

УДК 004.2; 004.31

О реализации алгоритма Форда — Фалкерсона в вычислительной системе с многими потоками команд и одним потоком данных

Попов А. Ю.^{1,*}

[* alexpopov@bmstu.ru](mailto:alexpopov@bmstu.ru)

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

В работе рассматривается реализация алгоритма поиска максимального потока на графе для разработанной в МГТУ им. Н.Э. Баумана вычислительной системы с многими потоками команд и одним потоком данных (МКОД). Ключевой особенностью данной архитектуры является глубокая аппаратная поддержка операций над множествами и структурами данных. Функции хранения и доступа к ним реализованы на специализированном процессоре обработки структур (СП). Преимуществом такой системы является возможность параллельного исполнения частей вычислительных задач, связанных с доступом к множествам структурам данных одновременно с арифметически-логической обработкой информации. Приведены форматы команд процессора обработки структур, предложена методика модификации алгоритмов при реализации в МКОД-системе, предложены варианты модификации архитектуры МКОД-системы, ведущие к повышению ее производительности.

Ключевые слова: максимальный поток; алгоритм Форда — Фалкерсона; много потоков команд и один поток данных; процессор обработки структур

Введение

Алгоритмы оптимизации на сетях и графах находят широкое применение при решении практических задач. Однако, вместе с масштабным внедрением информационных технологий в человеческую деятельность усугубляются требования к объемам входных данных и скорости поиска решения. Несмотря на то, что к настоящему времени исследованы и реализованы большое количество алгоритмов для различных моделей ЭВМ и вычислительных систем, решение ключевых задач оптимизации для реальных размерностей задач остается затруднительным. В связи с этим поиск новых и более эффективных вычислительных структур, а также модификация известных алгоритмов являются актуальными.

В работе рассматривается реализация алгоритма поиска максимального потока на графе для разработанной в МГТУ им. Н.Э.Баумана вычислительной системы с многими потоками команд и одним потоком данных (МКОД). Ключевой особенностью данной архитектуры

является глубокая аппаратная поддержка операций над множествами и структурами данных. Функции хранения и доступа к ним реализованы на специализированном процессоре обработки структур (СП), который способен на аппаратном уровне выполнять такие операции, как: добавление, удаление, поиск, пересечение, дополнение, объединение и другие. Преимуществом такой системы является возможность параллельного исполнения частей вычислительных задач, связанных с доступом к множествам структурам данных одновременно с арифметически-логической обработкой информации.

В предыдущих работах представлены общие принципы организации вычислительного процесса и особенности выполнения программ в МКОД системе, представлена структура и принципы функционирования процессора обработки структур, показаны общие принципы решения графовых задач в такой системе и проведены экспериментальные исследования эффективности полученных алгоритмов.

В данной работе приведены форматы команд процессора обработки структур, предложена методика модификации алгоритмов при реализации в МКОД системе, предложен вариант алгоритма Форда — Фалкерсона поиска максимального потока на графе для МКОД модели вычислительной системы, предложены варианты модификации архитектуры МКОД системы, ведущие к повышению ее производительности.

1. Принципы функционирования вычислительной системы с многими потоками команд и одним потоком данных

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. Для этого в архитектуру вычислительной системы были внесены существенные изменения, был разработан специализированный блок обработки множеств и структур данных: процессор обработки структур (рис. 1). Это устройство обрабатывает лишь ту часть информации, которая определяет взаимные отношения хранимых данных, т.е. структурную часть структур данных. Предпо-

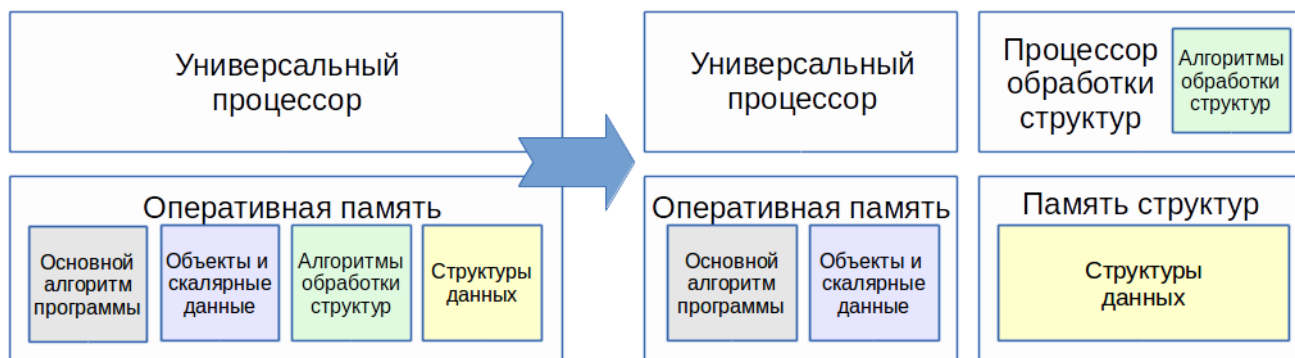


Рис. 1. Проект по созданию универсальной вычислительной системы с аппаратной поддержкой обработки множеств и структур данных

ложим, к примеру, что в алгоритме необходимо хранить множество ключей и соответствующих им значений. Тогда, если нужно выбрать элемент с ключом k , процессор обработки структур (СП) выполняет поиск и выборку элемента по структурной составляющей, после чего передает информационную составляющую (сами данные, ассоциированные с ключом k) центральному процессору. Таким образом, действия, исполняемые двумя процессорами, могут выполняться параллельно и независимо, а общий поток команд обработки данных для такой вычислительной системы разделяется на два потока команд: обработки информационной составляющей и обработки структурной составляющей структуры данных. Указанная вычислительная система, как показано в [1, 2, 3], относится к классу «много потоков команд и один поток данных». Примеров разработки и широкого применения ВС данного класса нет, так как до данного проекта не были определены архитектурные принципы, позволяющие применять такую систему для решения широкого круга задач. Применение же структур данных достаточно широко и универсально. Однако, реализация новых архитектурных принципов выявила несовместимость подходов к разработке алгоритмов на сетях и графах, принятых для общеизвестных вычислительных систем (например, систем с многими потоками команд и многими потоками данных), что привело к необходимости создания методики разработки или модификации алгоритмов других моделей вычислительных систем для модели МКОД.

При этом следует учитывать, что в МКОД системе предусмотрены два режима работы процессора обработки структур:

- **режим сопроцессора** — управляющие команды для обработки множеств и структур данных формируются в центральном процессоре системы и поступают через очередь управляющих команд (рис. 2);

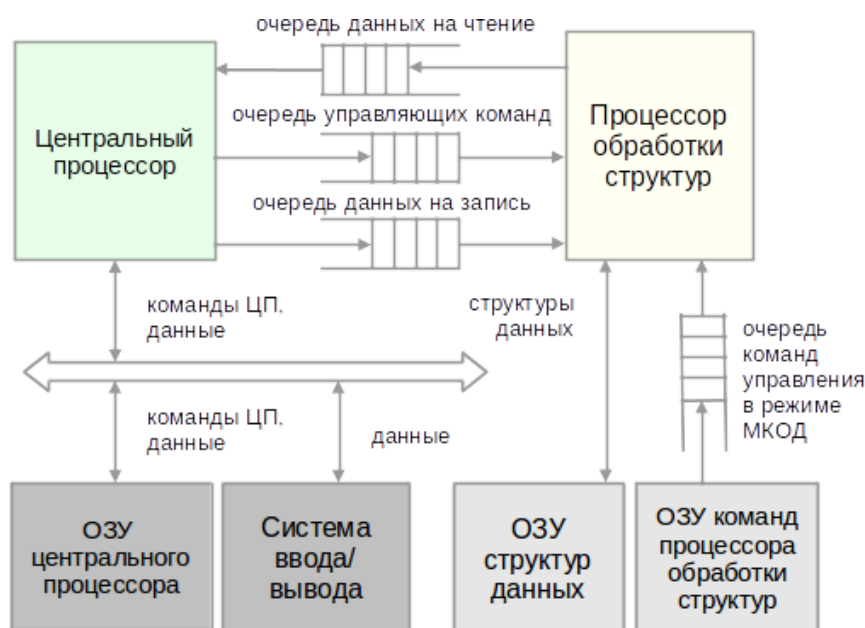


Рис. 2. Архитектура вычислительной системы с многими потоками команд и одним потоком данных

- **режим МКОД** — команды выбираются процессором обработки структур из локального ОЗУ команд. СП также имеет независимую память для хранения структур, что обеспечивает параллельность и независимость арифметически-логической обработки ЦП и обработки множеств и структур данных в СП.

Анализ способов повышения производительности для МКОД систем показывает, что как при доминирующей арифметической обработке, так и при доминирующей обработке структур данных, для увеличения производительности необходимо:

- сокращение времени обработки структур данных при аппаратной реализации в СП;
- сокращение временных затрат на приём и передачу данных;
- поиск алгоритмов с меньшими зависимостями между арифметической обработкой и обработкой структур данных.

Традиционное представление методов и алгоритмов решения задач дискретной оптимизации опирается на обобщенное представление о структуре и программной модели вычислительной машины фон Неймана с одним потоком команд и одним потоком данных (ОКОД). Однако, это оказывается недостаточно для реализации тех же алгоритмов в параллельных вычислительных системах, а также в разработанной МКОД системе. Основной задачей данного исследования является совершенствование методики модификации известных алгоритмов в их представлении, максимально учитывающее специфику исполнения программ в МКОД, опирающуюся на имеющиеся ресурсы и программную модель данной вычислительной системы. Подробные результаты проектирования МКОД и тестирования системы на примере алгоритма Дейкстры приведены в [4, 5].

2. Подходы к созданию методики модификации алгоритмов

Основным отличием между версиями алгоритмов, представленных в обобщенном виде (например, в псевдокоде), и для МКОД системы, является необходимость в последней представлять информацию в виде структур данных, а действия алгоритма в виде операций над этими структурами. При традиционном подходе применение структур данных оговаривается в ходе дополнительных исследований, хотя и имеет существенное влияние на вычислительную сложность алгоритмов. Так, например, в [6, 7] целесообразность выбора тех или иных структур данных приводит к возникновению новых модифицированных версий алгоритмов, позволяет снизить оценку вычислительной сложности. Исследование операций над структурами данных и их выбор для конкретных алгоритмов, проведенный в [8], показали, что подходы к применению структур данных опираются на небольшой набор команд, таких как: добавление, удаление, поиск, пересечение и т. д.. Именно эти команды и легли в основу набора машинных инструкций процессора обработки структур в МКОД системе. Таким образом, в МКОД системе нет необходимости производить выбор конкретных структур данных, так как их хранение и обработка уже реализована аппаратно. Ниже представлены основные этапы методики.

Этап 1. Представление информационных моделей алгоритма в виде структур данных. Как известно, структуры данных представляют собой абстрактное представление об отображении в оперативную память двух множеств: ключей и ассоциированных с ними данных. В некоторых случаях наличие ключей в структуре уже приобретает информативный характер и используется в алгоритмах. Представление алгоритма в виде операций над структурами данных требует принятия решений о количестве применяемых структур, а также о соответствии информации, используемой в алгоритме, ключам и данным структур. Указанный переход представляет процесс, аналогичный подготовке информации к хранению в реляционной базе данных. В частности, определяются составные ключевые поля, собранные в виде конкатенации данных.

Пример 1 (представление графа $G(V, E)$). Пусть в алгоритме требуется вести обход графа, например, методом поиска в глубину. Тогда основной операцией будет поиск вершин $v \in Adj[u]$, инцидентных указанной, и последующий переход к обработке всех связанным вершин. Но, поскольку 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.

Т а б л и ц а 1

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

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

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 (табл. 2). 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 по записи $G.KEY[u,0]$, за которой будут располагаться остальные записи, получить которые можно простым обходом соседних вершин по команде NEXT.

Т а б л и ц а 2

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

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

Этап 2. Представление алгоритма в базисе операций над структурами данных.

На данном этапе все действия алгоритма должны быть представлены в виде действий над wybranymi ранее структурами. В настоящий момент в МКОД системе реализовано 15 команд обработки множеств и структур данных (табл. 3).

Т а б л и ц а 3

Базовые операции над структурами данных

Команда	Мнемокод	Операнды*			Результаты*		
Поиск	SEARCH	R	K	–	C	K	3
Добавление	INSERT	R	K	3	C	K	3
Удаление	DELETE	R	K	–	C	K	3
Удаление структуры	DELSTR	R	–	–	C	–	–
Максимум	MAX	R	–	–	C	K	3
Минимум	MIN	R	–	–	C	K	3
Мощность	POWER	R	–	–	C	K	3
И	AND	R	A	B	C	–	–
ИЛИ	OR	R	A	B	C	–	–
НЕ	NOT	R	A	B	C	–	–
Срез LS	LS	R	A	K	C	–	–
Срез LSEQ	LSEQ	R	A	K	C	–	–
Срез GR	GR	R	A	K	C	–	–
Срез GREQ	GREQ	R	A	K	C	–	–
Следующий	NEXT	R	K	–	C	K	3
Переход по тегу	JT	Тег	Адрес	–	–	–	–

* R — номер структуры результата; A, B — номера исходных структур; K — ключ; 3 — значение; C — статус.

Все указанные команды изменяют регистр статуса, по которому можно определить, было ли выполнение команды успешным. В результате команд SEARCH, DELETE, MAX, MIN, NEXT в очередь данных попадают ключ и значение найденных записей (KEY, DATA), которые могут быть использованы ЦП в алгоритме. Операнды R, A и B являются номерами структур, над которыми выполняются команды: операнд R указывает на номер структуры, в которой будет сохранен результат; структуры A и B используются в И-ИЛИ-НЕ операциях и срезах в качестве исходных.

Результатом выполнения второго этапа методики является алгоритм, который может быть реализован в МКОД системе в режиме сопроцессора. В этом случае центральный процессор исполняет команды арифметически-логической обработки и передает команды обработки структур в очередь управляющих команд (см. рис. 2). Результаты исполнения команд извлекаются из «очереди данных на чтение».

Пример 2 (Выполнение базовых операций над структурами данных).

Поиск по ключу: $\text{Data} \leftarrow \text{SEARCH}(\text{G}, \text{Key})$;

Добавление: $\text{INSERT}(\text{G}, \text{Key}, \text{Data})$;

Объединение структур: $\text{OR}(\text{Result}, \text{Source_A}, \text{Source_B})$.

Этап 3. Выделение в алгоритме потоков арифметико-логической обработки и обработки структур. На данном этапе общий алгоритм разделяется на два взаимосвязанных потока операций, выполняемых на центральном процессоре и процессоре обработки структур. Для обеспечения эквивалентности результатов с последовательным вариантом необходимо включить в состав потока ЦП элементы синхронизации: команды обработки очередей PUT и GET. Приведем краткое описание процесса выполнения команд в режиме МКОД. Обработка команд начинается со стадии загрузки из локального ОЗУ процессора обработки структур кодов операций, операндов и тегов (тег — это бит достоверности, указывающий на готовность операнда к исполнению). Команды СП содержат различное количество операндов и, соответственно, тегов (см. табл. 3). В последней версии МКОД реализовано 6 форматов команд, содержащих от 1 до 3 операндов. Команда СП может быть выполнена когда все необходимые операнды валидны, т.е. их теги установлены. Если какой-либо операнд не является валидным, СП ожидает его поступления из ЦП. Последовательность передаваемых тегов для СП не имеет значение, так как вместе со значением тега передается его порядковый номер. Однако, для упрощения псевдокода алгоритма примем порядок передачи тегов «слева-направо», а номера тегов указывать не будем. Наиболее сложными в аппаратной реализации и временной сложности являются И-ИЛИ-НЕ команды и команды среза. Время выполнения таких команд может существенно отличаться в зависимости от заполнения памяти структур СП и линейно зависит от размерности задачи ($O(n)$). Остальные команды выполняются за время $O(1)$.

Пример 3 (разделение алгоритма на потоки ЦП и СП: команда поиска). Пусть необходимо выполнить поиск в структуре G по ключу, значение которого заранее для СП неизвестно. Данную ситуацию будем отображать знаком «?», что означает, что тег операнда не валиден, в результате чего для выполнения команды требуется получить его по шине данных из ЦП. Результат выполнения команды SEARCH отправляется в очередь данных в ЦП:

$\text{Data} \leftarrow \text{SEARCH}(\text{G}, \text{Key})$		
Поток ЦП		Поток СП
PUT(Key)	$\Rightarrow$	SEARCH(G,?)
$\text{Data} \leftarrow \text{GET}$	$\Leftarrow$	

Далее рассмотрим применение данной методики на примере алгоритмов нахождения максимального потока в транспортной сети.

3. Задача нахождения максимального потока в транспортной сети

Проблема определения максимального потока на графовой модели часто возникает в сетевых приложениях, энергетике, на транспорте, при планировании производственной деятельности и многих других важных приложениях. Часто возникает необходимость решать данную задачу в оперативном режиме на фоне постоянно изменяющихся входных данных, как, например, в задачах сетевого планирования.

Задача формулируется следующим образом. Дан граф $G(V, E)$ (рис. 3), в котором вес ребер интерпретируется как пропускная способность $c(u, v)$. Задача состоит в поиске максимального потока из вершины - источника s в сток t . На графе определены **потоки** $f(u, v)$ для ребер из u в v , для которых соблюдаются следующие правила:

- поток из u в v не превосходит пропускной способности. $f(u, v) \leq c(u, v)$;
- отрицательный поток эквивалентен аналогичному потоку в обратном направлении: $f(u, v) = -f(v, u)$;
- поток не изменяется при прохождении через узел: $\sum_v f(u, v) = 0 \iff f_{in}(u) = f_{out}(u)$ для всех узлов u (за исключением вершин s и t).

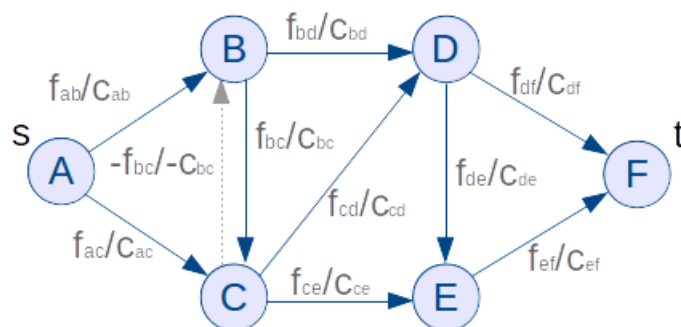


Рис. 3. Архитектура вычислительной системы с многими потоками команд и одним потоком данных

Рассматриваемый в данном исследовании алгоритм основан на итерационном поиске некоторого потока на остаточной сети $G_f(V, E_f)$, т.е. сети с пропускной способностью $c_f(u, v) = c(u, v) - f(u, v)$, которая получена из графа $G(V, E)$ благодаря сокращению пропускной способности ребер, входящих в ранее найденные потоки.

4. Алгоритм Форда — Фалкерсона

Алгоритм Форда — Фалкерсона состоит в поиске на остаточной сети дополняющего потока из s в t , который увеличивает суммарный найденный поток. После этого остаточная сеть модифицируется и итерация повторяется. Если дополняющий поток найти невозможно, то это означает, что найденный суммарный поток является максимальным. Ниже приведен псевдокод алгоритма, выраженный в общих терминах операций над множествами и графами.

Алгоритм 1. ФОРД-ФАЛКЕРСОН ($G(V, E), s, t$)

```
1:  $flow \leftarrow 0$  ▷ Максимальный поток
2:  $f(u, v) \leftarrow 0$  для всех ребер  $(u, v) \in E$  ▷ Инициализация потоков
3:  $G_f \leftarrow G$  ▷ Остаточная сеть
4: ЦИКЛ ПОКА ( $path \leftarrow \text{ПОИСК В ШИРИНУ}(G_f, s, t) \neq \emptyset$ ) ▷ Дополняющий путь
5:   ЦИКЛ  $(u, v) \in path$  ▷ Поиск потока для найденного пути  $path$ 
6:     ЕСЛИ  $c_f(path) > c(u, v)$  ТО
7:        $c_f(path) \leftarrow c(u, v)$ 
8:     ВСЕ ЕСЛИ
9:   ВСЕ ЦИКЛ
10:  ЦИКЛ  $(u, v) \in path$  ▷ Изменение остаточной сети
11:     $f(u, v) \leftarrow f(u, v) - c_f(path)$  ▷ Пропускная способность прямого потока
12:     $f(v, u) \leftarrow f(v, u) + c_f(path)$  ▷ Пропускная способность обратного потока
13:  ВСЕ ЦИКЛ
14:   $flow \leftarrow flow + c_f(path)$ 
15: ВСЕ ЦИКЛ ПОКА
16: РЕЗУЛЬТАТ  $\leftarrow flow$ 
```

Алгоритм использует метод поиска в ширину для отыскания пути на остаточной сети G_f , который подробно описан в [7]. Пусть граф $G(V, E)$ задан подобно примеру 2, что обеспечивает обход графа операциями NEXT(структура, ключ). Необходимо на основе G создать структуру остаточной сети, которая будет использоваться для следующих операций:

- поиск всех вершин, инцидентных вершине u ;
- определение потока для всех ребер, исходящих из вершины;
- изменение пропускной способности ребер;
- добавление обратных ребер (v, u) к вершине v для найденных потоков и их изменение.

В связи с этим в качестве ключа может быть выбраны индексы вершин u и v $G.KEY[u, v]$, как это было показано ранее в табл. 2. Поле данных будет состоять из остаточной пропускной способности ребра: $G.DATA[c]$. Для определения остаточного пути потребуется хранить множество достижимых из s вершин, что целесообразно сделать в отдельной структуре P , которая будет создаваться в ходе поиска пути по методу поиска в ширину, а после очередной операции будет удаляться операцией $DELSTR(P)$. Для метода поиска в ширину характерно раскрашивание вершин в цвета:

- черный цвет означает то, что вершина достигнута при поиске и алгоритм приступил к рассмотрению всех связанных с ней вершин. Кодировем данный цвет числом «0».
- серый цвет вершины означает что данная вершина добавлена в список, но связанные с ней вершины еще не рассмотрены. Кодировем данный цвет числом «1».
- белый цвет означает то, что вершина еще не рассмотрена и путь к ней не определен. Данный тип вершин будет отсутствовать в структуре P , что и будет указывать на это состояние.

Таким образом, алгоритм поиска в ширину будет вести поиск «серых» вершин (значение «1»), добавлять инцидентные с ними вершины в структуру P , менять цвет рассмотренной вершины на «черный» (значение «0»). Поле ключа $P.KEY$ состоит из цвета $color$ и номера вершины v : $P.KEY[color, v]$. Поле данных $P.DATA$ содержит номер u предыдущей вершины в пути: $P.DATA[u]$. Таким образом, алгоритм Форда — Фалкерсона представлен ниже.

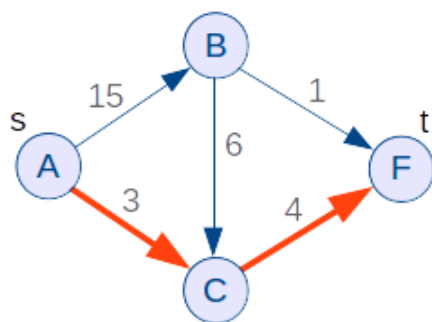
Алгоритм 2. ФОРД-ФАЛКЕРСОН ($G(V, E), s, t$) /псевдокод ЦП в режиме сопроцессора/

```

1:  $flow \leftarrow 0$                                 ▷ Максимальный поток
2:  $P \leftarrow \text{ПОИСК В ШИРИНУ}(G, s, t)$ 
3: ЦИКЛ ПОКА  $\text{SEARCH}(P, [0, t]) \neq 0$ 
4:    $v\_index \leftarrow t$                                 ▷ Вычисление  $c_f$ 
5:    $c_f \leftarrow \infty$ 
6:   ПОВТОРЯТЬ                                       ▷ Вычисление потока из  $s$  в  $t$ 
7:      $path \leftarrow \text{SEARCH}(P, [0, v\_index])$            ▷ Обход потока
8:      $u\_index \leftarrow path.DATA.u$                  ▷ Предшествующая вершина
9:      $u \leftarrow \text{SEARCH}(G, [u\_index, v\_index])$        ▷ Поиск вершины графа
10:     $c_f \leftarrow \min(u.DATA.c - c_f)$ 
11:     $v\_index \leftarrow u\_index$ 
12:  ЦИКЛ ПОКА ( $path \neq s$ )
13:     $v\_index \leftarrow t$                                 ▷ Изменение остаточной сети
14:    ПОВТОРЯТЬ                                       ▷ Вычисление потока из  $s$  в  $t$ 
15:       $path \leftarrow \text{SEARCH}(P, [0, v\_index])$ 
16:       $u\_index \leftarrow path.DATA.u$ 
17:       $u \leftarrow \text{SEARCH}(G, [u\_index, v\_index])$        ▷ Поиск вершины  $u$ 
18:       $v \leftarrow \text{SEARCH}(G, [v\_index, u\_index])$        ▷ Поиск вершины  $v$ 
19:       $\text{INSERT}(G, [u\_index, v\_index], [u.DATA.c - c_f])$ 
20:       $\text{INSERT}(G, [v\_index, u\_index], [v.DATA.c + c_f])$ 
21:       $v\_index \leftarrow u\_index$ 
22:    ЦИКЛ ПОКА ( $path \neq s$ )
23:     $\text{DELSTR}(P)$ 
24:     $P \leftarrow \text{ПОИСК В ШИРИНУ}(G, s, t)$ 
25: ВСЕ ЦИКЛ ПОКА
26: РЕЗУЛЬТАТ  $\leftarrow flow$ 

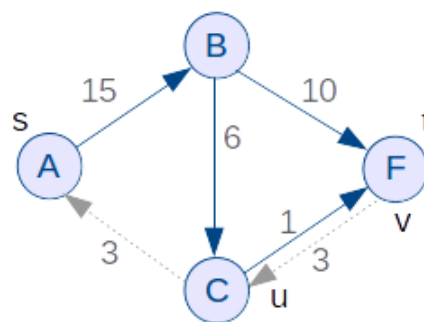
```

Следует заметить, что несмотря на использование псевдокода, он описан в низкоуровневых командах МКОД системы. Это свидетельствует о высокой аппаратной поддержке действий со структурами данных и упрощении программирования сложных структур при использовании данной технологии. Пример заполнения структур и первая итерация алгоритма Форда — Фалкерсона показаны на рис. 4. В начале работы алгоритма в строке 1 устанавливается нулевой максимальный поток. Далее начинается цикл поиска остаточного пути. В строках 2 и 24 используется алгоритм поиска в ширину для нахождения пути из вершину s в t , результатом которого является структура P . Если в структуре есть запись о вершине t , то путь найден. Далее в строках 3–12 итерационно происходит обход найденного пути и определяется величина потока c_f .



G.KEY		G.DATA
u	v	c
A	0	0
A	B	15
A	C	3
B	0	0
B	C	6
B	F	1
C	0	0
C	F	4
F	0	0

a



P.KEY		P.DATA
color	v	u
0	A	0
0	B	A
0	C	A
0	F	C

б

G.KEY		G.DATA
u	v	c
A	0	0
A	B	15
A	C	0
B	0	0
B	C	6
B	F	1
C	0	0
C	A	3
C	F	1
F	0	0
F	C	3

Рис. 4. Этапы выполнения алгоритма Форда — Фалкерсона на графе: *a* — начальное состояние структуры *G*; *б* — остаточная сеть и состояние структур *G* и *P* после первого шага алгоритма

В цикле в строках 14–22 выполняется модификация остаточной сети *G*. Путь, как и ранее, обходится в обратной последовательности, от вершины *t* к вершине *s*. В строке 15 происходит поиск очередной вершины потока, который используется в строке 16 для определения индекса предшествующей вершины *u*. Далее происходит поиск в остаточной сети записей о ребре (u, v) и обратном ребре (v, u) . Далее выполняется изменение информации об остаточной пропускной способности в этих ребрах в строках 19 и 20. После обработки потока структура *P* удаляется. Алгоритм поиска в ширину приведен ниже.

Алгоритм 3. ПОИСК В ШИРИНУ(GF, s, t) /псевдокод ЦП в режиме сопроцессора/

- 1: INSERT($P, [1, s], [0]$) ▷ Инициализация *P*, цвет «серый»
- 2: **ПОВТОРЯТЬ**
- 3: $path \leftarrow \text{MAX}(P)$
- 4: **ЕСЛИ** $path.KEY.color = 1$ **ТО** ▷ Еще есть не пройденные вершины *u*
- 5: $u_index \leftarrow path.KEY.v$ ▷ Индекс вершины *v*
- 6: $u \leftarrow \text{NEXT}(G, [u_index, 0])$ ▷ Вершины *u*

```

7:      ЦИКЛ ПОКА  $u.KEY.u == u\_index$ 
8:       $v\_index \leftarrow u.KEY.v$  ▷ Индекс вершины  $v$ 
9:      ЕСЛИ  $(v.DATA.c > 0)$  И  $(SEARCH(P, [0, v\_index]) == 0)$  ТО
10:          $INSERT(P, [1, v\_index], [u\_index])$ 
11:      ВСЕ ЕСЛИ
12:       $u \leftarrow NEXT(G, [u\_index, v\_index])$  ▷ Переход к следующему ребру
13:      ВСЕ ЦИКЛ ПОКА
14:      ВСЕ ЕСЛИ
15:       $u \leftarrow DELETE(P, [1, u\_index])$  ▷ Изменяем цвет вершины  $u$ 
16:       $INSERT(P, [0, u\_index], u.DATA.u)$ 
17: ЦИКЛ ПОКА  $(u\_index \neq t)$  И  $(path.KEY.color = 1)$  ▷ Пока не достигнем  $t$  или существует путь
18: РЕЗУЛЬТАТ  $\leftarrow P$ 

```

Алгоритм поиска в ширину предполагает проход по графу от исходной вершины s по каждому из ребер. В строке 1 инициализируется состояние структуры P добавлением вершины s серого цвета. Далее цикл в строках 2–17 повторяется до тех пор, пока не будет найден путь или не будут просмотрены все вершины графа. Так как ключ поиска в структуре P в старшей части содержит поле цвета, максимальный ключ в строке 3 будет выбран в первую очередь серого цвета со значением «1». Только если таких вершин нет, будет выбрана вершина черного цвета, в том числе вершина t . Если выбрана вершина серого цвета, происходит обработка связанных с ней вершин (строки 5–13). Новая вершина добавляется в P только после проверки остаточного потока и цвета вершины в строке 9.

5. Анализ работы алгоритма в режиме МКОД

Как было сказано ранее и в работах [2, 4], процессор обработки структур связан с локальной памятью команд (см. рис. 2), которая содержит коды операций над структурами и множествами (набор операций приведен в табл. 3). Операции могут содержать только часть из необходимых операндов, что заставляет блок выборки команд СП ожидать недостающих операндов от ЦП. Поэтому, оба потока команд должны быть синхронизированы, что выполняется со стороны ЦП посылкой операндов, тегов и меток переходов. СП, в свою очередь, посылает в ответ результаты выполнения команд через очередь данных, из которой ЦП получает необходимые результаты.

Рассмотрим набор операций над структурами данных, которые используются в алгоритме Форда — Фалкерсона. В алгоритме 2 наиболее часто используется команда поиска $SEARCH(структура, ключ)$. Операнды команды задают одну из нескольких структур в памяти процессора обработки структур, а также ключ поиска. В последней версии СП возможно одновременное хранение до семи различных структур. Соответственно, номера структур, указанные в алгоритме в виде P или G , представляют собой определенные це-

лые числа от 1 до 7 (структура 0 зарезервирована и не может использоваться в командах). Команда INSERT(структура,ключ,значение) также начинается с поиска по ключу. Если в структуре уже существует запись с указанным ключом, то происходит замещение ранее записанных данных на те, что указаны в команде. Команда перехода JT(тег,адрес) служит для синхронизации ЦП и СП. Поле «тег» — это битовое поле, которое определяет результат ветвления. Если тег равен нулю, переход по адресу не происходит, а код начинает выполняться со следующей команды за командой ветвления. Если тег равен единице, то происходит переход по адресу, указанному в команде. Тег также может быть не задан, что обозначается символом «?». Возможны несколько вариантов реализации команды:

- безусловный переход — JT(тег,адрес). Так как команда не ожидает операндов, она выполняется в СП незамедлительно.
- условный переход по тегу — JT(?,адрес). Процессор обработки структур ожидает из ЦП только тег направления перехода. По нулевому тегу совершается переход по указанному адресу, а по единичному тегу — переход к следующей команде. ЦП исполняет команду PUT(тег), по которой тег посылается в СП.
- условный переход по адресу — JT(тег,?). Процессор обработки структур ожидает адрес перехода, и при его поступлении выполняет переход. ЦП исполняет команду PUT(адрес), по которой адрес перехода посылается в СП.
- условный переход по адресу и тегу — JT(?,?). Процессор обработки структур ожидает тег и адрес перехода. Данный случай менее востребован в алгоритмах. ЦП исполняет команду PUT(тег,адрес).

Ниже представлен алгоритмы для обоих потоков. Слева показан поток ЦП, а справа — поток СП.

Алгоритм 4. ФОРД-ФАЛКЕРСОН ($G(V, E)$, s, t) /псевдокод в режиме МКОД/

Поток ЦП	Поток СП
1: $flow \leftarrow 0$	
2: $P \leftarrow \text{ПОИСК В ШИРИНУ}(G, s, t)$	
3: ЦИКЛ ПОКА GET $\neq 0$	▷ ff_label1: SEARCH(P, [0..t])
4: PUT(0)	▷ JT(?, ff_label4)
5: $v_index \leftarrow t$	
6: $c_f \leftarrow \infty$	
7: ПОВТОРЯТЬ	
8: PUT(1)	▷ JT(?, ff_label2)
9: $path \leftarrow \text{PUT}([0, v_index])$	▷ ff_label2: SEARCH(P, ?)
10: $u_index \leftarrow path.DATA.u$	
11: PUT([u_index, v_index])	▷ SEARCH(G, ?)
12: $u \leftarrow \text{GET}$	
13: $c_f \leftarrow \min(u.DATA.c - c_f)$	
14: $v_index \leftarrow u_index$	

```

15:   ЦИКЛ ПОКА ( $path \neq s$ )
16:   PUT(0)                                     ▷ JT(?, ff_label2)
17:    $v\_index \leftarrow t$ 
18:   ПОВТОРЯТЬ
19:     PUT(1)                                     ▷ JT(?, ff_label3)
20:     PUT( $v\_index$ )                             ▷ ff_label3: SEARCH(P, ?)
21:      $path \leftarrow GET$ 
22:      $u\_index \leftarrow path.DATA.u$ 
23:     PUT( $[u\_index, v\_index^*]$ )                 ▷ SEARCH(G, ?)
24:      $u \leftarrow GET$ 
25:     PUT( $[u\_index^*, v\_index^*]$ )                 ▷ SEARCH(G, ?)
26:      $v \leftarrow GET$ 
27:     PUT( $[u\_index^*, v\_index^*]$ )
28:     PUT( $u.DATA.c - c_f$ )                       ▷ INSERT(G, ?, ?)
29:     PUT( $v\_index^*, u\_index^*$ )
30:     PUT( $v.DATA.c + c_f$ )                       ▷ INSERT(G, ?, ?)
31:      $v\_index \leftarrow u\_index$ 
32:   ЦИКЛ ПОКА ( $path \neq s$ )
33:   PUT(0)                                     ▷ JT(?, ff_label3)
34:                                           ▷ DELSTR(P)
35:    $P \leftarrow \text{ПОИСК В ШИРИНУ}(G, s, t)$ 
36: ВСЕ ЦИКЛ ПОКА                             ▷ JT(1, ff_label1)
37: РЕЗУЛЬТАТ  $\leftarrow flow$                    ▷ ff_label4:

```

Остановимся более подробно на организации структурных конструкций при взаимодействии двух процессоров (табл. 4). Так как управление переходами осуществляет центральный процессор, код СП должен быть построен таким образом, чтобы соответствовать таким общепринятым операторам, как условия и циклы. При этом в код ЦП добавляются команды PUT для передачи тегов с условиями переходов. Возможны несколько вариантов организации каждого типа команд, однако представленные ниже варианты наиболее приемлемы по размерам передаваемых данных, так как требуют передачи только тега без использования адресов.

Алгоритм поиска в ширину, использующий указанные конструкции, показан ниже.

Алгоритм 5. ПОИСК В ШИРИНУ(G, s, t) /псевдокод ЦП в режиме МКОД/

Поток ЦП	Поток СП
1:	▷ INSERT(P, [1, s], [0])
2: ПОВТОРЯТЬ	
3: PUT(1)	▷ JT(?, bfs_label1)
4:	▷ bfs_label1 : MAX(P)
5: $path \leftarrow GET$	
6: ЕСЛИ $path.KEY.color = 1$ ТО	

```

7:      PUT(0)                                     ▷ JT(?,bfs_label4)
8:       $u\_index \leftarrow path.KEY.v$ 
9:      PUT([ $u\_index,0$ ])                         ▷ NEXT(G,?)
10:      $u \leftarrow GET$ 
11:     ЦИКЛ ПОКА  $u.KEY.u == u\_index$                 ▷ bfs_label2:
12:         PUT(0)                                  ▷ JT(?,bfs_label4)
13:          $v\_index \leftarrow u.KEY.v$ 
14:         PUT([ $0,v\_index$ ])                       ▷ SEARCH(P,?)
15:         ЕСЛИ ( $v.DATA.c \gg 0$ ) И ( $GET == 0$ ) ТО
16:             PUT(0)                              ▷ JT(?,bsf_label3)
17:             PUT([ $1,v\_index^*$ ])
18:             PUT([ $u\_index^*$ ])                   ▷ INSERT(P,?,?)
19:         ВСЕ ЕСЛИ
20:             PUT(1)
21:             PUT([ $u\_index^*,v\_index^*$ ])          ▷ bsf_label3: NEXT(G,?)
22:              $u \leftarrow GET$ 
23:         ВСЕ ЦИКЛ ПОКА                          ▷ JT(1,bfs_label2)
24:             PUT(1)
25:         ВСЕ ЕСЛИ
26:             PUT(1)
27:             PUT([ $1,u\_index^*$ ])                 ▷ bfs_label4: DELETE(P,?)
28:              $u \leftarrow GET$ 
29:             PUT([ $0,u\_index^*$ ])
30:             PUT( $u.DATA.u$ )                      ▷ INSERT(P,?,?)
31:     ЦИКЛ ПОКА ( $u\_index \neq t$ ) И ( $path.KEY.color=1$ )
32:         PUT(0)                                  ▷ JT(?, bfs_label1)
33:     РЕЗУЛЬТАТ  $\leftarrow P$ 

```

Как видно из приведенных алгоритмов, при решении задачи отыскания максимального потока по методу Форда — алкерсона и при принятых базовых принципах функционирования МКОД системы не достигается высокой параллельности при выполнении двух потоков. Это происходит в связи с тем, что потоки оказываются сильно связанными из-за пересылки большого количества данных. Однако, легко заметить, что часть команд использует повторно посылаемые данные, ранее уже использованные при обращении к структурам. Такие данные отмечены символом «*». В алгоритмах 5 и 4 количество команд, основанных на использовании повторно полученных данных достаточно велико и составляет около 30%. Общее количество команд, зависящих только от локальных данных СП составляет около половины от всех команд:

Таким образом возможно достичь большей степени параллельности МКОД системы, если использовать специальный регистровый файл, позволяющий создавать составные ключи и выполнять действия над ними внутри процессора обработки структур. В настоящее время ведется проектирование указанного блока.

Таблица 4

**Пример преобразования структурных конструкций одного потока в два потока
для исполнения в ЦП и СП**

Пример конструкции	Поток ЦП	Поток СП
ПОВТОРЯТЬ Действие ЦИКЛ ПОКА (Условие)	ПОВТОРЯТЬ PUT(1) Действие ЦИКЛ ПОКА (Условие) PUT(0)	JT(?,метка1) метка1: Действие JT(?,метка1)
ЦИКЛ ПОКА (Условие) Действие ВСЕ ЦИКЛ ПОКА	ЦИКЛ ПОКА (Условие) PUT(0) Действие ВСЕ ЦИКЛ ПОКА PUT(1)	метка1: JT(?,метка2) Действие JT(1,метка1) метка2:
ЕСЛИ (Условие) Действие ВСЕ ЕСЛИ	ЕСЛИ (Условие) PUT(0) Действие ВСЕ ЕСЛИ PUT(1)	JT(?,метка1) Действие метка1:
ЕСЛИ (Условие) Действие ИНАЧЕ Действие ВСЕ ЕСЛИ	ЕСЛИ (Условие) PUT(0) Действие ИНАЧЕ PUT(1) Действие ВСЕ ЕСЛИ	JT(?,метка1) Действие JT(1,метка2) метка1: Действие метка2:

Таблица 5

**Количество независимых и мало зависимых команд
обработки структур и множеств**

Алгоритм	Общее количество команд обработки структур	Количество независимых команд	Количество команд, использующих повторно полученные данные
Форд-Фалкерсон	9	1	3
Поиск в ширину	8	2	3

Заключение

Проведенные исследования реализации алгоритма Форда — Фалкерсона показывают, что МКОД системы способна решать подобные задачи на графовых моделях как в режиме сопроцессора, так и в случае независимого исполнения потоков ЦП и СП. На основе общей схемы алгоритма удалось получить модифицированную версию, представленную в базисных операциях над структурами данных, а также версию с двумя потоками команд и одним потоком данных. Это позволяет сделать следующие выводы:

- Разработанная методика модификации алгоритма была уточнена и продемонстрирована на одном из общих и важных алгоритмов оптимизации на сетях и графах.

- Выбранные шаблоны преобразования структурных операторов языка высокого уровня позволяют получить модификацию структурированных алгоритмов к виду, реализуемому в МКОД системе.

- Возможно и целесообразно создание средств автоматизированного преобразования и оптимизации программ на языках высокого уровня к МКОД формату.

- Необходимо совершенствовать структуру МКОД системы и процессора обработки структур. В частности, целесообразно включить в структуру процессора блок регистров, а также блок генерации составных ключей. Это позволит существенно повысить независимость потоков СП и ЦП и сократит объем межпроцессорного взаимодействия.

Список литературы

1. Попов А.Ю. Электронная вычислительная машина с многими потоками команд и одним потоком данных: пат. 71016 РФ. 2008. Бюл. № 5. 1 с.
2. 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.
3. Попов А.Ю. Электронная вычислительная машина с аппаратной поддержкой операций над структурами данных // Аэрокосмические технологии: тр. Второй Междунар. науч.-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.
4. 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 01.08.2014).
5. 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 01.08.2014).
6. Кнут Д. Искусство программирования. В 3 т. Т. 3. Сортировка и поиск: пер. с англ. 2-е изд. М.: Вильямс, 2000. 832 с.
7. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ: пер. с англ. М.: МЦНМО, 2000. 960 с.
8. Попов А.Ю. Модели и алгоритмы автоматизированной декомпозиции схем ЭВМ: дис. ... канд. техн. наук. М., 2003. 176 с.

On the implementation of the Ford — Fulkerson algorithm on the Multiple Instruction and Single Data computer system

Popov A. Yu.^{1,*}

^{*}alexpov@bmstu.ru

¹Bauman Moscow State Technical University, Moscow, Russia

Keywords: maximum flow, Ford — Fulkerson algorithm, MISD computer system, structure processor.

Algorithms of optimization in networks and direct graphs find a broad application when solving the practical tasks. However, along with large-scale introduction of information technologies in human activity, requirements for volumes of input data and retrieval rate of solution are aggravated. In spite of the fact that by now the large number of algorithms for the various models of computers and computing systems have been studied and implemented, the solution of key problems of optimization for real dimensions of tasks remains difficult. In this regard search of new and more efficient computing structures, as well as update of known algorithms are of great current interest.

The work considers an implementation of the search-end algorithm of the maximum flow on the direct graph for multiple instructions and single data computer system (MISD) developed in BMSTU. Key feature of this architecture is deep hardware support of operations over sets and structures of data. Functions of storage and access to them are realized on the specialized processor of structures processing (SP) which is capable to perform at the hardware level such operations as: add, delete, search, intersect, complete, merge, and others. Advantage of such system is possibility of parallel execution of parts of the computing tasks regarding the access to the sets to data structures simultaneously with arithmetic and logical processing of information.

The previous works present the general principles of the computing process arrangement and features of programs implemented in MISD system, describe the structure and principles of functioning the processor of structures processing, show the general principles of the graph task solutions in such system, and experimentally study the efficiency of the received algorithms.

The work gives command formats of the SP processor, offers the technique to update the algorithms realized in MISD system, suggests the option of Ford-Falkersona algorithm to search the maximum flow on direct graph for MISD model of the computing system and the options to update an architecture of MISD system that lead to its increasing performance.

References

1. 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).
2. 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'yuternye sistemy [Information technology and computer systems], pp. 83–87. (in Russian).
3. 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).
4. 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 01.08.2014 (in Russian).
5. 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 01.08.2014. (in Russian).
6. 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.).
7. 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.).
8. Popov A.Iu. Modeli i algoritmy avtomatizirovannoi dekompozitsii skhem EVM. Kand. diss. [Models and algorithms for automated decomposition of the computer scheme. Cand. diss.]. Moscow, 2003. 176 p. (in Russian).