых ассоциативных параллельных алгоритмов. В работах [10–11] приведена эффективная реализация ассоциативных операций STAR-машины и библиотеки базовых ассоциативных параллельных алгоритмов на графических ускорителях, позволяющая использовать разработанные алгоритмы на практике.

Для STAR-машины были построены как новые ассоциативные параллельные алгоритмы на графах, так и ассоциативные версии классических алгоритмов на графах. Кроме того, была построена группа динамических алгоритмов на графах, в частности ряд ассоциативных алгоритмов для динамической обработки кратчайших путей [12–15].

Вначале мы представим краткое описание STAR-машины. В разделе 2 приводится описание последовательного алгоритма Рамалингама. В разделе 3 описано представление динамических деревьев на STAR-машине, поскольку алгоритм Рамалингама использует эту структуру данных. В разделе 4 мы приводим ассоциативную версию алгоритма Рамалингама, доказываем ее корректность и перечисляем основные достоинства этой версии. В разделе 5 приводятся результаты тестирования инкрементального алгоритма на графическом ускорителе. Представленный алгоритм выполняется в несколько раз быстрее статического ассоциативного алгоритма на графах с различным числом достижимых вершин (общее число вершин до 5 000, число дуг до 90 000). И за счет обработки вершины вместе с исходящими ребрами число итераций ассоциативного инкрементального алгоритма может значительно сокращаться ($\sim |E|/|V|$) по сравнению с последовательным алгоритмом.

1. Модель ассоциативного параллельного процессора

В данном разделе приведем краткое описание STAR-машины, которая моделирует работу систем с вертикальной обработкой данных. В модели используются свойства как машины Staran [16], так и отечественного ассоциативного процессора. В работе [17] приводится сравнение STAR-машины с другими моделями ассоциативной обработки.

STAR-машина определяется как абстрактная модель типа SIMD с вертикальной обработкой информации (рис. 1); ее основными компонентами являются: последовательное устройство управления, в котором записаны программа и скалярные константы; устройство ассоциативной обработки, состоящее из p одноразрядных процессорных элементов; матричная память.

Для обработки 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 с помощью типа **word** – доступ по строкам. С каждой переменной типа **table** ассоциируется матрица из n строк и k столбцов, где $n \leq p$. С каждой переменной типа **slice** ассоциируется последовательность из p компонентов, принадлежащих множеству $\{0, 1\}$.

Рис. 1. STAR-машина
Fig. 1. STAR-machine

В данной работе для переменной X типа **slice** используются следующие операции: $CLR(X)$ – запись нулей в слайс X ; $X(i)$ – определение значение i -го бита в слайсе X ; $STEP(X)$ – определение позиции верхней единицы с ее последующим удалением; $SOME(X)$ проверяет, имеется ли в слайсе X хотя бы одна единица; побитовые операции *not*, *and*, *or*. Для переменной T типа **table** используются следующие операции: выделение i -го столбца $COL(i, T)$ и выделение j -й строки $ROW(j, T)$. Следуя Фостеру [16] будем считать, что каждая элементарная операция STAR-машины выполняется за единицу времени. Поэтому сложность любой процедуры будет оцениваться общим числом выполненных элементарных операций. Подробное описание STAR-машины и операций языка STAR приводится в работе [8].

2. Описание алгоритма Рамалингама для добавления новой дуги

Алгоритм [18] использует структуру «динамическое дерево» [19], с заданными операциями: $link(u, v)$ – к вершине u остоного дерева T добавить поддереву с корнем в вершине v («подвесить» поддерево с корнем в v к вершине u); $cut(u, v)$ – в остоном дереве T удалить дугу (u, v) вместе с поддеревом с корнем в вершине v («разрезать» дугу (u, v)).

Заданы: $G = (V, E)$ – ориентированный граф; T – динамическое остоное дерево; (u, v) – дуга, добавляемая в граф; $WorkSet$ множество дуг.

В этом алгоритме Рамалингама использует следующие понятия. Для каждой вершины $v \in V$ задано $support(v)$ – множество поддерживающих вершин для v . Поддерживающая вершина определяется следующим образом: u – поддерживающая вершина для v , если u – достижима, дуга $(u, v) \in E$ и u не является потомком v в дереве T . Для каждой вершины $v \in V$ $reachable(v)$ истинно тогда и только тогда, когда v – достижима. $Succ(v)$ – множество всех вершин, в каждую из которых заходит дуга из вершины v .

Алгоритм Рамалингама работает следующим образом.

1. Добавить дугу (u, v) в $E(G)$
2. **if** $reachable(u)=true$ **then**
3. $WorkSet := (u, v)$
4. **while** $WorkSet \neq \emptyset$ **do**
5. Выбрать и удалить очередную дугу, скажем (x, y) , из $WorkSet$
6. **if** $reachable(y)=false$ **then**
7. $support(y) := x$
8. $reachable(y) := true$
9. $link(x, y)$ в T
10. **for** $\forall z \in Succ(y)$

11. Добавить дугу (y,z) в множество WorkSet
12. **else**
13. **if** x не потомок y в T **then**
14. Добавить x в support(y)

Рис. 2. Остовное дерево на множестве достижимых вершин до и после добавления дуги (1, 5).

Двойной чертой отмечены дуги, попадающие в множество WorkSet

Fig. 2. A spanning tree on the set of reachable vertices before and after adding an arc (1, 5).

Double dashes indicate arcs that fall into the WorkSet

На рис. 2 показано изменение остовного дерева после добавления к графу дуги (1, 5). Дуги, которые добавлялись в множество WorkSet в процессе исполнения алгоритма Рамалингама, обозначены двойной чертой.

3. Представление динамических деревьев на STAR-машине

Во многих алгоритмах на STAR-машине деревья задаются как матрица смежности (допустим, T). Чтобы задать структуру «динамическое дерево» на STAR-машине, такое представление необходимо дополнить матрицей потомков по дереву $Desc$: в i -м столбце отмечены все потомки вершины i (т.е. те вершины, которые достижимы из вершины i в T).

В ассоциативном инкрементальном алгоритме Рамалингама из операций над динамическими деревьями [19] используются только операция $link(u, v, T, Desc)$ и ее вариация $linkAll(u, X, T, Desc)$, которые мы приведем ниже. Отметим, что при реализации на графическом ускорителе цикл **while** можно вычислять одновременно для всех столбцов.

procedure $link(u, v: integer; Var T: Table, Var Desc: Table)$

Var

X, Y :Slice;
w :Word;
k :integer;

begin

X:=COL(u,T); X(v):=1; COL(u,T):=X;

w:=ROW(u,Desc);

X:=COL(v,Desc);

while SOME(w) **do**

begin

k:=STEP(w);

Y:=COL(k,Desc);

Y:=Y or X;

COL(k,Desc):=Y;

end;

end;

Поскольку в ассоциативных алгоритмах у выбранной вершины параллельно обрабатываются все исходящие ребра, то предпочтительно использовать *пакетную перестройку поддеревьев*: несколько поддеревьев подвешивается к одному корню за один проход.

```

procedure linkAll( u: integer; X:Slice; Var T:Table, Var Desc: Table)
Var
Y,Z,Z1:Slice;
w      :Word;
k:integer;
begin
    Y:=COL(u,T); Y:=Y or X; COL(u,T):=Y;
    w:=ROW(u,Desc);
    Y:=X; CLR(Z);
    while SOME(Y) do
        begin
            k:=STEP(Y);
            Z1:=COL(k,Desc);
            Z:=Z or Z1;
        end;
    while SOME(w) do
        begin
            k:=STEP(w);
            Y:=COL(k,Desc);
            Y:=Y or Z;
            COL(k,Desc):=Y;
        end;
end;

```

4. Представление алгоритма Рамалингама на STAR-машине

Для представления алгоритма Рамалингама на STAR-машине будем использовать следующую структуру данных: матрица смежности G размером $n \times n$, каждый i -й столбец которой хранит головы дуг, выходящих из вершины i ; вершина s – источник; остовное дерево ST (Spanning Tree) задается матрицей размером $n \times n$, каждый i -й столбец которой хранит головы дуг, выходящих из вершины i и принадлежащих остовному дереву; матрица потомков $Desc$ размером $n \times n$, в i -м столбце которой отмечены '1' потомки вершины i ; вспомогательный слайс $WorkSet$. На каждой текущей итерации этот слайс хранит те недостижимые вершины исходного графа, которые являются головами дуг, выходящих из текущей обрабатываемой вершины.

Замечание. Заметим, что в s -м столбце матрицы $Desc$ отмечены '1' все вершины, которые достижимы в остовном дереве из вершины s . Обозначим этот столбец как слайс RV (Reachable Vertices).

4.1. Ассоциативная версия алгоритма Рамалингама

Опишем содержательное выполнение алгоритма Рамалингама на STAR-машине. Вначале по остовному дереву ST , заданному на множестве вершин, достижимых из выделенной вершины s , запоминаем в слайс, скажем Z , множество недостижимых вершин. Пусть дуга (i, j) добавляется в граф G . Если вершина i достижима в G , а вершина j недостижима, то добавляем вершину j в $WorkSet$, объявляем ее достижимой и добавляем дугу (i, j) в остовное дерево ST (с помощью операции $link(i, j, ST, Desc)$). Затем выбираем из множества $WorkSet$ обрабатываемую вершину, скажем k . С помощью слайса, скажем X , запоминаем те вершины, недостижимые в G , в которые заходят дуги из вершины k . Объявляем эти вершины достижимыми, добавляем их в $WorkSet$ и добавляем соответствующие дуги в остовное дерево ST (с помощью операции $linkAll(u, X, ST, Desc)$). Аналогично обрабатываются все вершины, хранящиеся в слайсе $WorkSet$, при этом обработанная вершина удаляется из слайса. Процесс завершается, когда слайс $WorkSet$ становится пустым, т.е. когда в остовное дерево ST нельзя добавить новые дуги. На рис. 3. показано остовное дерево на множестве достижимых вершин до и после добавления дуги $(1, 5)$. Двойной чертой отмечены вершины, попадающие в множество $WorkSet$.

Рис. 3. Остовное дерево на множестве достижимых вершин до и после добавления дуги (1, 5).

Двойной чертой отмечены вершины, попадающие в множество WorkSet

Fig. 3. A spanning tree on the set of reachable vertices before and after adding an arc (1, 5).

Double lines mark the vertices that fall into the WorkSet

procedure InsertEdgeReachability(i, j, s : integer; Var $G, ST, Desc$: Table)

Var

WorkSet, X, RV, Z: Slice ;

Begin

1. CLR(WorkSet);

 2. RV:=COL($s, Desc$);

// В слайсе Z отмечены недостижимые вершины.

3. Z:= not RV;

 // Дуга (i, j) добавляется к графу G .

 4. X:=COL(i, G); X(j):=1; COL(i, G):=X;

 // Проверяем, что вершина i достижима, а j недостижима.

 5. **if** (RV(i)=1) and (RV(j)=0) **then**

 6. **begin**

 // Вершина j добавляется в множество WorkSet

 7. WorkSet(j):=1;

 // Вершина j удаляется из множества недостижимых вершин.

 8. Z(j):=0;

 // Обновляются матрицы ST и $Desc$

 9. link($i, j, ST, Desc$);

 10. **while** SOME(WorkSet) **do**

 11. **begin**

 // Выбирается текущая обрабатываемая вершина k .

 12. k :=STEP(WorkSet);

 // В слайс X заносятся те недостижимые вершины, в которые заходят дуги из вершины k .

 13. X:=COL(k, G);

14. X:= X and Z;

 15. **if** SOME(X) **then**

 16. **begin**

// Эти вершины добавляются в множество WorkSet.

17. WorkSet:= WorkSet or X;

// Эти вершины удаляются из множества недостижимых вершин.

18. Z:= Z and not X ;

 // Матрицы ST и $Desc$ обновляются.

 19. linkAll($k, X, ST, Desc$);

 20. **end**;

 21. **end**;

 22. **end**;

End;

Утверждение. Пусть задан ориентированный граф G , заданы множество достижимых вершин RV , остовное дерево ST , построенное на множестве достижимых вершин, и матрица потомков $Desc$. Пусть дуга (i, j) добавляется к графу G . Тогда после выполнения процедуры *InsertEdgeReachability* дерево ST будет расширено до остовного на обновленном множестве достижимых вершин, и обновлена матрица потомков $Desc$.

Базис индукции. Доказываем для случая, когда *WorkSet* состоит из одной вершины, и эта вершина является тупиковой. Пусть дуга (i, j) добавлена в граф G . Тогда после выполнения процедуры *InsertEdgeReachability* дуга будет добавлена к дереву ST , обновлена матрица потомков $Desc$ (вершина j добавлена в поддереву с корнем в вершине i и в поддеревья всех предков вершины i).

Шаг индукции. Пусть утверждение справедливо для множества *WorkSet*, состоящего из k вершин. Рассмотрим случай, когда *WorkSet* содержит $k + 1$ вершину и k вершин уже обработаны. Это означает, что новые достижимые вершины добавлены в множества *RV* и *WorkSet*, соответствующие дуги добавлены в дерево *ST*, и соответствующим образом изменена матрица *Desc*.

Из доказательства следует, что сложность процедуры оценивается как $O(k)$, где k – число тех вершин, которые стали достижимыми в результате добавления новой дуги.

Сравним выполнение последовательного алгоритма Рамалингама с выполнением его ассоциативной версии. Напомним, что в последовательном алгоритме множество *WorkSet* хранит множество обрабатываемых дуг, в то время как в параллельной версии слайс *WorkSet* хранит только те вершины, недостижимые в G , в которые заходят дуги из текущей добавленной вершины.

Fig. 4. The spanning tree on the set of reachable vertices after adding an arc (1, 5).

92

Перечислим основные достоинства ассоциативной версии. В последовательном алгоритме после добавления новой вершины в множество достижимых вершин и добавлении соответствующей дуги в остовное дерево (строки 8–9) необходимо *последовательно* добавить в множество *WorkSet* все дуги, выходящие из данной вершины (цикл в строках 10–11). В ассоциативной версии при обработке вершины из слайса *WorkSet*, скажем u , *одновременно* определяются те недостижимые вершины, в которые заходят дуги из вершины u . Эти вершины *одновременно* добавляются в слайс *WorkSet* (строка 17) и объявляются достижимыми (строка 18), а соответствующие дуги добавляются в дерево *ST*, и соответствующим образом обновляется матрица Desc (строка 19). Очевидно, что это приводит к существенному уменьшению числа итераций (рисунок 4).

5. Тестирование инкрементального алгоритма на графических ускорителях

Как уже упоминалось, для тестирования и использования ассоциативных алгоритмов была построена реализация STAR-машины на графических ускорителях. С помощью этой реализации представленный алгоритм тестировался на случайных графах с нормальным распределением ребер (степень вершин варьировала от 2 до 20) и различным числом достижимых вершин (общее количество вершин до 5 000, число дуг до 90 000).

Тестирование проводилось на видеокарте NVIDIA GEFORCE 920m. В ходе тестирования время работы динамического ассоциативного алгоритма (D_A) сравнилось со временем работы статического ассоциативного алгоритма (S_A), а также подсчитывались изменение числа достижимых вершин после добавления дуги и число дуг, исходящих из вершин, которые стали достижимыми после добавления дуги. Отметим, что эти величины определяют число итераций ассоциативного (D_A) и последовательного (D_S) динамических алгоритмов, соответственно.

Как видно из рис. 5, время работы статического алгоритма зависит от мощности обновленного множества достижимых вершин ($|VRout|$), а время работы динамического алгоритма линейно зависит от изменения мощности множества достижимых вершин ($|VRout| - |VRin|$). Отметим, что динамический алгоритм выполняется до 10 раз быстрее, чем статический.

Рис. 5. Сравнение времени работы динамического ассоциативного алгоритма со временем работы статического ассоциативного алгоритма

Fig. 5. The ratio of the runtime of the dynamic associative algorithm to the runtime of the static associative algorithm

На рис. 6 приводится количество итераций, которое требуется для обработки заданного графа после добавления новой дуги как ассоциативным динамическим алгоритмом, так и последовательным динамическим алгоритмом в зависимости от мощности множества достижимых вершин. Из этого рисунка видно, что число итераций для динамического ассоциативного алгоритма и динамического последовательного алгоритма могут значительно различаться в зависимости от структуры обрабатываемого графа.

Рис. 6. Количество итераций ($|WorkSet|$), необходимых для обработки графа после добавления дуги, заходящей в вершину v
 Fig. 6. The number of iterations ($|WorkSets|$) that are necessary for processing the graph after adding an arc going to the vertex v

Полученное ускорение зависит от усредненной степени вершин графа и исходного множества достижимых вершин.

Заключение

В статье приводится ассоциативная версия алгоритма Рамалингама для динамической обработки множества достижимых вершин в потоковых графах с выделенным источником после добавления новой дуги. Этот алгоритм использует простую структуру данных, которая позволяет выполнять доступ к данным по содержимому памяти. Параллельная реализация упомянутого алгоритма Рамалингама на STAR-машине представлена в виде процедуры *InsertEdgeReachability*, корректность которой доказана. В работе также перечисляются основные достоинства ассоциативной версии. Результаты тестирования на графических ускорителях показывают, что ассоциативный динамический алгоритм выполняется до десяти раз быстрее статического ассоциативного алгоритма, а также делает значительно меньше итераций, чем последовательный динамический алгоритм.

Благодарность. Авторы статьи выражают благодарность рецензенту за полезные замечания.

ЛИТЕРАТУРА

1. Repts T. Program analysis via graph reachability // Information and software technology, 1998. V. 40, № 11. P. 701–726.
2. Zhang Q., Su Z. Context-sensitive data-dependence analysis via linear conjunctive language reachability // POPL 2017: Proc. of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages. 2017. P. 344–358.
3. Hanauer K., Henzinger M., Schulz Ch. Fully Dynamic Single-Source Reachability in Practice: an Experimental Study // Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments, ALENEX, Salt Lake City, 2020. P. 106–119.
4. Jin R., Hong H., Wang H., Ruan N., Xiang Y. Computing Label-Constraint Reachability in Graph Databases // Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. New York: Association for Computing Machinery, 2010. P. 123–134.
5. Potter J.L. Associative Computing: a Programming Paradigm for Massively Parallel Computers. Boston : Perseus Publishing, 1991. 304 p.
6. Yavits L., Kvatinsky S., Morad A., Ginosar R. Resistive associative processor // IEEE Computer Architecture Letters. 2015. V. 14, № 2. P. 148–151.
7. Nedaa A. A hardware fast tracker for the ATLAS trigger // Physics of Particles and Nuclei Letters. 2016. V. 13, № 5. P. 527–531.
8. Nepomniaschaya A.S., 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.
9. Nepomniaschaya A.Sh. Basic associative parallel algorithms for vertical processing systems // Bulletin of the Novosibirsk Computing Center. Series: Comp. Science. 2009. № 9. P. 63–77.
10. 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.
11. 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.
12. 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.

13. Nepomniaschaya A.Sh. Efficient parallel implementation of the Ramalingam decremental algorithm for updating the shortest paths subgraph // *Computing and Informatics*. 2013. V. 32. P. 331–354.
14. Nepomniaschaya A.Sh. Associative version of the Ramalingam decremental algorithm for the dynamic all-pairs shortest-path problem // *Bulletin of the Novosibirsk Computing Center. Series: Comp. Science*. 2016. № 39. P. 37–50.
15. Nepomniaschaya A.Sh. Associative version of the Ramalingam incremental algorithm for the dynamic all-pairs shortest-path problem // *Bulletin of the Novosibirsk Computing Center. Series: Comp. Science*. 2016. № 40. P. 75–86.
16. Foster C.C. Content Addressable Parallel Processors. New York : John Wiley & Sons, 1976. 233 p.
17. Непомнящая А.Ш., Владыко М.А. Сравнение моделей ассоциативного вычисления // *Программирование*. 1997. № 6. С. 41–50.
18. Ramalingam G. Bounded Incremental Computation. 1996. V. 1098. P. 149–155.
19. Sleator D.D., Tarjan R.E. A Data Structure for Dynamic Trees // *Journal of Computer and System Sciences*. 1983. V. 26. P. 362–391.

Поступила в редакцию 18 марта 2020 г.

Nepomniaschaya A.Sh., Snytnikova T.V. (2021) ASSOCIATIVE VERSION OF THE RAMALINGAM INCREMENTAL ALGORITHM FOR THE DYNAMIC SINGLE-SOURCE REACHABILITY PROBLEM. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika*. [Tomsk State University Journal of Control and Computer Science]. 54. pp. 86–96

DOI: 10.17223/19988605/54/11

The paper provides a parallel implementation of the Ramalingam sequential algorithm for dynamically processing a graph with a single source after inserting a new arc. For this purpose, a model of associative (content-addressable) processors of the SIMD type with vertical data processing is used. The study of associative data processing began in the 60s. But studying the associative processing has never taken off, because only a limited amount of memory could be placed on a single die. However, progress in the computer industry and semiconductor technology in recent years has made the associative processing attractive again. Such an architecture best suits for solving problems of non-numeric data processing. In particular, this includes relational algebra, graph theory, seismic data processing. The main advantages of this architecture include data parallelism at the basic level, the use of two-dimensional tables as a data structure, and a massively parallel search by content. To model of vertical processing systems the STAR-machine is used. The work of STAR-machine is given by the language STAR. This language contains a group of elementary operations for processing of bit columns, as well as a library of standard procedures for efficient processing of matrix data. Every associative algorithm is represented as a procedure written in the language STAR. Since common associative architecture does not exist yet, there is a problem of the efficient implementing the STAR-machine on real appropriate SIMD architecture. Therefore, the STAR machine was efficiently implemented on graphics accelerators.

The Ramalingam algorithm receives as input the graph G , the set of reachable vertices from the source s , the dynamic spanning tree T , and the arc (u, v) inserted to the graph. After performing the Ramalingam algorithm the tree T is updated to the spanning tree on the updated set of reachable vertices. At the beginning, the Ramalingam sequential algorithm adds a new arc (u, v) to graph G . If the vertex u is reachable in the graph G and the vertex v is unreachable, then the arc (u, v) is added to the tree T and the vertex v becomes reachable. After that, all arcs outgoing from vertex v are added to the set Work Sets of processed arcs. Then each arc from this set is updated by analogy with the described method.

To implement the Ramalingam algorithm on the STAR-machine, a special data structure is built, that allows one to access data by content. With a parallel implementation of this algorithm, the set of WorkSets was represented as a bit column. At each current iteration, the unreachable vertices entering the processing vertex are included into the Workset. These vertices become reachable, and the corresponding arcs are simultaneously added to the tree T . Note that in the *sequential algorithm*, the set WorkSet stores a set of processed arcs. While in the associative version, the slice WorkSet stores only unreachable vertices entering the processing vertex. The correctness of this associative version was proved.

Using the implementation of the STAR-machine on the GPU, this algorithm was tested on the NVIDIA GEFORCE 920m on graphs with various numbers of reachable vertices (total number of vertices up to 5000, number of arcs up to 90,000). Test results show that the associative dynamic algorithm runs up to ten times faster than the static associative algorithm. Moreover, it performs significantly less iterations than the sequential dynamic algorithm.

Keywords: SIMD; GPU; content-addressable memory; flowgraph; dynamic algorithm.

NEPOMNIASCHAYA Anna Shmylevna (Candidate of Physical and Mathematics, Senior Researcher, Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russian Federation).
E-mail: anep@ssd.sccc.ru

SNYTNIKOVA Tatyana Valentinovna (Junior Researcher, Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russian Federation).
E-mail: snytnikovat@ssd.sccc.ru

REFERENCES

1. Reps, T. (1998) Program analysis via graph reachability. *Information and software technology*. 40(11). pp. 701–726.
2. Zhang, Q. & Su, Z. (2017) Context-sensitive data-dependence analysis via linear conjunctive language reachability. *POPL 2017: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. pp. 344–358. DOI: 10.1145/3093333.3009848
3. Hanauer, K., Henzinger, M. & Schulz, Ch. (2020) Fully Dynamic Single-Source Reachability in Practice: An Experimental Study. *Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments*. ALENEX, Salt Lake City. pp. 106–119. DOI: 10.1137/1.9781611976007.9
4. Jin, R., Hong, H., Wang, H., Ruan, N. & Xiang, Y. (2010) Computing Label-Constraint Reachability in Graph Databases. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. New York: Association for Computing Machinery. pp. 123–134. DOI: 10.1145/1807167.1807183
5. Potter, J.L. (1991) *Associative Computing: a Programming Paradigm for Massively Parallel Computers*. Boston: Perseus Publishing.
6. Yavits, L., Kvatinisky, S., Morad, A. & Ginosar R. (2015) Resistive associative processor. *IEEE Computer Architecture Letters*. 14(2). pp. 148–151. DOI: 10.1109/LCA.2014.2374597
7. Nedaa, A. (2016) A hardware fast tracker for the ATLAS trigger. *Physics of Particles and Nuclei Letters*. 13(5). pp. 527–531. DOI: 10.1134/S1547477116050368
8. Nepomniaschaya, A.Sh. & Dvoskina, M.A. (2000) A simple implementation of Dijkstra’s shortest path algorithm on associative parallel processors. *Fundamenta Informaticae. IOS Press*. 43. pp. 227–243. DOI: 10.3233/FI-2000-43123412
9. Nepomniaschaya, A.Sh. (2009) Basic associative parallel algorithms for vertical processing systems. *Bulletin of the Novosibirsk Computing Center. Series: Computer Science*. 9. pp. 63–77.
10. Snytnikova, T.V. & Nepomniaschaya, A.Sh. (2016) Solution of graph problems by means of the STAR-machine being implemented on GPUs. *Prikladnaya diskretnaya matematika – Applied Discrete Mathematics*. 3(33). pp. 98–115. DOI: 10.17223/20710410/33/9
11. Snytnikova, T.V. (2018) GPU Implementation of the associative computing model on GPU: the basic procedure library of the language STAR. *Novye vychislitel'nye tekhnologii*. 19(1). pp. 85–95.
12. Nepomniaschaya, A.Sh. (2012) Associative version of Ramalingam algorithm for dynamically updating the shortest-path subgraph after inserting a new edge into a graph. *Kibernetika i sistemnyy analiz – Cybernetics and Systems Analysis*. 48. pp. 358–368. DOI: 10.1007/s10559-012-9415-x
13. Nepomniaschaya, A.Sh. (2013) Efficient parallel implementation of the Ramalingam decremental algorithm for updating the shortest paths subgraph. *Computing and Informatics*. 32. pp. 331–354.
14. Nepomniaschaya, A.Sh. (2016a) Associative version of the Ramalingam decremental algorithm for the dynamic all-pairs shortest-path problem. *Bulletin of the Novosibirsk Computing Center. Series: Comp. Science*. 39. pp. 37–50. DOI: 10.31144/bncc.cs.2542-1972.2016.n40.p75-86
15. Nepomniaschaya, A.Sh. (2016b) Associative version of the Ramalingam incremental algorithm for the dynamic all-pairs shortest-path problem. *Bulletin of the Novosibirsk Computing Center. Series: Comp. Science*. no. 40. pp. 75–86. DOI: 10.31144/bncc.cs.2542-1972.2016.n40.p75-86
16. Foster, C.C. (1976) *Content Addressable Parallel Processors*. New York: John Wiley & Sons.
17. Nepomniaschaya, A.Sh. & Vladyko, M.A. (1997) Comparison of associative calculation models. *Programmirovaniye – Programming and Computer Software*. 6. pp. 41–50.
18. Ramalingam, G. (1996) *Bounded Incremental Computation*. 1098. pp.149–155.
19. Sleator, D.D. & Tarjan, R.E. (1983) A Data Structure for Dynamic Trees. *Journal of Computer and System Sciences*. 26. pp. 362–391. DOI: 10.1016/0022-0000(83)90006-5