Наука и Образование
МГТУ им. Н.Э. Баумана
Сетевое научное издание
ISSN 1994-0408

Ссылка на статью:
// Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2015. № 10. С. 453–472.
DOI: **10.7463/1015.0820736**
Представлена в редакцию: 08.09.2015
Исправлена: 09.11.2015

УДК 004.2+004.31

Реализация алгоритмов обхода графа в вычислительной системе с многими потоками команд и одним потоком данных

Попов А. Ю.[1,*]

[*]alexpopov@bmstu.ru

[1]МГТУ им. Н.Э. Баумана, Москва, Россия

В ходе проекта, реализуемого на кафедре «Компьютерные системы и сети» МГТУ им. Н.Э. Баумана, разработана вычислительная система с многими потоками команд и одним потоком данных, в которой реализованы новые архитектурные принципы обработки множеств и структур данных. В рамках проекта создано новое устройство — процессор обработки структур, которое существенно повышает производительность вычислительной системы при решении задач дискретной оптимизации. Принципы работы новой системы отличаются от известных, что приводит к необходимости модифицировать существующие алгоритмы и методики разработки программ. В предыдущих работах приведены общие принципы организации вычислительного процесса в МКОД-системе, представлена структура и особенности работы процессора обработки структур, показаны общие принципы решения задач дискретной оптимизации на графах. В данной работе рассмотрены два алгоритма: алгоритм поиска в ширину и алгоритм поиска в глубину. Исследуются варианты реализации алгоритмов для двух режимов работы: режима сопроцессора и режима МКОД. Приводятся результаты экспериментального сравнения производительности МКОД системы с универсальными ЭВМ.

Ключевые слова: граф; много потоков команд и один поток данных; процессор обработки структур; алгоритм поиска в ширину; алгоритм поиска в глубину; обход графа

Введение

Статья посвящена очередному этапу создания научных основ функционирования принципиально новой архитектура вычислительной системы с многими потоками команд и одним потоком данных (МКОД), разработка которой ведется в МГТУ им. Н.Э. Баумана. Новая архитектура основана на глубокой аппаратной поддержке операций дискретной математики и параллельной обработке множеств и структур данных. В рамках проекта создано новое устройство — процессор обработки структур (СП), которое существенно повышает производительность вычислительной системы при решении задач дискретной оптимизации. Принципы работы новой системы отличаются от известных, что привело к необходимости модификации существующих алгоритмов и методик разработки программ.

В предыдущих работах приведены общие принципы организации вычислительного процесса в МКОД-системе, представлена структура и особенности работы процессора обработки структур, показаны общие принципы решения задач дискретной оптимизации на графах. В данной работе рассмотрены алгоритмы поиска в ширину и в глубину, которые часто используются в составе других алгоритмов оптимизации на графах. Исследуются варианты реализации алгоритмов для двух режимов работы СП: режима сопроцессора и режима МКОД. Приводятся результаты экспериментального сравнения производительности МКОД-системы с универсальными ЭВМ.

1. Вычислительная система с многими потоками команд и одним потоком данных

Набор команд современных микропроцессоров состоит из сотен инструкций, однако лишь малая их часть составляет команды непосредственной обработки чисел: команды арифметической и логической обработки, сдвиги, функция синуса и некоторые другие [1, 2, 3]. Эти команды реализуют операции над числами лишь для ограниченных разделов математики: арифметики, логики и тригонометрии. При этом большинство исполняемых команд в вычислительном потоке не связано с вычислениями напрямую, а носят сопутствующий характер. Это такие типы команд, как: команды пересылки данных, ветвления, команды управления. Указанное противоречие говорит о недостаточной эффективности современных средств вычислительной техники и необходимости повышения процента команд обработки.

Анализ кода программ показывает, что команды пересылки составляют около 50% потока инструкций, обрабатываемых процессором [4, 5]. Другие команды, такие как команды поиска минимума/максимума в блоке векторной обработки (SSE, Streaming SIMD Extensions) закрывают лишь незначительную часть потребностей по аппаратной поддержке вычислительных алгоритмов. Вместе с тем при решении практических задач на ЭВМ используется существенно большее количество математических методов и операций, включая операции над множествами в дискретной математике, методы математической логики, дифференциального и интегрального исчисления, линейной алгебры, аналитической геометрии, математической статистики. Однако очевидно, что их реализация требует существенного изменения в структуре электронных вычислительных машин и является крайне трудоемкой. В ряде работ отмечается, что при использовании специальных аппаратных средств на основе ПЛИС FPGA удается достичь существенного ускорения вычислительного процесса [6, 7, 8, 9, 10].

В [11, 12, 13] дано определение структур данных, являющихся представлением множеств в памяти ЭВМ: структура данных представляют собой два взаимосвязанных множества: множество данных (информационную составляющую), а также множество отношений (структурную составляющую). В практике программирования в ссылочных структурах (списках, деревьях) структурная составляющая представлена в виде ссылок на ячейки памяти. В векторных структурах логика отношений заложена в алгоритмы операций над струк-

турой и определяется непосредственным расположением элементов множества в памяти [14, 15]. Таким образом, структура данных является целостным представлением какой-либо информации в памяти ЭВМ при помощи отношений структурного порядка на множестве ячеек памяти. Обработка двух видов информации (данных и их структурных отношений) может вестись независимо, что и использовано в системе, аппаратно реализующей обработку структур данных и множеств.

В ходе проекта, проводимого в МГТУ им. Н.Э. Баумана были разработаны новые принципы построения вычислительных систем, которые позволяют увеличить количество совмещаемых операций при обработке множеств и структур данных [12]. Для этого в архитектуру вычислительной системы были внесены изменения, был разработан специализированный блок обработки множеств и структур данных: процессор обработки структур. Этот устройство обрабатывает лишь ту часть информации, которая определяет взаимные отношения хранимых данных, т.е. структурную часть структур данных (рис. 1). В [11, 12, 13] показано, что такая система относится к классу ЭВМ с многими потоками команд и одним потоком данных. Примеров разработки и широкого применения универсальных вычислительных средств с подобной архитектурой до данного проекта не имеется.

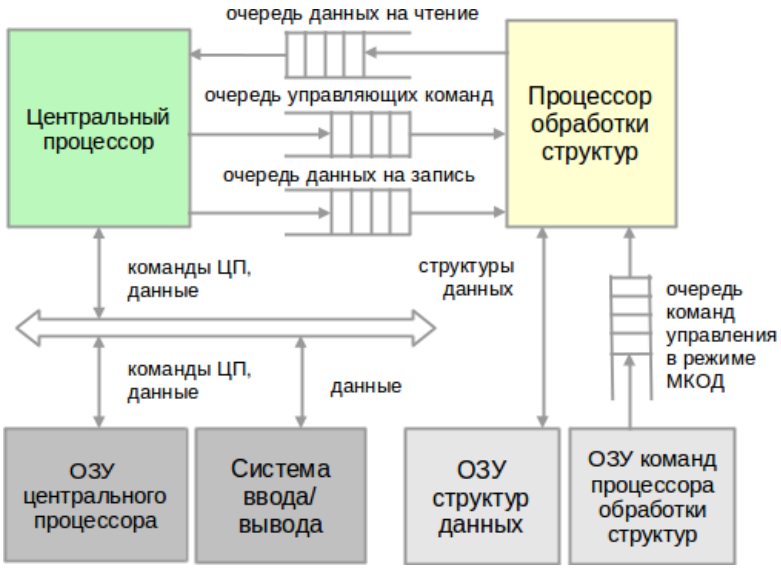

Рис. 1. Архитектура вычислительной системы с многими потоками команд и одним потоком данных

Применение структур данных востребовано при программировании в таких важных приложениях, как: базы данных, робототехника, операционные системы, САПР, научные и инженерные задачи оптимизации. Поэтому была начата разработка широкого круга специальных алгоритмов, эффективно решаемых в такой системе. В [16, 17, 18, 19] приведены примеры таких алгоритмов. В [18] приведена методика разработки и модификации алгоритмов для модели функционировании МКОД-системы. В работах [16, 17] приведены результаты проектирования МКОД и тестирования системы.

2. Варианты представления графа в МКОД-системе

Примеры представления графов в адресной памяти в виде матриц инциденций, матриц смежности, векторов Айлифа или списков смежных вершин широко известны благодаря большому количеству трудов, посвященных данной тематике [14, 15]. Однако представленные в этих работах варианты опираются на модель адресной памяти, в которой возможно разместить данные непосредственно по смежным адресам (матрицы) или организовать структуру данных с помощью указателей (списки). Основным же отличием алгоритмов в МКОД-системе является представление действий в виде операций над структурами данных, в которых сохраняются данные в соответствии с их ключами. Так как функции хранения и обработки структур данных реализованы аппаратно в СП, программная модель представления данных существенно отличается от адресной: используется модель ассоциативного представления в виде пар «ключ-значение» (модель «ключ-значение»). Именно поэтому первым этапом методики преобразования алгоритмов [17, 18] является представление информационных моделей алгоритма в модели «ключ-значение». Для задач оптимизации на графах важным является способ представления исходного графа, в связи с чем рассмотрим варианты представления взвешенного графа (как неориентированного, так и ориентированного).

Представление графа $G(V, E)$ списком смежных вершин. Пусть в алгоритме требуется выполнить проход по вершинам графа, используя связывающие их ребра. Для этой цели потребуется выполнить перебор элементов множества смежных вершин $v \in Adj[u]$, и последующий переход к одной из них. Так как степень вершин различна, требуется также хранить количество исходящих ребер count. Поле G.KEY хранит номера вершин u и порядковый номер ребра. Поле данных G.DATA хранит номер инцидентной вершины v и вес ребра c, как показано в табл. 1.

Таблица 1

Пример представления графа $G(V, E)$ списком смежных вершин (G.KEY$[u, i]$, G.DATA$[v, c]$)

G.KEY	G.DATA
u,0	count
u,1	v_1,c
...	...
u,count	v_{count},c

Представление графа $G(V, E)$ списком инцидентных ребер. Если в алгоритме требуется поиск ребер, соединяющих вершины (u, v), граф может быть представлен другим образом. Поле G.KEY в этом случае составляется из номера вершины u и v, а поле данных G.DATA хранит вес ребра c. Чтобы упростить обход всех ребер, инцидентных вершине,

в структуру могут быть добавлены специальные маркеры с нулевым индексом вершины v. Это позволит найти стартовую позицию в структуре данных в так называемой индексной записи $G.\text{KEY}(u, 0)$ за которой будут располагаться остальные записи, получить которые можно простым обходом соседних вершин по команде NEXT (табл. 2).

Таблица 2

Пример представления графа $G(V, E)$ списком инцидентных ребер ($G.$KEY$[u, v]$, $G.$DATA$[c]$)

G.KEY	G.DATA
u,0	0
u,v	c

Представление графа $G(V, E)$ упорядоченным списком инцидентных ребер. Часто требуется хранить граф таким образом, чтобы множество ребер было упорядочено по их весу: минимальный ключ должен принадлежать ребру с наименьшим весом. Так как связность в общем случае не является уникальной и в графе могут присутствовать несколько ребер с одинаковым весом, следует использовать более сложный составной ключ. В старшей части ключа должен храниться вес ребра c, а в младшей будут храниться номера вершин (u, v), т.е. поле $G.\text{KEY} = (c, u, v)$. Поле $G.$DATA может оставаться пустым, так как необходимая информация об инцидентности вершин и ребер имеется в составном ключе (табл. 3. Однако в алгоритме может возникнуть необходимость хранить дополнительные данные (флаги, переменные и пр.).

Таблица 3

Пример представления графа $G(V, E)$ упорядоченным списком инцидентных ребер ($G.$KEY$[c, u, v]$, $G.$DATA$[]$)

G.KEY	G.DATA
c,u,v	

В зависимости от выполняемых в алгоритме действий возможно использование как одного варианта представления, так и несколько вариантов одновременно.

3. Алгоритм поиска в ширину

Алгоритмы обхода графа применяются в более сложных алгоритмах в качестве основы для перебора множества вершин, достижимых из некоторой начальной вершины. Наиболее распространенными алгоритмами обхода является поиск в ширину и поиск в глубину.

Задача ставится следующим образом. Пусть необходимо найти вершину графа, обеспечивающую выполнение некоторого условия оптимальности, для чего следует совершить

обход всех его вершин. Алгоритм поиска в ширину состоит в построении множества вершин, достигнутых из начальной вершины. Для каждой из них определяется список смежных вершин, не просмотренных в алгоритме ранее. Найденные таким образом вершины включаются в множество, которое служит для хранения последовательности обхода графа, в то время как текущая вершина исключается из множества.

Особенностью алгоритма поиска в ширину является то, что порядок рассмотрения вершин графа соответствует последовательности их добавления в множество. Это означает, что на первой итерации рассматриваются все вершины, связанные с начальной. На второй итерации рассматриваются все вершины, связанные с вершинами из первой итерации и так далее.

Как было отмечено ранее, алгоритмы обхода графа применяются в других алгоритмах оптимизации и мало используются в качестве самостоятельных алгоритмов (например, в работе [18] алгоритм поиска в ширину был использован в составе алгоритма Форда-Фалкерсона). Поэтому целесообразно рассматривать реализацию таких алгоритмов с учетом возможных вариантов представления графа в прикладных алгоритмах оптимизации.

Поясним сказанное следующими доводами. При реализации алгоритмов обхода графа в универсальной ЭВМ с адресной моделью памяти исходный граф может быть задан в виде списков смежных вершин или матрицы смежности[15]. Первый вариант предпочтителен с точки зрения объема памяти, занимаемого структурой графа, в то время как второй вариант обеспечивает меньшую вычислительную сложность при необходимости определения смежности двух вершин. Судить об эффективности выбора тех или иных структур данных для алгоритмов обхода графа, работающих в универсальных ЭВМ можно только на основании набора операций всего прикладного алгоритма и оценки его вычислительной сложности.

В работах [16, 17, 18] отмечалось, что процессор обработки структур использует аппаратную реализацию (B+)-деревьев для представления пар «ключ-значение» в локальной памяти структур (см. рис. 1). Как показывают исследования, проведенные ранее для алгоритмов Дейкстры и Форда —- Фалкерсона, реализация графа на основе сильно ветвящегося (B+)-дерева может быть оправдана следующими причинами:

- оценка вычислительной сложности прикладного алгоритма, включающего действия по обходу графа, оказывается ниже, если граф представлен с помощью (B+)-дерева;
- емкостная сложность алгоритма при использовании (B+)-деревьев оказывается ниже (например, в сравнении с матрицей смежности);
- структура (B+)-дерева может обрабатываться во многих случаях параллельно, в отличии от списков;
- временная сложность реализации алгоритма при использовании (B+)-дерева с большой степенью ветвления вершин может мало отличаться от линейной (например, графики экспериментов в [17] близки к линейным).

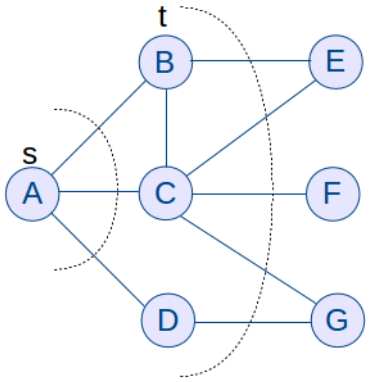

Рис. 2. Пример обхода графа алгоритмом поиска в ширину.
Последовательность обхода вершин: A, B, C, D, E, F, G

На примере графа, представленного на рис. 2, алгоритм поиска в ширину обеспечивает следующую последовательность обхода вершин: A; B; C; D; E; F; G. Пусть задан граф $G(u, v)$ в котором нужно достичь каждой вершины, начиная с вершины s. Можно описать алгоритм поиска в ширину следующими действиями:

1. *Множество* $= s$.
2. Выбрать некоторый элемент t из *Множества*.
3. Выполнить действия над t (в зависимости от действий в алгоритме). Если условие поиска соблюдается, то конец алгоритма.
4. Вершину t графа G отметить, как рассмотренную. Удалить вершину t из *Множества*.
5. Добавить все вершины, связанные ребрами с t и не являющиеся рассмотренными на предыдущих шагах алгоритма, в *Следующее Множество*.
6. Если *Множество* не пусто, то перейти к пункту 2. Иначе к пункту 7.
7. Если *Следующее Множество* вершин пусто, то конец алгоритма (все вершины достигнуты). Иначе переход на следующий уровень: перейти к пункту 2 и рассматривать *Следующее Множество* как *Множество*.

Так как при обходе графа требуется лишь чтение списка смежных вершин, такой вариант обеспечивает вычислительную сложность $O(|V| + |E|)$ [15]. Оценим вычислительную сложность алгоритма при использовании (B+)-деревьев. Вычислительная сложность операций добавления, удаления, поиска для такого рода деревьев составляет $O(\log(n))$, что дает большую вычислительную сложность. Алгоритм предполагает добавление всех вершин графа в *Множество*, что требует $O(|V|\log(|V|))$ операций. Для проверки условия в п. 5 требуется снова выполнить обращение за $O(\log(|V|))$ к каждой инцидентной вершине, что выполняется не более $2|E|$ раз. Таким образом, общая вычислительная сложность решения увеличивается в $\log(|V|)$ и составляет $O(|V|\log(|V|)+|E|\log(|V|)) = O((|V|+|E|)\log(|V|))$.

Обычно для хранения *Множества* и *Следующего Множества* используется единая структура очереди, что обеспечивает просмотр всех вершин одного уровня до перехода на следующий. Очередность рассмотрения вершин на текущем уровне существенного зна-

чения не имеет, однако может рассматриваться вариант выборки вершины с минимальным значением целевой функции (например, накопленной длины пути от исходной вершины). Для реализации структуры очереди Q в СП необходимо определить алгоритм формирования ключей и соответствующих им значений, обеспечивающих процедуру обслуживания FIFO. Для этого будем использовать порядковый номер вершины в порядке обхода дерева. Индекс вершины будем хранить в поле значения: $Q.\text{KEY} = (i)$, $Q.\text{DATA} = (u)$.

Структура G должна обеспечивать поиск всех вершин, инцидентных указанной, а также определение и изменение флага для пройденной вершины. Для алгоритма поиска в ширину может быть использован вариант 1 представления графа в виде списка смежных вершин. Флаг пройденной вершины может быть сохранен в записи, выделенной для описания мощности множества смежных вершин, т.е. для $G.\text{KEY} = (u, 0)$ поле $G.\text{DATA} = (|Adj[t]|, flag)$. Для всех остальных записей при $G.\text{KEY} = (u, i)$, $i = 1 \ldots |Adj[t]|$ поле данных содержит индекс смежной вершины $G.\text{DATA} = (v)$. Приведем псевдокод обхода графа с помощью поиска в ширину на основе выбранных структур. В начальный момент структура Q пуста, а структура G содержит списки смежных вершин и для всех вершин поле $flag = 0$.

Алгоритм 1. ПОИСК В ШИРИНУ(G,s) /псевдокод алгоритма в ОКОД-системе/

```
1:  i = 0                                          ▷ Индекс позиции вершины в порядке обхода
2:  INSERT(Q, i, s)                                ▷ Добавление начальной вершины s
3:  ПОВТОРЯТЬ
4:      t ← SEARCH(G, (MIN(Q).DATA, 0))            ▷ Читаем информацию об очередной вершине
5:      A ← t.KEY.|Adj[t]|                         ▷ Определяем количество смежных с t вершин
6:      ЦИКЛ j = 1 to A                            ▷ Для всех смежных вершин
7:          v ← SEARCH(G, (t.KEY, j))              ▷ Читаем информацию об очередной вершине v, смежной с t
8:          ЕСЛИ v.DATA.flag == 0 ТО               ▷ Если вершина v еще не пройдена
9:              i = i + 1                          ▷ Увеличиваем индекс позиции
10:             INSERT(Q, i, v.KEY)                ▷ Добавление новой вершины v
11:             INSERT(G, (v, 0), (v.DATA.|Adj[t]|, 1))   ▷ Запись флага flag = 1
12:         ВСЕ ЕСЛИ
13:     ВСЕ ЦИКЛ
14: ЦИКЛ ПОКА POWER(Q) ≠ 0                         ▷ Пока очередь не пуста
15: КОНЕЦ                                          ▷ Все вершины дерева пройдены
```

Поясним работу алгоритма. В строках 1 и 2 происходит инициализация индекса и очереди Q: в нее добавляется начальная вершина s. Далее в цикле в строках 3–14 выполняется выборка очередной вершины и добавление всех смежных с ней вершин. В строке 4 производится чтение информации t из графа (структуры G). Для этого производится обращение к очереди Q и выбирается запись на наименьшим ключом (индексом). Найденная запись в поле DATA содержит индекс вершины, по которому можно обратиться в структуру G. Ключ $(\text{MIN}(Q).\text{DATA}, 0)$ позволяет обратиться к записи, содержащей поля

$|Adj[t]|$ (мощность множества смежных вершин) и $flag$ (флаг, указывающий на то, что вершина не была ранее рассмотрена). Далее в цикле в строках 6–13 для каждой смежной вершины производится проверка флага $(v.\text{DATA}.flag)$, и если флаг равен нулю, то вершина рассматривается впервые. В этом случае увеличивается индекс записи в очереди (строка 9). Далее, в очередь добавляется новая вершина с индексом i и полем данных, равным индексу вершины $v(v.\text{KEY})$. После этого в строке 11 происходит изменение флага на единицу: $\text{INSERT}(G,(v,0),(v.\text{DATA}.|Adj[t]|,1))$.

Методика преобразования структурных конструкций в коды центрального процессора и процессора обработки структур обсуждалась в [18]. Рассмотрим вариант реализации алгоритма в МКОД-системе.

Алгоритм 2. ПОИСК В ШИРИНУ(G,s) /псевдокод алгоритма в МКОД-системе/

```
1:  –Поток ЦП–                                          ▷ –Поток СП–
2:  i=0
3:  PUT(i)
4:  PUT(s)                                              ▷ INSERT(Q,?,?)
5:  ПОВТОРЯТЬ
6:      PUT(1)                                          ▷ JT(?,@1)
7:      GET(temp)                                       ▷ @1:MIN(Q)*
8:      PUT((temp.DATA, 0))
9:      GET(t)                                          ▷ SEARCH(G,?)*
10:     A ← t.KEY.|Adj[t]|
11:     ЦИКЛ j = 1 to A
12:         PUT(0)                                      ▷ @2:JT(?,@3)
13:         PUT(t.KEY, j)
14:         GET(v)                                      ▷ SEARCH(G,?)*
15:         ЕСЛИ v.DATA.flag == 0 ТО
16:             PUT(0)                                  ▷ JT(?,@4)
17:             i = i + 1
18:             PUT(i)
19:             PUT(v.KEY)                              ▷ INSERT(Q,?,?)
20:             PUT(v.KEY, 0)
21:             PUT(v.DATA.|Adj[t]|, 1)                 ▷ INSERT(G,?,?)*
22:         ИНАЧЕ
23:             PUT(1)
24:         ВСЕ ЕСЛИ
25:     ВСЕ ЦИКЛ                                        ▷ @4:JT(1,@2)*
26:     PUT(1)
27: ЦИКЛ ПОКА GET() ≠ 0                                 ▷ @3:POWER(Q)*
28: PUT(0)                                              ▷ JT(?,@1)
29: КОНЕЦ
```

Алгоритм работы СП составлен таким образом, что для синхронизации двух потоков не происходит передача адресов переходов, а передаются лишь теги для команд условных переходов (см. набор команд в [17, 18]). Цикл ПОКА с постусловием преобразован в два потока команд: строки 5, 6, 27, 28. В строках 3 и 4 происходит передача тегов для выполнения команды вставки вершины в очередь Q (INSERT$(Q, ?, ?)$). При проходе тела цикла на команду в строке 6 $JT(?, @1)$ поступает тег 1, что позволяет перейти СП на следующую команду с меткой @1. Таким образом тело цикла ПОКА с постусловием выполняется хотя бы один раз. После проверки условия в строке 27 (GET$() \neq 0$), в коде ЦП либо происходит возврат в начало цикла и на команду в строке 28 $JT(?, @1)$ поступает снова тег 1. В противном случае, когда происходит выход из цикла, передается тег 0. Это приводит к переходу из команды JT к следующей команде и завершению программы.

В строках 11, 12, 25, 26 организован счетный цикл. Если ЦП осуществил вход в тело цикла, передается тег 0 в строке 12 в команду СП с меткой @2: $JT(?, @3)$. При этом происходит переход на следующую команду SEARCH. В конце цикла происходит безусловный возврат назад на метку @2: $JT(1, @2)$. Если же цикл завершился, ЦП переходит к команде 26, где передает тег перехода 1. Это приводит в команде JT с меткой @2 к переходу на метку @3 и выходу СП из цикла. Условный оператор в строках 15, 16, 22, 23, 24 организован аналогичным образом, но возврата к началу оператора не происходит. Вместо этого, если условие в строке 15 не выполняется, происходит переход по альтернативной ветви на метку @4, находящуюся непосредственно за циклом.

При оптимизации алгоритма возможно сократить зависимости потоков ЦП и СП. Операторы, отмеченные знаком (*), используют операнды повторно. Оператор в строке 9 (SEARCH(G,?)) использует тег $temp$.DATA, который ранее был получен в команде MIN(Q) в строке 7. При сохранении этого операнда во внутреннем регистре устройства загрузки команд СП, данная команда может быть запущена на исполнение сразу за предыдущей. Аналогичная ситуация наблюдается и с оператором t.KEY в строке 14, а также с оператором вставки в строке 21. Последний требует формирования составного ключа на основе поля данных. Таким образом, из семи операторов обработки структур данных пять (70%) могут выполняться независимо.

4. Алгоритм поиска в глубину

Алгоритм поиска в глубину использует иную стратегию: в первую очередь рассматриваются вершины, смежные с последней рассмотренной. Для новой рассмотренной вершины выполняются необходимые действия, после чего переходят уже к ее смежным вершинам. Таким образом, в алгоритм обеспечивает проход вглубь графа или дерева, что используется при поиске начального решения в деревьях поиска (алгоритм Диница, алгоритмы разрезания гиперграфа и пр.).

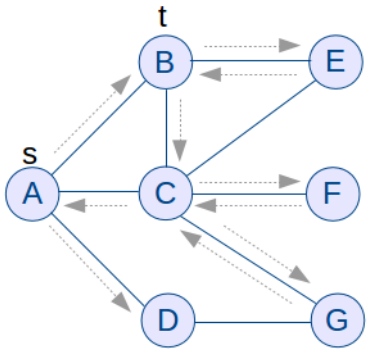

Рис. 3. Пример обхода графа алгоритмом поиска в глубину.
Последовательность обхода вершин: A, B, E, C, F, G, D

На примере графа на рис. 3, алгоритм обеспечивает последовательность обхода вершин: A, B, E, C, F, G, D. Алгоритм поиска в глубину реализует следующие действия.

1. *Множество* $= s$.

2. Выбрать некоторый элемент t из *Множества*.

3. Выполнить действия над t (в зависимости от действий в алгоритме). Если условие поиска соблюдается, то конец алгоритма.

4. Вершину t графа G отметить, как рассмотренную. Удалить вершину t из *Множества*.

5. Добавить все вершины, связанные ребрами с t, и не являющиеся рассмотренными на предыдущих шагах алгоритма, в *Множество*.

6. Если *Множество* не пусто, то перейти к пункту 2. Иначе конец алгоритма (все вершины достигнуты).

Отличие этого алгоритма от алгоритма поиска в ширину заключается в порядке выборки вершин из Множества. Если для алгоритма поиска в ширину использовалась процедура обслуживания First In First Out (может быть реализовано с помощью очереди), то в алгоритма поиска в глубину должна обеспечиваться процедура обслуживания Last In First Out, (стек). Таким образом, алгоритмы отличаются структурой данных, представляющей множество вершин. Общая вычислительная сложность алгоритма при использовании списка смежных вершин составляет $O(|V| + |E|)$, а для (B+)-дерева: $O((|V| + |E|)\log(|V|))$.

Для организации стека может быть использован аналогичный принцип хранения номера пройденной вершины в виде ключа. Тогда для извлечения последней помещенной вершины можно использовать операцию MAX(Q) в строке 4. Код алгоритма в МКОД-системе в режиме сопроцессора представлен ниже.

Алгоритм 3. ПОИСК В ГЛУБИНУ(G,s) /псевдокод алгоритма в ОКОД системе/

1: $i = 0$ ▷ Индекс позиции вершины в порядке обхода
2: INSERT(Q, i, s) ▷ Добавление начальной вершины s
3: **ПОВТОРЯТЬ**
4: $t \leftarrow$ SEARCH$(G, (\textbf{MAX(Q)}.\text{DATA}, 0))$ ▷ Читаем информацию об очередной вершине
5: $A \leftarrow t.\text{KEY}.|Adj[t]|$ ▷ Определяем количество смежных с t вершин

```
6:    ЦИКЛ j = 1 to A                                  ▷ Для всех смежных вершин
7:        v ← SEARCH(G, (t.KEY, j))   ▷ Читаем информацию об очередной вершине v, смежной с t
8:        ЕСЛИ v.DATA.flag == 0 ТО                     ▷ Если вершина v еще не пройдена
9:            i = i + 1                                ▷ Увеличиваем индекс позиции
10:           INSERT(Q, i, v.KEY)                      ▷ Добавление новой вершины v
11:           INSERT(G, (v, 0), (v.DATA.|Adj[t]|, 1))  ▷ Запись флага flag = 1
12:       ВСЕ ЕСЛИ
13:   ВСЕ ЦИКЛ
14: ЦИКЛ ПОКА POWER(Q) ≠ 0                             ▷ Пока очередь не пуста
15: КОНЕЦ                                              ▷ Все вершины дерева пройдены
```

При разделении одного потока команд на два потока в МКОД-системе отличие также состоит только в строке 7 (алгоритм представлен ниже).

Алгоритм 4. ПОИСК В ГЛУБИНУ(G,s) /псевдокод алгоритма в МКОД-системе/

```
1: –Поток ЦП–                                          ▷ –Поток СП–
2: i = 0
3: PUT(i)
4: PUT(s)                                              ▷ INSERT(Q,?,?)
5: ПОВТОРЯТЬ
6:    PUT(1)                                           ▷ JT(?,@1)
7:    GET(temp)                                        ▷ @1:MAX(Q)*
8:    PUT((temp.DATA, 0))
9:    GET(t)                                           ▷ SEARCH(G,?)*
10:   A ← t.KEY.|Adj[t]|
11:   ЦИКЛ j = 1 to A
12:       PUT(0)                                       ▷ @2:JT(?,@3)
13:       PUT(t.KEY, j)
14:       GET(v)                                       ▷ SEARCH(G,?)*
15:       ЕСЛИ v.DATA.flag == 0 ТО
16:           PUT(0)                                   ▷ JT(?,@4)
17:           i = i + 1
18:           PUT(i)
19:           PUT(v.KEY)                               ▷ INSERT(Q,?,?)
20:           PUT(v.KEY, 0)
21:           PUT(v.DATA.|Adj[t]|, 1)                  ▷ INSERT(G,?,?)*
22:       ИНАЧЕ
23:           PUT(1)
24:       ВСЕ ЕСЛИ
25:   ВСЕ ЦИКЛ                                         ▷ @4:JT(1,@2)*
26:   PUT(1)
27: ЦИКЛ ПОКА GET() ≠ 0                                ▷ @3:POWER(Q)*
28: PUT(0)                                             ▷ JT(?,@1)
29: КОНЕЦ
```

Заметим, что в ряде источников используется рекурсивный вариант алгоритма поиска в глубину, позволяющий представить указанные действия более лаконично [15]. Негативным свойством рекурсивной организации алгоритмов является передача параметров и адресов возврата через стек, что в скрытой форме увеличивает емкостную сложность алгоритмов. Как и для других архитектур, для МКОД-системы в случае рекурсии будет использован стек в оперативной памяти центрального процессора. Поэтому в тех случаях, когда возможно обойтись без использования рекурсии, целесообразно это делать и для МКОД-системы.

Аналогично алгоритму поиска в ширину обстоит дело и с зависимостями между командами в потоке ЦП и потоке СП: 70% команд являются независимыми. Указанная оптимизация возможна только при наличии в составе СП блока специальных регистров, хранящих операнды команд или результаты обработки структур данных. В настоящее время указанный блок разрабатывается и будет представлен в следующей версии процессора обработки структур.

5. Исследование производительности процессора обработки структур

Разработанные алгоритмы были реализованы в МКОД-системе и было исследовано количество тактов их работы. С целью выявления эффективности алгоритмов управления СП была разработана программная реализация (B+)-дерева для ЭВМ на основе универсальных процессоров с микроархитектурой x86 и ARM11 (рис. 4). Для сравнения результатов измерений были использованы ЭВМ со следующими параметрами:

• **ЭВМ с многими потоками команд и одним потоком данных (МКОД)**: разрядность поля ключа и данных — 32 бита; максимальное количество ключей в структуре — $2 * 10^6$; количество уровней аппаратного (B+)-дерева — 8; количество вершин на одном уровне — 8; локальная память — 256 МБ, DDR; ширина шины памяти — 64 бит; частота шины памяти —

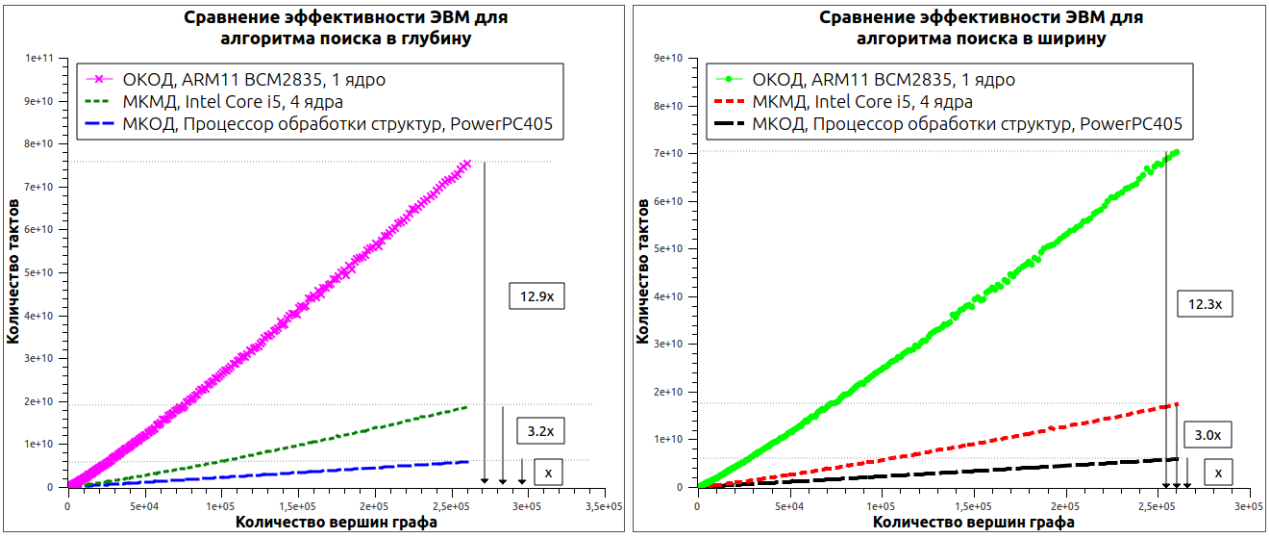

Рис. 4. Исследование производительности процессора обработки структур на алгоритмах обхода графа

100 МГц; частота СП — 100 МГц, ПЛИС FPGA Virtex II Pro; режим работы СП — режим сопроцессора;

• **ЭВМ с многими потоками команд и многими потоками данных (МКМД)**: объем оперативной памяти — 4 ГБ, DDR3; количество процессорных ядер — 4, core i5; частота процессоров — 2530 МГц; операционная система — Ubuntu Linux 12.04 i386; компилятор — c99; ширина шины памяти — 64 бит; частота шины памяти — 1333 МГц;

• **ЭВМ с одним потоком команд и одним потоком данных (ОКОД)**: объем оперативной памяти — 512 МБ, DDR3; количество процессорных ядер — 1, ARM11 Broadcom 2835; частота процессоров — 700 МГц; операционная система — Raspbian; компилятор — gcc; ширина шины памяти — 64 бит; частота шины памяти — 400 МГц.

Следует отметить, что объем оперативной памяти СП в 16 раз меньше объема оперативной памяти ЭВМ с процессором Intel. Поэтому в экспериментах количество элементов в графе было ограничено и изменялось в пределах от 400 до 260000 вершин. Учитывая то, что средняя мощность множества $Adj[u]$ составляет 8 (т.е. каждой вершине инциденты восемь ребер) максимальное количество записей в структуре приближается к 2 миллионам. СП должен также хранить структуру Q, содержащую до 260000 записей. Таким образом, указанное количество вершин позволяет избежать существенной загрузки памяти структур СП, при которой начинают проявляться краевые эффекты в работе с памятью: фрагментация, автоматическое сжатие, сдвиги вершин. Подробное описание способа хранения информации в памяти структур СП и проблем максимального заполнения представлены в предыдущих работах [16, 17]. Ниже показаны результаты измерения количества тактов, затрачиваемых на выполнение обоих версий алгоритма обхода графа для двух экспериментальных ЭВМ (рис. 5).

При программной реализации алгоритмов в МКМД системе используется функция выделения памяти malloc(), которая обращается к менеджеру памяти операционной системы, функционирующему независимо от тестовой программы. В связи с этим, в полученных результатах для ЭВМ МКМД не учитываются влияние параллельного выполнения кода менеджера памяти на разных ядрах и работа менеджера памяти до и после кода эксперимента.

Эксперименты показали, что аппаратная эффективность МКОД-системы в режиме сопроцессора в 3 раза выше аппаратной эффективности ЭВМ на многоядерном микропроцессоре Intel Core i5 и в 12 раз эффективнее ЭВМ на микропроцессоре ARM11. Для СП разница во времени работы двух вариантов алгоритма обхода оказалась незначительной (0,2%), в то время как в универсальной системе разница составила 6% для микропроцессора ARM11 (меньшее время достигнуто для алгоритма поиска в ширину) и 7% для микропроцессора Intel Core i5. При этом аппаратная сложность МКОД примерно в 800 раз меньше [17]. Применение же режима МКОД позволит еще больше увеличить ускорение за счет большей независимости команд СП и ЦП. В настоящее время ведется проектирование следующей версии процессора обработки структур, что позволит провести испытания системы при максимальной независимости потоков.

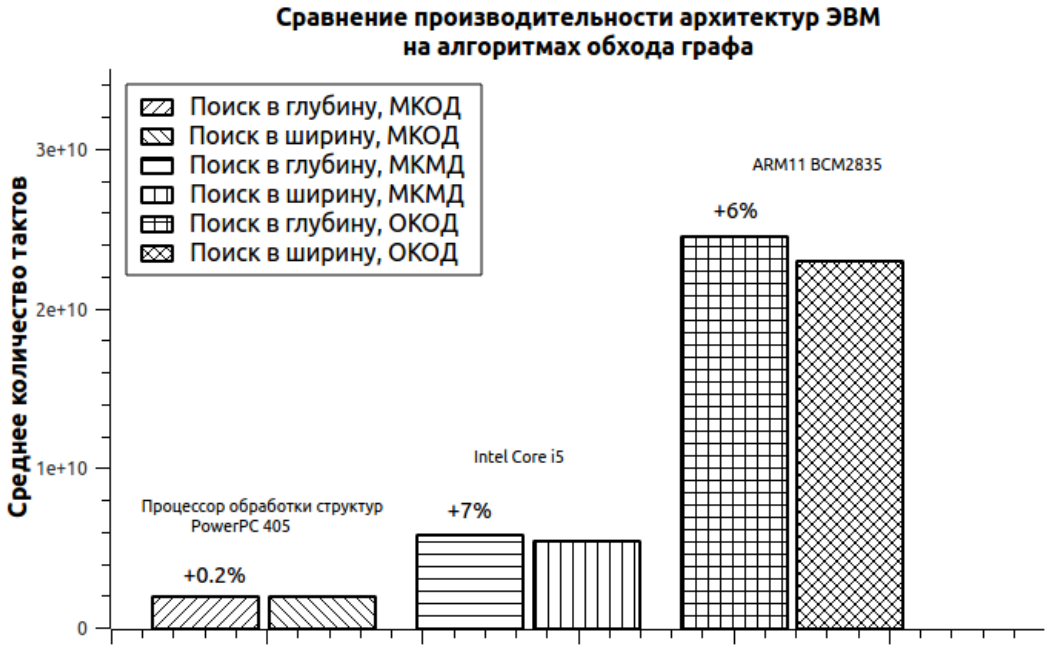

Рис. 5. Сравнение эффективности ЭВМ с различной архитектурой на алгоритмах обхода графа

Заключение

Цель представления алгоритма для МКОД-системы — перенос ряда трудоемких действий алгоритма из центрального процессора в Процессор обработки структур. Именно представление алгоритма в базисе операций СП позволяет осуществить разделение потока команд на два потока, возможность чего и доказывается в алгоритмах 1 и 3. Так как это успешно удалось, это означает, что МКОД вычислительная система может быть использована для решения задач обхода графа.

Предложенные варианты представления графов в виде структур данных позволяют выполнять обращение к информации с такими запросами, как:

- определение количества ребер, инцидентных вершине и выборка номеров вершин, смежных с вершиной (вариант 1);
- определение всех ребер, инцидентных вершине (вариант 2);
- выборка ребер графа в порядке увеличения их весов (вариант 3).

В рассмотренных алгоритмах обхода графа по методу поиска в ширину и в глубину был использован вариант 1 представления графа, однако в алгоритмах оптимизации [17, 18] могут использоваться и другие варианты.

Реализация полученных алгоритмов и их экспериментальное исследование позволяют сделать вывод, что МКОД-система способна решать задачи оптимизации на графах за меньшее количество тактов: на обход графа затрачено примерно в 3 раза меньше тактов по сравнению с ЭВМ на микропроцессоре Intel и в 12 раз меньше, чем для ЭВМ с микропроцессором ARM11. Однако, такое аппаратное не всегда оказывается достаточным для

получения ускорения по времени, так как тактовая частота МКОД в 25 раз меньше тактовой частоты микропроцессора Intel Core i5. Поэтому целесообразно применять все возможные способы дальнейшего совершенствования архитектуры МКОД, а также вести разработку СП на более совершенной элементной базе.

Список литературы

1. Орлов С.А., Цилькер Б.Я. Организация ЭВМ и систем: Учебник для вузов. 2-е изд. СПб.: Питер, 2011. 688 с.
2. Harris D.M., Harris S.L. Digital Design and Computer Architecture. 2nd Ed. Boston: Morgan Kaufmann, 2012. 721 p.
3. Таненбаум Э.С., Остин Т. Архитектура компьютера: пер. с англ. 6-е изд. СПб.: Питер, 2015. 816 с. [Tanenbaum A., Austin T. Structured computer organization. 6th ed. Prentice Hall, 2012. 800 p.].
4. Huang I.J., Peng T.C. Analysis of x86 instruction set usage for DOS/Windows applications and its implication on superscalar design // IEICE Transactions on Information and Systems. 2002. Vol E85-D, no. 6, P. 929–939.
5. Kankowski P. x86 Machine Code Statistics // strchr.com: website. Available at: http://www.strchr.com/x86_machine_code_statistics, accessed 01.09.2015.
6. Coole J., Stitt G. Traversal Caches: A Framework for FPGA Acceleration of Pointer Data Structures // International Journal of Reconfigurable Computing. 2010. Vol 2010. Art ID 652620. DOI: 10.1155/2010/652620
7. Williams J., Massie Ch., George A.D., Richardson J., Gosrani K., Lam H. 2010. Characterization of Fixed and Reconfigurable Multi-Core Devices for Application Acceleration // Transactions on Reconfigurable Technology and Systems. 2010. Vol. 3, no. 4. Art. no. 19. DOI: 10.1145/1862648.1862649
8. Integrated Cryptographic and Compression Accelerators on Intel Architecture Platforms. SOLUTION BRIEF. Intel QuickAssist Technology. Order No. 329879-001US. Intel Corporation, 2013. 5 p. Available at: http://www.intel.ru/content/dam/www/public/us/en/documents/solution-briefs/integrated-cryptographic-compression-accelerators-brief.pdf, accessed 01.09.2015.
9. Kumar Sn., Shriraman A., Srinivasan V., Lin D., Phillips J. SQRL: hardware accelerator for collecting software data structures // Proceedings of the 23rd international conference on Parallel architectures and compilation (PACT'14). ACM, New York, NY, USA, 2014. P. 475–476. DOI: 10.1145/2628071.2628118
10. Singh M., Leonhardi B. Introduction to the IBM Netezza warehouse appliance // In Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research

(CASCON'11) / ed. by M. Litoiu, E. Stroulia, S. MacKay. IBM Corp., Riverton, NJ, USA, 2011. P. 385–386.

11. Попов А.Ю. Электронная вычислительная машина с многими потоками команд и одним потоком данных: пат. 71016 Российская Федерация. 2008. Бюл. № 5. 1 с.
12. Попов А.Ю. Реализация электронной вычислительной машины с аппаратной поддержкой операций над структурами данных // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. 2011. Спец. вып. Информационные технологии и компьютерные системы, С. 83–87.
13. Попов А.Ю. Электронная вычислительная машина с аппаратной поддержкой операций над структурами данных // Аэрокосмические технологии: тр. Второй Междунар. научно-техн. конф., посвященной 95-летию со дня рождения академика В.Н. Челомея (РФ, Москва–Реутов, 19–20 мая 2009 г.). Т. 1 / ОАО «ВПК «НПО машиностроения»; МГТУ им. Н.Э. Баумана. М.: МГТУ им. Н.Э. Баумана, 2009. С. 296–301.
14. Кнут Д. Искусство программирования. В 3 т. Т. 3. Сортировка и поиск: пер с англ. 2-е изд. М.: Вильямс, 2000. 832 с.
15. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ: пер. с англ. М.: МЦНМО, 2000. 960 с.
16. Попов А.Ю. Применение вычислительных систем с многими потоками команд и одним потоком данных для решения задач оптимизации // Инженерный журнал: наука и инновации. 2012. № 1. Режим доступа: http://engjournal.ru/catalog/it/hidden/80.html (дата обращения 01.09.2015.)
17. Попов А.Ю. Исследование производительности процессора обработки структур в системе с многими потоками команд и одним потоком данных // Инженерный журнал: наука и инновации, 2013, № 11. Режим доступа: http://engjournal.ru/catalog/it/hidden/1048.html (дата обращения 01.09.2015).
18. Попов А.Ю. О реализации алгоритма Форда-Фалкерсона в вычислительной системе с многими потоками команд и одним потоком данных // Наука и образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2014. № 9. С. 162–180. DOI: 10.7463/0914.0726416
19. Подольский В.Э. Об организации параллельной работы некоторых алгоритмов поиска кратчайшего пути на графе в вычислительной системе с многими потоками команд и одним потоком данных // Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2015. № 4. DOI: 10.7463/0415.0764268

Science&Education of the Bauman MSTU
Electronic journal
ISSN 1994-0408

Science and Education of the Bauman MSTU, 2015, no. 10, pp. 453–472.
DOI: **10.7463/1015.0820736**
Received: 08.09.2015
Revised: 09.11.2015

The implementation of graph traversal algorithms on the Multiple Instruction and Single Data stream computer system

Popov A. Yu.[1,*]

[*]alexpopov@bmstu.ru

[1]Bauman Moscow State Technical University, Russia

Keywords: graph, MISD computer system, structure processor, depth-first search, breadth-first search, graph traversal

The network- and graph-based optimization algorithms are widely used in solving practical problems. However, with the large-scale introduction of information technologies in human activities there become compounded requirements for volumes of input data and retrieval rate of finding a solution. Despite the fact that to date, are researched and implemented a large number of algorithms for different models of computers and computer systems, the solution of key problems of optimization for real dimension of the problem remains difficult. Therefore the search for new and more efficient computing structures, as well as modification of the known algorithms are relevant.

The use of data structures is demanded for programming in such important applications as databases, robotics, operating systems, CAD, scientific and engineering optimization problems. Thus, we started developing a wide range of special algorithms, effectively solved in such a system. In previous works there are examples of such algorithms, the technique of the development and modification of algorithms to model the MISD system operation, and the results of MISD design and system testing.

The paper considers two algorithms: a depth-first search algorithm and a breadth-first search one. It studies the options of implementation algorithms for two modes: a coprocessor mode and a MISD mode. The paper also presents the results of experimentally compared performance of MISD system and mainframes.

References

1. Orlov S.A., Tsil'ker B.Ya. *Organizatsiya EVM i sistem* [Organization of computers and systems]. St. Petersburg, Piter Publ., 2011. 688 p. (in Russian).
2. Harris D.M., Harris S.L. *Digital Design and Computer Architecture*. 2nd ed. Boston, Morgan Kaufmann, 2012. 721 p.

3. Tanenbaum A., Austin T. *Structured computer organization*. 6th ed. Prentice Hall, 2012. 800 p. (Russ. ed.: Tanenbaum A., Austin T. *Arkhitektura komp'yutera*. St. Petersburg, Piter Publ., 2012. 816 p.).
4. Huang I.J., Peng T.C. Analysis of x86 instruction set usage for DOS/Windows applications and its implication on superscalar design. *IEICE Transactions on Information and Systems*, 2002, vol. E85-D, no. 6, pp. 929–939.
5. Kankowski P. *x86 Machine Code Statistics*. strchr.com: website. Available at: http://www.strchr.com/x86_machine_code_statistics, accessed 01.09.2015.
6. Coole J., Stitt G. Traversal Caches: A Framework for FPGA Acceleration of Pointer Data Structures. *International Journal of Reconfigurable Computing*, 2010, vol. 2010, art. ID 652620. DOI: 10.1155/2010/652620
7. Williams J., Massie Ch., George A.D., Richardson J., Gosrani K., Lam H. 2010. Characterization of Fixed and Reconfigurable Multi-Core Devices for Application Acceleration. *ACM Transactions on Reconfigurable Technology and Systems*, 2010, vol. 3, no. 4, art. no. 19. DOI: 10.1145/1862648.1862649
8. Integrated Cryptographic and Compression Accelerators on Intel Architecture Platforms. SOLUTION BRIEF. Intel QuickAssist Technology. Order No. 329879-001US. Intel Corporation, 2013. 5 p. Available at: http://www.intel.ru/content/dam/www/public/us/en/documents/solution-briefs/integrated-cryptographic-compression-accelerators-brief.pdf, accessed 01.09.2015.
9. Kumar Sn., Shriraman A., Srinivasan V., Lin D., Phillips J. SQRL: hardware accelerator for collecting software data structures. *Proceedings of the 23rd international conference on Parallel architectures and compilation (PACT'14)*. ACM, New York, NY, USA, 2014, pp. 475–476. DOI: 10.1145/2628071.2628118
10. Singh M., Leonhardi B. Introduction to the IBM Netezza warehouse appliance. In: Litoiu M., Stroulia E., MacKay S., eds. *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research (CASCON'11)*. IBM Corp., Riverton, NJ, USA, 2011, pp. 385–386.
11. Popov A.Iu. *Elektronnaia vychislitel'naia mashina s mnogimi potokami komand i odnim potokom dannykh* [Electronic computer with multiple instruction streams and single data stream]. Patent RF, no. 71016, 2008. (in Russian).
12. Popov A.Iu. Implementation of an electronic computer with hardware support for operations on data structures. *Vestnik MGTU. Ser. Priborostroenie = Herald of the Bauman MSTU. Ser. Instrument Engineering*, 2011, Spec. iss. *Informatsionnye tekhnologii i komp'iuternye sistemy* [Information technology and computer systems], pp. 83–87. (in Russian).
13. Popov A.Iu. Electronic computer with hardware support for operations on data structures. *Aerokosmicheskie tekhnologii: tr. Vtoroi mezhdunarodnoi nauchno-tekhnicheskoi konferentsii,*

posviashchennoi 95-letiiu so dnia rozhdeniia akademika V.N. Chelomeia [Aerospace Technology: proceedings of the Second International Scientific and Technical Conference dedicated to the 95th anniversary of the birth of Academician V.N. Chelomei], RF, Reutov — Moscow, 19-20 May 2009, Vol. 1. Moscow, Bauman MSTU Publ., 2009, pp. 296–301. (in Russian).

14. Knuth D.E. *The Art of Computer Programming. Vol. 3. Sorting and Searching*. 2nd ed. Reading, Massachusetts, Addison-Wesley, 1998. (Russ. ed.: Knuth D.E. *Iskusstvo programmirovaniia. V 3 t. Vol. 3. Sortirovka i poisk*. Moscow, Vil'iams Publ., 2000. 832 p.).
15. Cormen T.H., Leiserson C.E., Rivest R.L. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990. (Russ. ed.: Cormen T.H., Leiserson C.E., Rivest R.L. *Algoritmy: postroenie i analiz*. Moscow, MTsNMO Publ., 2000. 960 p.).
16. Popov A.Iu. Application of computer systems with multiple instruction streams and single data stream for solving optimization problems. *Inzhenernyy zhurnal: nauka i innovatsii = Engineering Journal: Science and Innovation*, 2012, no. 1. Available at: http://engjournal.ru/catalog/it/hidden/80.html, accessed 01.09.2015. (in Russian).
17. Popov A.Iu. The study of the structure processor performance in the computer system with multiple-instruction streams and single-data stream. *Inzhenernyy zhurnal: nauka I innovatsii = Engineering Journal: Science and Innovation*, 2013, no. 11. Available at: http://engjournal.ru/catalog/it/hidden/1048.html, accessed 01.09.2015. (in Russian).
18. Popov A.Yu. On the implementation of the Ford-Fulkerson algorithm on the Multiple Instruction and Single Data computer system. *Nauka i obrazovanie MGTU im. N.E. Baumana = Science and Education of the Bauman MSTU*, 2014, no. 9, pp. 162–180. DOI: 10.7463/0914.0726416 (in Russian).
19. Podol'skii V.E. On the Organization of Parallel Operation of Some Algorithms for Finding the Shortest Path on a Graph on a Computer System with Multiple Instruction Stream and Single Data Stream. *Nauka i obrazovanie MGTU im. N.E. Baumana = Science and Education of the Bauman MSTU*, 2015, no. 4, pp. 189–214. DOI: 10.7463/0415.0764268 (in Russian).