Об оптимальном кешировании FIFO-очередей[1]

А. В. Соколов д. ф.-м. н.[2]
ПетрГУ, ИПМИ КарНЦ РАН, Петрозаводск
avs@krc.karelia.ru

В работе предложена математическая модель в виде случайного блуждания по целочисленной решетке в треугольнике и алгоритм оптимального управления FIFO-очередью в двухуровневой памяти. Предполагается, что мы работаем с очередью, которая может превзойти выделенный размер быстрой памяти. В этом случае мы переносим часть элементов очереди, которая расположены в конце очереди в память второго уровня, и включаем и исключаем элементы в быстрой памяти. В случае нового переполнения или опустошения самых старых элементов в быстрой памяти, мы производим перераспределение очереди между уровнями памяти,так, чтобы сохранить FIFO-порядок работы, и чтобы среднее число перераспределений очереди было минимальным, т. е. среднее время работы между перераспределениями очереди было максимальным.

Ключевые слова: FIFO-очереди, динамические структуры данных, математическое моделирование, кеширование.

1. Введение

FIFO очереди [1, 2] широко используются в компьютерных сетях, операционных системах, графических системах, устройствах промышленной автоматики и в других приложениях. Ряд фирм выпускает микросхемы, реализующие работу с FIFO-очередями. При работе с FIFO-очередями возможна ситуация переполнения очереди, что приводит к снижению эффективности, так как в некоторых приложениях аварийное прерывание в этом случае вообще говоря недопустимо.

Хорошо известны аппаратные и программные механизмы кэш-памяти и виртуальной памяти, когда обмены между уровнями памяти при переполнении быстрой памяти происходят автоматически. Кэш и виртуальная памяти пытаются работать одинаково со всеми структурами данных и программами. Понятно, что универсальный механизм не может быть использован во всех ситуациях. Например, в операционных системах реального времени механизм

[1]Работа выполнена при финансовой поддержке РФФИ (грант №12-01-00253-а) и Программы стратегического развития ПетрГУ в рамках реализации комплекса мероприятий по развитию научно-исследовательской деятельности.

виртуальной памяти не используется, так как не может обеспечить ограничения по времени. Для некоторых, наиболее важных, структур данных известны альтернативные реализации. Обсудим, например, некоторые варианты организации управления стеками.

Понятие стека широко используется при разработке программного и аппаратного обеспечения. Ряд языков (например, форт [3–5] и Postscript [2]), которые используют обратную польскую запись, так и называются стековыми. Стеки используются в трансляторах, в алгоритмах машинной графики, в комбинаторных алгоритмах, при разработке архитектуры компьютеров и вообще везде, где процессы имеют рекурсивный характер.

При разработке RISC процессоров для работы со скалярными аргументами и локальными переменными функций организуются перекрывающиеся окна регистров постоянного или переменного размеров [6, 7]. В этом случае получаем один стек с элементами типа "окно" в главной памяти, у которого несколько верхних элементов образуют циклический стек на регистрах. В архитектуре Intel Itanium у каждой процедуры есть регистровый стек с элементами переменного размера для хранения параметров. Выходные параметры одной процедуры являются входными параметрами другой.

При переполнении или опустошении регистровой вершины стека на практике применяют несколько программных и аппаратных методов [6–9]:

1. Large Stack. Можно сделать стек большим и предполагать, что переполнения не будет, а если оно все же произошло, то совершить аварийное завершение работы. Такое решение в некоторых приложениях, например в системах реального времени, недопустимо.

2. Demand fed single element stack manager. В этой стратегии вершина стека реализована аппаратно, как циклический буфер, а продолжение стека находится в памяти. При переполнении и опустошении аппаратного стека перемещается один элемент (или несколько элементов) в память из буфера или наоборот.

3. Paging stack manager. В этом методе вершина реализована программно как часть специальной памяти. При опустошении вершины стека из памяти копируется половина буфера, а при переполнении нижняя половина буфера перемещается в память, верхняя половина перемещается в начало буфера.

4. Barometr-Pointer Algorithm. Предлагались реализации стека,

когда перемещения элементов между уровнями памяти происходят не при переполнении или опустошении, а при отклонении от оптимального состояния вершины (обычно считается, что это половина быстрого буфера) в фоновом режиме, т. е. параллельно с обычным выполнением команд (аналог опережающей подкачки и откачки в страничной виртуальной памяти) [9].

5. В [8] предлагалась реализация стека, когда при переполнении и при опустошении быстрой вершины стека переносится k элементов стека.

6. An associative cashe. Кэш-память для стековых машин не дает преимуществ по сравнению с рассмотренными методами, т.к. сложнее для аппаратной реализации и в тоже время не учитывает специфики стека как структуры LIFO. Но, если в стек нужно помещать структуры данных переменного размера, такие как строки и структуры, использование кэш-памяти, как утверждается в работе [6], в некоторых случаях может быть целесообразно.

Во всех этих методах вершина стека является продолжением стека, находящегося в памяти второго уровня. Возможны также аппаратные методы реализации, когда вершина стека является копией стека, находящегося в памяти второго уровня. Тогда запоминание элементов при переполнении не нужно, а требуется лишь восстановление при опустошении.

В работах [8,10] предложены математические модели и алгоритмы оптимального управления вершиной одного стека в двухуровневой памяти, а в [11] рассмотрена задача оптимального управления в двухуровневой памяти вершинами двух стеков, растущих навстречу друг другу в быстрой памяти.

FIFO-очереди также достаточно важны особенно для сетевых приложений. Так алгоритмы работы с очередями играют первостепенное значение при разработки сетевых процессоров [12, 13].

В связи с этим представляется полезным анализировать некоторые альтернативные методы кеширования FIFO-очередей. В этой статье мы будем рассматривать чистые очереди (FIFO), в которых доступ внутрь не разрешается. На практике чаще используются именно такие структуры данных.

Мы рассмотрим ситуацию, когда пользователь при переполнении быстрой памяти не желает прерывать задачу, а хочет довести решение до конца. Предполагается, что свободных участков быстрой памяти больше нет, но есть память второго уровня, которую мы

хотим задействовать для работы с очередью. Отметим, что важным применением очереди является сопряжение нерегулярного источника данных с их обработчиком. Очередь тогда выполняет роль буфера (например, буфера ввода/вывода, когда устройства ввода/вывода могут производить соответствующие операции одновременно с работой программы), обеспечивающего обработчику возможность работать в свойственном ему ритме. Предложенный в данной работе алгоритм будет здесь полезен, если в ситуации переполнения буфера мы можем терять данные, поступающие на обработку. Тогда мы, затратив некоторое время на перераспределение очереди, будем в течение определенного промежутка времени (который мы максимизируем) работать без потерь данных и без перераспределений. Такой метод управления очередью можно применять в сетевых устройствах, (где пакеты в случае переполнения очереди теряются), если память второго уровня достаточно быстрая.

Разработанные специально под конкретную структуру данных алгоритмы обмена между уровнями памяти в некоторых ситуациях будут работать лучше, чем универсальные алгоритмы замещения в страничной виртуальной памяти или в кэш-памяти. Предположим, что в некой гипотетической кэш-памяти мы работаем с одной большой FIFO-очередью. Тогда алгоритм замещения страниц LRU будет при переполнении удалять ту страницу, где находится почти самая старая часть очереди, т. е. следующую страницу после страницы с началом очереди (или ее саму, если начало очереди лежит на границе страницы), так как он удаляет страницу, к которой дольше всего не было обращений. Та страница, в которой находится начало не будет кандидатом на удаление, так как к ней обращались позже для удаления элементов. Оптимальная же стратегия должна удалять элементы (страницы в случае страничной памяти), к которым дольше всего не будет обращений [14], то есть конец очереди. Только опять же не ту страницу, где находится конец очереди (так как эта страница может понадобиться в быстрой памяти для вставки новых элементов), а следующую страницу. Мы предполагаем, что очередь будет занимать достаточно много страниц. Если же, например, есть всего три страницы и в одной находится конец, в другой начало, а средняя страница вся заполнена элементами очередей, то и LRU и оптимальный алгоритм замещения будут удалять среднюю страницу.

В этой статье мы пытаемся решить вопрос о том, сколько эле-

ментов надо удалять, а сколько оставлять в быстрой памяти при переполнении очереди. Предлагается математическую модель и основанный на ней алгоритм оптимального управления FIFO-очередью, в случае когда длина очереди становится больше выделенного ей размера быстрой памяти.

2. Последовательное представление циклической очереди в двухуровневой памяти

Под последовательной реализацией очереди обычно понимают циклическую реализацию [1], так как в противном случае очередь быстро уходит в бесконечность, если не происходит больших последовательных серий исключений, которые опустошают очередь. Пусть R — это указатель на конец очереди, F — указатель на место, непосредственно перед началом очереди (рис. 1).

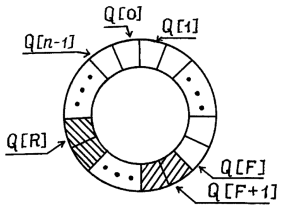

Рис. 1

Включение элемента происходит в конец очереди, а исключение — из начала очереди. Если F догнал R, то произошло опустошение очереди, а если R догнал F (рис. 2), то происходит аварийное завершение по переполнению.

Пусть n — это размер выделяемого для очереди массива. Он на 1 больше размера очереди так как в циклической реализации [1], один элемент массива обычно оставляют свободным, чтобы отличить пустую очередь от переполненной. Можно вместо этого хранить длину очереди, но это эквивалентно потере одной ячейки.

Пусть в памяти второго уровня очередь хранится в массиве (файле) $Q2$. Переменные $R2, F2$ — указатели на конец и начало очереди в памяти второго уровня, S — указатель на последний начальный элемент очереди в памяти первого уровня. Мы здесь

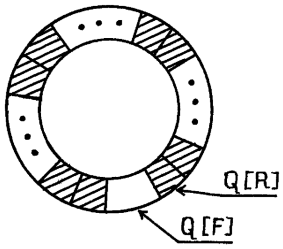

Рис. 2

считаем, что очередь в памяти второго уровня также представлена циклически, хотя для нашей модели и алгоритма это не существенно. В случае переполнения в быстрой памяти мы оставляем y_0 начальных элементов очереди, а $n - y_0$ конечных элементов переписываем в память второго уровня. При этом с помощью указателя $S = (F + y_0)\ mod\ n$ запоминаем то место в очереди, начиная с которого, если переменная F будет указывать на него, мы должны вернуть порцию данных с памяти второго уровня (рис.3).

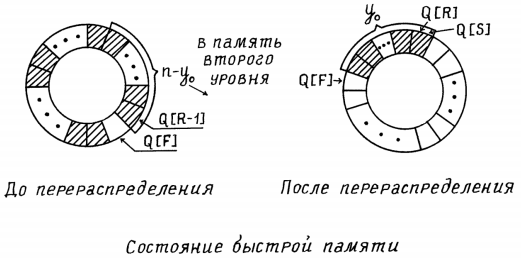

Рис. 3

S указывает туда же, куда и новое значение R, т. е. на то место, после которого будут включаться новые элементы в очередь.

Рассмотрим вариант перераспределений очереди для конкретных значений $n = 6, y_0 = 4$. Пусть мы произвели 6 включений в очередь. Тогда мы получим следующую картину после перераспределения памяти (рис. 4).

Теперь перераспределение памяти будет необходимо в двух случаях.

1. Быстрая память опять переполнится, и тогда мы должны

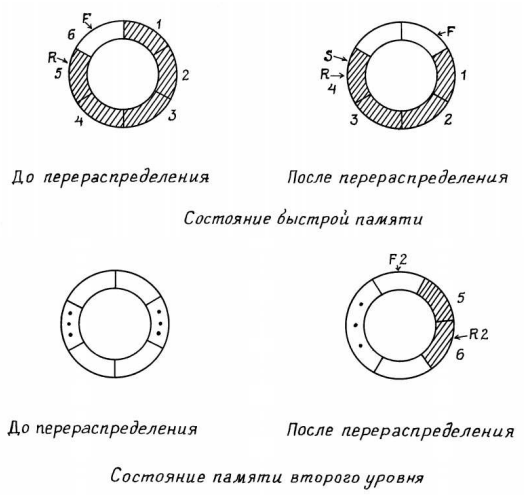

Рис. 4

произвести реорганизацию очереди таким образом, чтобы получить в быстрой памяти y_0 начальных (самых старых) элементов. Этот кусок очереди будет формироваться следующим образом. Во-первых, это l начальных (самых старых) элементов очереди, которые сохранились с момента предыдущего перераспределения, где l вычисляется следующим образом:

```
if S>F then l:= S-F else l:=n-F+S
```

Следующие $y_0 - l$ элементов восстанавливаются из памяти второго уровня. При этом возможна ситуация, что в памяти второго уровня хранится меньше чем $y_0 - l$ элементов. Тогда недостающие элементы — это самые старые из последних (вновь вставленных) элементов очереди из быстрой памяти. Эти элементы сдвигаются в конец куска очереди в быстрой памяти. Остальные вновь вставленные элементы мы выталкиваем из быстрой памяти в конец куска очереди, находящейся в памяти второго уровня.

Ниже (рис. 5)показана ситуация, когда в памяти второго уровня хранится не меньше чем $y_0 - l$ элементов. Таким образом, мы выталкиваем из быстрой памяти все $n - l$ последних элементов.

Приводим состояние очереди (рис. 6), если в ситуации, изобра-

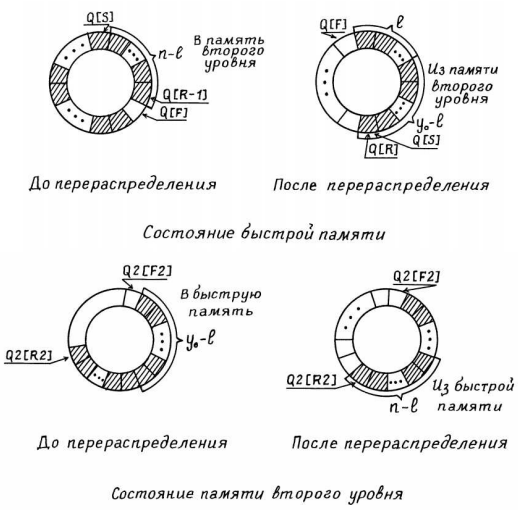

Рис. 5

женной на рисунке 4, было произведено 3 исключения и 5 включений.

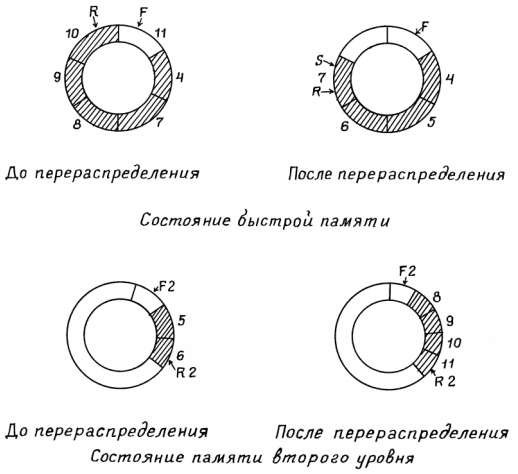

Рис. 6

2. Мы исчерпаем число наиболее старых элементов в быстрой памяти, и тогда надо y_0 самых старых элементов восстановить из начала очереди в памяти второго уровня. В это время в быстрой памяти хранится l последних элементов очереди, где l вычисляется по следующей формуле:

```
if R>F then l:= R-F else l:=n-F+R
```

Эти элементы мы выталкиваем в конец очереди второго уровня (рис. 7).

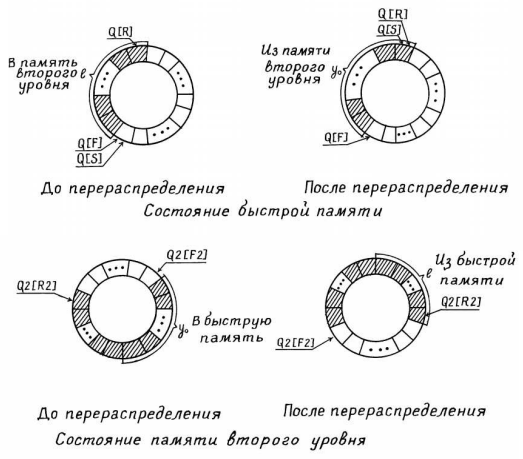

Рис. 7

Возможна ситуация, когда в памяти второго уровня находится элементов меньше, чем y_0. Тогда недостающие элементы мы берем из самых старых элементов в быстрой памяти, не выталкивая их в память второго уровня. Ниже представлено состояние очереди (рис. 8), если в ситуации, изображенной на рисунке 4, было произведено 3 исключения, 4 включения и еще 2 исключения (последнее исключение должно удалить элемент 5, и оно вызывает перераспределение очереди). Надо отметить, что в данном случае в памяти второго уровня у нас остается только один элемент. Такие крайние случаи можно исключить, если разрешить алгоритму оставлять в быстрой памяти количество элементов, отличающееся от оптимального значения y_0 на некоторую небольшую константу.

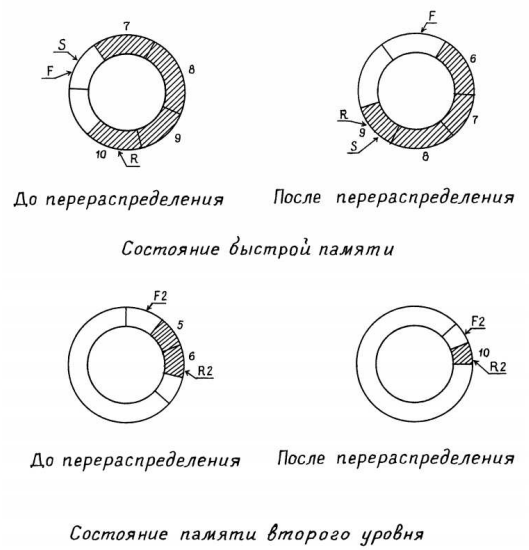

Рис. 8

Таким образом, мы всегда будем после перераспределения иметь y_0 начальных элементов в быстрой памяти, чтобы можно было производить исключения из части очереди, находящейся в быстрой памяти, и в то же время часть быстрой памяти оставлять свободной, чтобы можно было и включать новые элементы в быструю память. Мы считаем, что очередь почти всегда (за исключением, возможно, начала и конца работы) больше доступного участка быстрой памяти, и, значит, в памяти второго уровня почти всегда будет храниться очередь, состоящая из средних по времени жизни элементов.

Нашей задачей будет определение такого значения y_0, чтобы среднее число перераспределений очереди было минимальным, т. е. чтобы среднее время работы между перераспределениями очереди было максимальным.

3. Математическая модель

Обозначим через y текущее число начальных элементов в быстрой памяти, а через x — текущее число новых элементов, вставленных в конец части очереди, находящейся в быстрой памяти. Пусть

p обозначает вероятность включения нового элемента в очередь, q — вероятность исключения элемента из очереди, где $p + q = 1$, $m = n - 1$ — максимальное число элементов в очереди, которое на 1 меньше, чем число свободных мест в быстрой памяти.

Тогда в качестве математической модели мы будем иметь двумерное случайное блуждание в области $x \geq 0$, $y \geq 0$, $x + y \leq m$ (рис. 9).

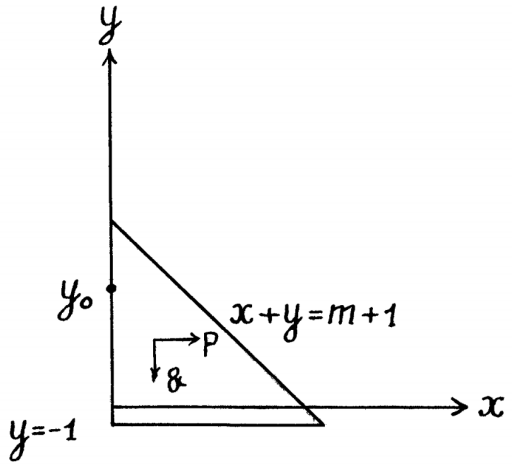

Рис. 9

Блуждание начинается в точке $y = y_0, x = 0$. В каждый момент времени находясь в точке (x, y), мы можем попасть в точку $(x+1, y)$ с вероятностью p и в точку $(x, y-1)$ с вероятностью q. Блуждание имеет два поглощающих барьера $x + y = m + 1$ и $y = -1$. Попадание на первый барьер означает переполнение быстрой памяти, а на второй барьер — исчерпание числа начальных элементов в быстрой памяти.

Рассмотрим более подробно модель (рис. 10) для конкретного значения $m = 2$.

Вычислим значения среднего времени блуждания до переполнения для различных возможных значений $y_0 = 0, 1, 2$; для $p = q = 1/2$. Для этого нам надо будет подсчитать число путей из начальной точки до поглощения на одном из поглощающих барьеров. Будем обозначать путь буквами П и Н, что будет означать переходы за

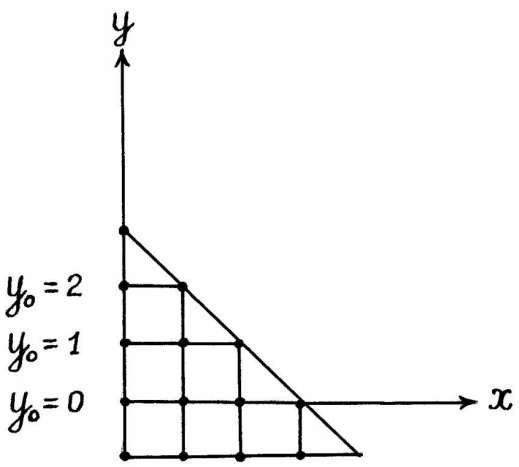

Рис. 10

один шаг соответственно вправо и вниз.

Тогда для начального состояния $y_0 = 0$ легко видеть, что мы имеем один путь до поглощения длины 1 — Н, один путь длины 2 — ПН, и два пути ППН и ППП длины 3. Вероятность пути длины n в нашем случае будет равна p^n. Следовательно, среднее время до поглощения будет вычисляться по формуле $T(0,0) = 1 \times 0.5 + 2 \times 0.5^2 + 2 \times 3 \times 0.5^3 = 1.75$. Для начального состояния $y_0 = 1$ будем иметь два пути НН и ПП длины 2, два пути ПНН и НПН длины 3 и четыре пути НППН, НППП, ПНПН и ПНПП длины 4. Среднее время $T(0,1) = 2 \times 2 \times 0.5^2 + 2 \times 3 \times 0.5^3 + 4 \times 4 \times 0.5^4 = 2.75$. И для блуждания с началом в точке $y_0 = 2$ будем иметь один путь длины 1 — П, два пути ННН и НПП длины 3, два пути НПНН, ННПН длины 4, и четыре пути НПНПН, НПНПП, ННППН, ННППП длины 5. Среднее время $T(0,2) = 1 \times 0.5 + 2 \times 3 \times 0.5^3 + 2 \times 4 \times 0.5^4 + 4 \times 5 \times 0.5^5 = 2.375$. В данном случае оптимальное начальное состояние $y_0 = 1$, так как в нем мы имеем максимальное среднее время блуждания до поглощения $T(0,1) = 2.75$.

В общем случае задача формулируется следующим образом: какое значение y_0 следует выбрать, чтобы среднее время блуждания до поглощения было максимальным?

Будем для вычисления среднего времени блуждания использовать метод разностных уравнений [16]. Обозначим через $T(x,y)$

среднее время блуждания до поглощения, если начальной точкой блуждания была точка (x, y). Тогда

$$T(x, y) = pT(x + 1, y) + qT(x, y - 1) + 1,$$

так как в этом случае с вероятностью p мы переходим в состояние $(x + 1, y)$, и тогда среднее время будет равно $T(x + 1, y)$; с вероятностью q переходим в состояние $(x, y - 1)$, и тогда среднее время будет равно $T(x, y-1)$. При этом в любом случае пройдет один шаг до переполнения памяти.

Попав на границы области блуждания, мы уже не двинемся ни на один шаг, и, следовательно, граничные условия будут иметь вид:

$$T(x, -1) = 0, T(x, m - x + 1) = 0.$$

Нашей задачей является нахождение такого значения y_0, чтобы $T(0, y_0)$ было максимальным.

Алгоритм для вычисления функции $T(0, y_0)$, очевидно, может быть такой. Мы видим, что $T(m, 0) = pT(m + 1, 0) + qT(m, -1) + 1 = 1$. Теперь можно последовательно вычислить, идя от больших значений аргументов к меньшим, значения функции $T(x, y)$ для точек, лежащих на прямых $y = 0, y = 1, \ldots y = m - 1, y = m$.

4. Результаты вычислений

Был предложен алгоритм и программа на C++, решающие данную задачу. Приведем вычисленные оптимальные значения числа начальных элементов, оставляемых в очереди y_0, и значения среднего времени до переполнения при оптимальных значениях параметра $T(y_0)$ для некоторых значений p и q при нескольких значениях m (табл. 1).

Вычисления показали, что в случае $p = q = 1/2$, $y_0 \neq m/2$, как могло показаться на первый взгляд, а $y_0 \approx 0.7m$, хотя вид точной формулы для произвольного m неясен. Например, для $m = 100$, $y_0 = 82$.

Приведены оптимальные значения y_0 для p, изменяющегося от 0 до 1 с шагом 0.1 при $m = 20$ (табл. 2).

Интересно отметить, что максимальное значение среднего времени блуждания до перераспределения памяти будет при $p = 0.4, y_0 =$

Таблица 1

$p=q=1/2$			$p=2/3, q=1/3$		$p=1/3, q=2/3$	
m	y_0	$T(y_0)$	y_0	$T(y_0)$	y_0	$T(y_0)$
1	1	1.75	0	1.67	1	1.89
2	1	2.75	1	2.74	2	2.73
3	2	3.84	1	3.63	2	3.81
4	3	4.83	2	4.75	3	5.00
5	3	6.05	2	5.76	4	6.18
6	4	7.27	3	6.92	5	7.34
7	5	8.42	3	8.00	6	8.5
8	5	9.68	4	9.18	7	9.65
9	6	11.01	4	10.31	7	10.97
10	7	12.29	5	11.52	8	12.3
11	8	13.53	5	12.69	9	13.62
12	8	14.93	6	13.91	10	14.95
13	9	16.32	6	15.11	11	16.27
14	10	17.66	7	16.35	12	17.59
15	11	18.97	7	17.57	13	18.91
16	11	20.43	8	18.82	14	20.22
17	12	21.87	8	20.06	15	21.54
18	13	23.28	9	21.32	15	22.86
19	14	24.66	9	22.58	16	24.27
20	14	26.12	9	22.58	16	24.27

Таблица 2

p	y_0	$T(y_0)$
0	20	21
0.1	19	21.95
0.2	19	23.49
0.3	18	25.26
0.4	16	26.27
0.5	14	26.12
0.6	12	24.89
0.7	9	23.32
0.8	6	21.8
0.9	3	20.45
1.0	0	21

16. Вычисления для других значений параметров показали, что $T(y_0)$ получает максимальное значение при $p \approx 0.45$. Так для $m = 100$ максимальное значение $T(y_0) = 156.17$ достигается при $p = 0.45, y_0 = 88$. В некотором смысле при таких значениях вероятностей включения и исключения элементов оптимальное управление очередью может быть наиболее эффективным.

5. Заключение

В случае, если мы работаем с несколькими FIFO-очередями в общей быстрой памяти и известно, что переполнение очередей возможно, мы можем разбить быструю память на части и в каждой из частей работать по рассмотренному принципу. Если известны вероятностные характеристики очередей, то можно разделить память между очередями так, чтобы максимизировать среднее время до переполнения памяти [15]. Остается нерешенной проблема будет ли лучше такой способ работы в сравнении с методом представления FIFO-очередей в классической кэш-памяти.

Список литературы

[1] *Кнут Д.* Искусство программирования для ЭВМ. Т. 1. М.: Вильямс, 2001. 736 с.

[2] *Седжвик Р.* Фундаментальные алгоритмы на C++. К.: Диасофт, 2001. 688 с.

[3] *Баранов С.Н.* Язык Форт в СССР и России. Виртуальный компьютерный музей URL: http://www.computer-museum.ru/histsoft/fortran_sorucom_2011

[4] *Баранов С.Н., Ноздрунов Н.Р.* Язык Форт и его реализации. — Л.: Машиностроение, Ленинградское отделение, 1988. 156 с.

[5] *Бураго А.Ю., Кириллин В.А., Романовский И.В.* ФОРТ — язык для микропроцессоров. Л.: Знание, 1989. 36 с.

[6] *Koopman P.* Stack Computers. Ellis Horwood, 1989. URL: http://www.cs.cmu.edu/ koopman/stackcomputers/

[7] *Катеванис М.Г.Х., Секуин К.Х., Паттерсон Д.А., Шербурн Р.У.* RISC: Эффективные архитектуры для СБИС-компьютеров. Электроника СБИС. Проектирование микроструктур. — М.:Мир, 1989.

[8] *Hasegava M., Shigei Y.* High-speed top-of-stack scheme for VLSI processor: a management algorithm and its analysis // Proceedings of 12th Symposium on Computer Architecture. June. 1985. P. 48–54.

[9] *Stanley T., Wedig R.* A performance analysis of automatically managed top of stack buffers // Proceeding of 14th Symposuim on Computer Architecture. June. 1987. P. 272-281.

[10] *Аксенова Е.А., Лазутина А.А., Соколов А.В.* Исследование немарковской модели управления стеком в двухуровневой памяти // Программирование. № 1. 2004. С. 37–46.

[11] *Аксенова Е.А., Соколов А.В.* Оптимальное управление двумя параллельными стеками в двухуровневой памяти // Дискретная математика. Том 19, выпуск 1. М. 2007. С. 67–75.

[12] *Гюнтер М.* (Wind River) Оптимизированное ПО для многоядерных процессоров обеспечивает прорыв в пропускной способности сетей // МКА: ВКС. № 4. 2012. С. 38–47. www.mka.ru

[13] *Егоров В.* Многоядерные интегрированные сетевые процессоры высокой пропускной способности // Электронные компоненты. № 7. 2009. С. 29–33. http://www.russianelectronics.ru/leader-r/review/2192/doc/46335/

[14] *Михновский С.Д., Шор Н.З.* Оценка минимального числа пересылок при динамическом распределении страничной памяти // Кибернетика. 1965. № 5. С. 18–21.

[15] *Драц А.В., Соколов А.В.* Оптимальное размещение в памяти одного уровня n стеков и/или очередей // Стохастическая оптимизация в информатике. Вып.4. 2008. С. 72–89.

[16] *Феллер В.* Введение в теорию вероятностей и ее приложения. — М.: Мир. 1964.