ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №1(32), 2017, с. 83–103

УДК 004.258+004.942

Е. А. Барковский, Р. И. Кучумов, А. В. Соколов

Оптимальное управление двумя work-stealing деками в общей памяти при различных стратегиях перехвата работы

Аннотация. В параллельных балансировщиках задач, работающих по стратегии work-stealing, каждый процессор имеет свой дек (deque) задач. Один конец дека используется только владельцем для добавления и извлечения задач, а другой — для перехвата другими процессорами.

Целью работы является построение и анализ математических моделей процесса работы с двумя циклическими деками, расположенными в общей памяти. Параметрами этих моделей являются вероятности операций на каждом шаге дискретного времени (возможно как последовательное, так и параллельное выполнение операций). Модели строятся в виде случайных блужданий по целочисленной решетке на плоскости. На основе вышеупомянутых моделей решены задачи оптимального разделения памяти при некоторых стратегиях перехвата элементов. В качестве критерия оптимальности рассматривается максимальное среднее время до переполнения памяти.

Проведены статистические исследования по оценке вероятностей операций работы с деками для нескольких типов задач, выполняемых в реализованном балансировщике. Для полученных вероятностей операций работы с деками проведены численные эксперименты по анализу разработанных моделей.

Ключевые слова и фразы: work-stealing балансировщики, work-stealing деки, структуры данных, цепи Маркова, случайные блуждания.

Введение

Стратегии балансировки параллельных вычислений разделяют на статические и динамические. Статическая балансировка используется, когда известно практически все про выполняемые задачи. В этом случае можно заранее определить оптимальное расписание задач (например, минимизировать среднее время решения). Такие задачи

Работа поддержана грантом РФФИ №15-01-03404-а.

считаются NP-полными и встречаются достаточно редко. При динамической балансировке балансировщик во время работы использует некую относительно простую стратегию балансировки, которая дает результат, близкий к оптимальному.

На сегодняшний день существует два основных способа динамической (не зависящей от конкретной задачи) балансировки многопроцессорных параллельных вычислений на многоядерной архитектуре [1, 2]: work-sharing и work-stealing. В первом методе балансировщик передает задания от наиболее загруженного процессора к наименее нагруженному. Метод work-stealing имеет другой подход. Процессоры, ставшие пустыми, пытаются перехватить часть работы у других процессоров. Этот метод реализован в большом числе систем, например: Cilk, Cilk++; TBB; TPL; X10 и другие.

В этом методе балансировки нагрузки каждый процессор решает ряд задач, указатели на которые хранятся в деке (deque) этого процессора. Когда процессор создает новую задачу, он добавляет указатель на задачу в свой дек; когда процессору нужна задача, он берет указатель на задачу из вершины дека. Если процессор узнает, что его дек пуст, он перехватывает указатели на задачи у другого процессора. Первые две операции выполняются как в LIFO-стеке, а перехват происходит из основания дека — как в FIFO-очереди. В терминологии Д. Кнута это означает, что мы работаем с деком с ограниченным входом [3]. В [1] рассматривался перехват одного элемента, в [4] — половины элементов.

Существуют разные способы представления деков с ограниченным входом в памяти. Можно воспользоваться методом «связное представление». Модель такого метода будет схожа с уже построенной моделью связного представления стеков и очередей [5]. В [6] был предложен и проанализирован метод представления стеков и очередей в виде связного списка массивов (страничное представление). В [7] такой метод был предложен для деков. В [8] была предложена модель work-stealing балансировщика. Она была построена на основе аппарата теории массового обслуживания, но в ней не рассматривался какой-либо конкретный способ представления деков в памяти.

В этой работе мы предлагаем математическую модель последовательного циклического представления work-stealing деков (как FIFO-очередей), где каждый дек расположен в отдельном участке памяти [9]. На каждом шаге дискретного времени с ними могут произойти операции с заданными вероятностями. Ранее такие модели были построены нашим коллективом для представления некоторых динамических структур данных: стеки, очереди, приоритетные очереди [10–12] и др.

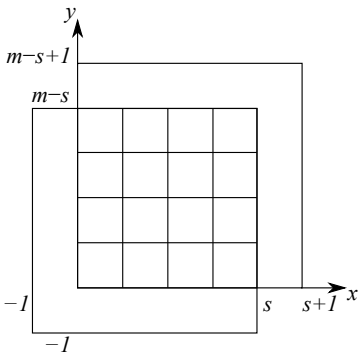

Рис. 1. Область блуждания

В статье будет рассматриваться работа двух деков. Этот конкретный случай важен не только как первый шаг к построению общей модели, но у него есть и свое независимое значение. Так, среди многоядерных архитектур есть такие, в которых отсутствует кэш память. Для примера, в архитектуре AsAP-II каждое ядро имеет два FIFO буфера, а в архитектуре SEAforth — два стека (для хранения данных и адресов возврата) [**13**]. В этих архитектурах очереди и стеки реализованы циклически и разделены друг от друга, также допускается потеря элементов при переполнении. Work-stealing деки могут быть реализованы аппаратно, по принципу вышеупомянутых архитектур. В этом случае важной задачей является исследование оптимального управления двумя деками. В случае произвольного количества ядер можно конструировать требуемые микросхемы из «двудековых».

1. Математическая модель

Пусть в памяти размера m мы работаем с двумя последовательными циклическими деками, где все элементы имеют фиксированный размер в одну структурную единицу. Для последовательного представления деков выделим каждому некоторое количество единиц памяти из общего объема m. Пусть s — количество единиц памяти, выделенное первому деку, тогда $m - s$ — количество памяти, выделенное второму деку. Здесь нам понадобятся введённые в [**22**] обозначения.

Пусть x и y текущие длины первого и второго дека соответственно. В качестве математической модели мы рассматриваем случайное блуждание по двухмерной целочисленной решетке в области $-1 \leq x \leq s+1$, $-1 \leq y \leq m-s+1$ (рис. 1).

Некоторые вероятностные характеристики операций, производимых над деками, заранее известны:

p_1 — вероятность включения элемента в первый дек,
p_2 — вероятность включения элемента во второй дек,
p_{12} — вероятность включения элементов параллельно в оба дека,
q_1 — вероятность исключения элемента из первого дека,
q_2 — вероятность исключения элемента из второго дека,
q_{12} — вероятность исключения элементов параллельно из обоих деков,
pq_{12} — вероятность включения в первый дек и исключения из второго,
pq_{21} — вероятность включения во второй дек и исключения из первого,
r — вероятность сохранения длины деков (чтение или отсутствие операции).

При этом $p_1 + p_2 + p_{12} + q_1 + q_2 + q_{12} + pq_{12} + pq_{21} + r = 1$. Как только дек становится пустым, он начинает перехватывать элементы у другого дека. Соответственно, исключение элемента из пустого дека не является аварийной ситуацией.

Блуждание начинается в начале координат. Прямые $x = s + 1$ и $y = m - s + 1$ образуют два поглощающих экрана — когда процесс попадает на эти экраны, память переполняется и программа аварийно прекращается. Прямые $x = -1$, $y = -1$ образуют два отражающих экрана. Введение данных экранов позволяет установить процесс перехвата элемента. Формально деки попадают на экраны, но фактически — в область $1 \le x < s$, $1 \le y < m - s$. Конкретная точка перехода зависит от стратегии перехвата.

Для случая перехвата одного элемента: если процесс находится на точках $(x, -1)$ или $(-1, y)$, он перейдет на точки $(x - 1, 1)$ или $(1, y - 1)$. Для случая перехвата половины элементов: процесс перейдет на точки $(x/2, x/2)$ или $(y/2, y/2)$ соответственно. Если x или y являются нечетными, мы округляем до нужного целого. Для случая перехвата произвольного количества элементов (o): процесс перейдет на точки $(x - o, o)$ или $(o, y - o)$.

Если размер памяти пустого дека не позволяет перенести все элементы (или у непустого дека количество элементов меньше или равно o), то мы перехватываем только один элемент. В будущем можно рассмотреть и другие способы построения модели оптимального перехвата, например, перехватывать половину дека в таких ситуациях.

Выбор критерия оптимальности определяется областью применения. В сетевых устройствах (маршрутизаторах) потеря элементов при переполнении является обычной ситуацией, и в этом случае мы рассматривали критерий оптимальности — минимальная средняя доля потерянных при переполнении элементов.

В программных или аппаратных системах, решающих вычислительные задачи, потеря элементов является аварийной ситуацией. В таком случае критерием оптимальности должно быть максимальное среднее время до переполнения. В случае переполнения требуется перераспределить память, завершить работу программы, или приостановить выполнение потока, пока не появится свободная память.

Здесь в качестве критерия оптимальности рассматривается максимальное среднее время до переполнения памяти (т. е. до попадания на прямые $x = s + 1$ и $y = m - s + 1$).

Так, в задачах оптимального разделения памяти, нужно найти такое s, при котором система будет работать дольше; для произвольного количества элементов для перехвата — значение o. Для решения этих задач используются результаты теории поглощающих цепей Маркова [14]. Был найден вид переходной матрицы P и ее подматрицы Q, описывающей поведение процесса до выхода из множества невозвратных состояний. Эта подматрица требуется для построения фундаментальной матрицы поглощающей цепи $N = (I - Q)^{-1}$. Сумма элементов матрицы N в строке, соответствующей начальному состоянию, является средним временем до поглощения (переполнения), если процесс начал работу из нуля $(x, y) = (0, 0)$.

Так как для каждого значения s и o мы имеем свою цепь Маркова, то можно сказать, что мы решаем задачу нахождения оптимальной цепи Маркова в смысле указанного критерия оптимальности или, другими словами, задачу целочисленного нелинейного программирования, где функция критерия оптимальности задается алгоритмически.

2. Матрица переходов случайного блуждания

Рассмотрим задачу оптимального разделения памяти, случай перехвата одного элемента. Представим случайное блуждание в виде поглощающей марковской цепи. Пусть матрица переходов [14]

$$P = \begin{bmatrix} Q & R \\ O & I \end{bmatrix}$$

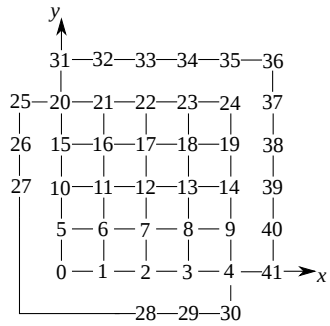

Рис. 2. Нумерация состояний при $m = 8$, $s = 4$

состоит из блоков Q невозвратных состояний, R, описывающего переход процесса из невозвратных состояний в поглощающие, единичной и нулевой матриц.

Определим нумерацию состояний, как показано на рис. 2. Сначала мы нумеруем состояния в области $0 \le x \le s$, $0 \le y \le m - s$. Затем состояния, в которых происходят перехваты — отражающий экран ($x = -1$, $y = -1$), и поглощающие состояния (прямые $x = s + 1$ и $y = m - s + 1$).

Утверждение 1. *Блок Q матрицы P имеет структуру*

$$Q = \begin{bmatrix} Q_1 & Q_2 \\ Q_3 & Q_4 \end{bmatrix},$$

где:

- *подматрица Q_1 описывает блуждание в области $x \le s$, $y \le m-s$;*
- *подматрица Q_2 описывает переходы из области $x \le s$, $y \le m - s$ на отражающий экран;*
- *подматрицы Q_3 и Q_4 описывают переходы с отражающего экрана.*

Лемма 1. *Матрица Q_1 размера $(s+1)(m-s+1)$ имеет следующий вид:*

$$Q_1 = \begin{bmatrix} D & A & O & \dots & O \\ B & C & A & O & \dots \\ O & B & C & \ddots & \dots \\ \vdots & \vdots & \ddots & \ddots & A \\ O & O & \dots & B & F \end{bmatrix},$$

где A, B, C, D, F являются квадратными подматрицами размера $(s+1)$, O — нулевая подматрица.

Докажем лемму 1 с помощью математической индукции.

Доказательство. (*1*) База индукции. Пусть общий объем памяти равен $m = 2$, $s = 1$ — количество памяти, выделенное первому деку. Тогда размер матрицы Q_1 будет 4×4. Подматрицы A, B, C, D, F имеют размер 2×2 (подматрица C здесь не представлена ввиду маленького значения памяти m). Матрица Q_1 имеет следующую форму:

$$Q_1 = \left[\begin{array}{cc|cc} q_1+q_2+q_{12}+r & p_1+pq_{12} & p_2+pq_{21} & p_{12} \\ q_1+q_{12} & q_2+r & pq_{21} & p_2 \\ \hline q_2+q_{12} & pq_{12} & q_1+r & p_1 \\ q_{12} & q_2 & q_1 & r \end{array}\right].$$

(*2*) Индуктивное предположение. Предположим, что для размера памяти m лемма 1 верна. Размерность Q_1 будет $(s+1)(m-s+1)$, размерность подматриц — $(s+1)$.

(*3*) Индуктивный переход. Проверим, что при размере памяти $(m+1)$ лемма 1 верна. Так как добавилась еще одна единица памяти, то увеличился размер одного из деков. Рассмотрим два случая:

(а) Единица памяти добавлена в первый дек, размеры m и s увеличились на единицу: $m = m+1$, $s = s+1$. Тогда область случайных блужданий увеличится по оси OX – будет добавлено $(s+1)$ новых состояний. Размерность матрицы Q_1 увеличится на $(m-s+1)$, размерность всех подматриц увеличится на единицу, и их количество останется прежним.

Пусть размерность подматриц будет $(s+2)$. Проследим за изменением структуры на примере подматрицы A. К ней будут добавлены одна строка и один столбец, и ее вид будет следующим:

$$A_{(s+2)\times(s+2)} = \left[\begin{array}{ccc|c} & A_{(s+1)\times(s+1)} & & \begin{array}{c}0\\ \vdots \\ p_{12}\end{array} \\ \hline 0 & \dots & pq_{21} & p_2 \end{array}\right].$$

Общий вид подматрицы не изменится. Аналогично это можно показать и для остальных подматриц.

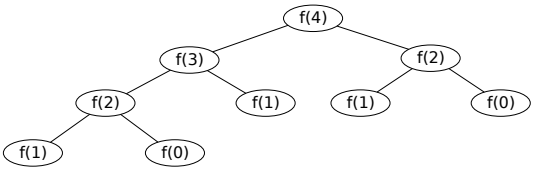

Рис. 3. Граф задач для вычисления четвертого числа Фибоначчи

(b) Единица памяти была добавлена во второй дек. Размеры m и s станут $m = m+1$, $s = s$. Область блуждания поднимется вверх по оси OY, т. е. будет добавлено $(m - s + 1)$ новых состояний. Размерность матрицы Q_1 увеличится на $(s + 1)$. Размерности подматриц не изменятся, но количество подматриц A, B, C увеличится на один, т. е. общий вид матрицы Q_1 сохранится.

Таким образом, в обоих случаях матрица Q_1 не изменяет свою структуру — лемма 1 доказана. □

Используя этот метод можно доказать, что и остальные подматрицы Q сохраняют свой вид. Утверждение 1 доказано.

Вид матрицы в оставшихся задачах доказывается подобным образом.

В пункте 3 статьи будет описан балансировщик, обладающий функцией сбора статистики, в результате чего мы получаем вероятности выполнения операций с деками (включение элемента, исключение и т.д.) для некоторых классов задач. Полученные вероятности являются входными данными для построенной математической модели и будут использованы в пункте 4, где приводятся результаты численных экспериментов с разработанной моделью.

3. Алгоритм работы балансировщика

Задачи, выполняемые балансировщиком, состоят из инструкций, которые должны быть выполнены последовательно, но во время своей работы они могут создавать подзадачи, которые можно выполнять параллельно с родительской. Таким образом, задачи можно представить в виде графа, в котором каждая задача указывает на создавшую её родительскую задачу.

Например, задачу для вычисления чисел Фибоначчи можно записать (на псевдокоде) следующим образом:

Pseudocode

```
function f(n)
    if n < 2 then
        return n;
    end if
    x = spawn f(n - 1);
    y = spawn f(n - 2);
    sync;
    return x + y;
end function
```

При выполнении $\mathtt{f}(n)$ задача создаст две подзадачи: $\mathtt{f}(n-1)$ и $\mathtt{f}(n-2)$ (случай $n = 4$ см. на рис. 3). Созданные подзадачи будет помещены в очередь потока и могут быть украдены и выполнены другими потоками.

Для того, чтобы дождаться завершения подзадач, вызывается функция *sync*, которая передает управление балансировщику, позволяя ему выполнять другие задачи, пока у родительской не завершатся все подзадачи. Задачи для выполнения поток сначала извлекает из своей очереди, а когда они заканчиваются, начинает красть из очереди случайно выбранного потока.

3.1. Очереди задач

Так как над очередью задач могут выполняться операции добавления и извлечения только ее владельцем, а операция перехвата — другими потоками, то в реализациях балансировщиков используют очереди с двумя концами, т. н. деки (double ended queues).

В одной из первых работ [15] для хранения задач используется массив фиксированного размера, а операции над концами деков выполняются с неблокирующими синхронизациями. Для этого вместе с индексами концов хранятся счетчики количества их изменений, и каждый поток их обновляет, используя атомарное сравнение с обменом. Основной недостаток этой реализации — это отсутствие динамического изменения размера.

В работе [7] для решения этой проблемы предлагают использовать двусвязные списки вместо массивов, но в результате появляются накладные расходы при работе со списками.

В одной из последних и наиболее цитируемых работ [9] предложена реализация деков на основе циклического массива, с неблокирующими синхронизациями операций и с возможностью динамического

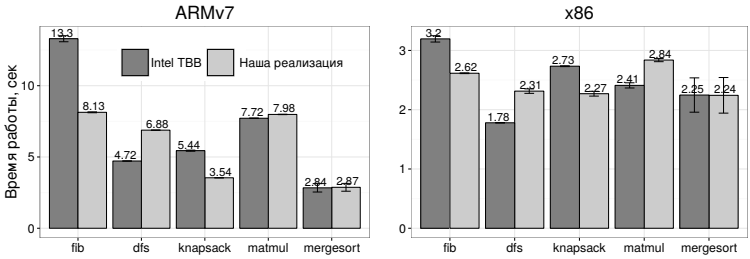

Рис. 4. Сравнение среднего времени работы балансировщиков на процессорах с архитектурами ARMv7 и x86

изменения размера. Относительно других реализаций она нуждается в меньших накладных расходах на выполнение операций и по использованию памяти. Работающие версии можно встретить в балансировщиках задач в библиотеке Intel TBB и в языке Rust. В [16] приводится её модификация для слабых моделей памяти.

3.2. Реализация балансировщика

Нами был реализован на языке C++ 11 work-stealing балансировщик задач, работающий по описанному выше алгоритму и использующий реализацию деков из [16]. (Исходные коды и реализация на сайте https://github.com/rkuchumov/staccato). В [17] описана реализация балансировщика, а здесь мы даем его краткое описание с целью пояснить возможности балансировщика по проведению статистических исследований по оценке вероятностей операций работы с деками с использованием терминологии, рисунков и пояснений из [17]"

Разработанный балансировщик сравнивался с балансировщиком задач из Intel TBB, в котором используется такой же алгоритм планирования и реализация деков. Сравнения производились на Intel Pentium N3530 2.16GHz, 4 CPU, 64 L1, с Linux 4.4.0. и на ARMv7l rev 5, 4 CPU, 32 L1, с Linux 4.4.8 при одинаковых параметрах компиляции (рис. 4). В таблице 1 показаны тестовые задачи, использованные при сравнении.

3.3. Сбор статистики работы балансировщика

Для анализа работы балансировщика были добавлены счетчики количества увеличений и уменьшений индексов концов (B_i^+, B_i^- и T_i^+) и счетчики операций создания, завершения и перехвата задач. Для

Таблица 1. Используемые тестовые задачи

Тест	Размерность	Описание
fib	35	Числа Фибоначчи по рекуррентной формуле
knapsack	34	Задача о рюкзаке методом ветвей и границ
matmul	256×256	Перемножение матриц
mergesort	2^{24}	Сортировка слиянием
dfs	3×300	Обход графа задач

профилирования изменения размеров деков на каждом интервале времени (τ_j) был создан специальный поток, который сохранял значения счетчиков в вместе с моментами времени работы балансировщика (t_i).

Время определялось с помощью процессорной команды RDTSCP, которая позволяет получить количество тактов работы процессора. При этом сама команда выполняется за десятки тактов [18]. Для минимизации недетерминизма при параллельной работе балансировщик запускался на ядрах процессора, изолированных от работы балансировщика потоков операционной системы.

Интервал времени $d\tau$ выбирался не больше минимальной разности между соседними моментами времени. Если между t_i и t_{i+1} произошло k интервалов времени и значения счетчика изменялось на n единиц, то на каждом таком интервале времени предполагалось, что значения изменялось равномерно, т. е. на n/k. Тогда размер дека на каждом интервале времени вычислялся как $u(\tau_j) = B^+_{\tau_j} - B^-_{\tau_j} - T_{\tau_j}$.

Далее изменение размеров (du) деков округлялось и определялось среднее количество изменений на $0, \pm1, \pm2$ и т.д. единиц.

Вероятности изменения размеров для задачи перемножения матриц и задачи о рюкзаке при интервале времени $d\tau = 100$ тактов приведены в таблице 2 (при реализации задачи о рюкзаке использовался алгоритм из работы [19]).

Для получения вероятностей, используемых в модели, нужно вычислить произведения всех вероятностей изменения на 0 и ±1 единицу, считая что изменения размеров деков — независимые случайные величины. Например, вероятность включения в первый дек (p_1) равняется произведению вероятностей изменения первого дека на 1 и второго на 0. Изменения размеров на ±2 и больше единиц можно не учитывать, т. к. их вероятности блики к нулю.

Таблица 2. Вероятности изменения размеров дека

l	-2	-1	0	1	2
Задача перемножения матриц					
Дек 1	0.0000047	0.0976776	0.8046355	0.0976741	0.0000080
Дек 2	0.0000000	0.1343020	0.7313987	0.1342942	0.0000044
Задача о рюкзаке					
Дек 1	0.0000000	0.0282798	0.9434404	0.0282795	0.0000002
Дек 2	0.0000000	0.0538008	0.8924009	0.0537951	0.0000031

Таблица 3. Входные данные тестовых задач

Задача	p_1	p_2	q_1	q_2	p_{12}	q_{12}	pq_{12}	pq_{21}
Умножение матриц	0.071	0.108	0.071	0.108	0.014	0.014	0.013	0.013
О рюкзаке	0.025	0.050	0.025	0.050	0.002	0.002	0.002	0.002

4. Некоторые примеры численного анализа

На основе вышеописанных математических моделей разработаны программы, генерирующие матрицу переходных вероятностей P (в этих программах задачи решаются с помощью теории поглощающих цепей Маркова), также были построены имитационные модели изучаемых процессов.

В следующих таблицах приведены некоторые результаты вычислений. Исходными данными являются оценки вероятностей (по частоте) работы системы при решении задачи перемножения матриц и задачи о рюкзаке, приведённые в таблице 3. Они были получены в результате экспериментов с разработанным work-stealing балансировщиком задач [17].

Так как аналитическое решение не было получено, нам нужно решать задачи для конкретных m и $m = 100$ используется для примера.

В таблице 4 рассматривается оптимальное разделение памяти в сравнении с разделением пополам ($s = 50$). Судя по результатам, можно заключить следующее: если мы разделяем память оптимально, система работает дольше. Для примера, когда дек перехватывает один элемент, разница во времени до переполнения между оптимальным разделением и делением пополам составляет 204 для задачи

Таблица 4. Среднее количество операций до переполнения дека при $m = 100$

Количество элементов для перехвата	Разбиение памяти пополам ($s = 50$)	Оптимальное разбиение памяти
Задача перемножения матриц		
Один	11345	11549 ($s = 46$)
Половина	11989	12165 ($s = 47$)
$o = 16$	11954	
Задача о рюкзаке		
Один	30736	32902 (s = 43)
Половина	32722	34594 (s = 45)
$o = 16$	32704	

перемножения матриц и 2166 для задачи о рюкзаке, то есть система работает на 204 (или 2166) операции дольше, если память разделяется оптимально. Если дек перехватывает половину элементов (при вышеприведенных задачах), разницы во времени составляют 176 и 1872. В наших экспериментах перехват половины элементов оказался более выгодной стратегией в сравнении с перехватом одного элемента.

В нижней части таблицы 4 перехват произвольного количества элементов сравнивается с перехватом одного и половины элементов. Так, для задачи перемножения матриц, перехватывая 16 элементов за раз, система работает на 609 операций дольше, чем при перехвате одного элемента; для задачи о рюкзаке, перехватывая 18 элементов за раз — на 1968 операций дольше. В целом, перехват произвольного количества элементов (при вышеописанной постановке) дает результаты, близкие к перехвату половины элементов, немного уступая им.

5. Возможные способы управления памятью для деков

На практике, в том числе и в описанной в данной статье реализации балансировщика, для работы с деками в виде циклических массивов часто используются классические методы работы с кучей в трансляторах C/C++. Для каждого дека из кучи запрашивается массив. Когда он становится достаточно большой, то запрашивается новый массив, например, в два раза больший — туда копируются элементы дека, а старый массив отдается в список свободных блоков кучи. Если же наоборот, дек стал достаточно мал, то те же действия

производятся с массивом, меньшим старого в два раза [9]. Мы же в этой статье рассматриваем задачу поиска оптимальных альтернативных методов управления памятью для деков, в предположении, что известны некоторые их вероятностные характеристики.

Понятно, что перераспределять свободную память между деками осмысленно только тогда, когда ее достаточно много. Здесь работают те же аргументы, что и в анализе алгоритма Гарвика в [3]. Этот алгоритм, предназначенный для последовательного представления n стеков, при переполнении одного из стеков производит полное перераспределение свободной памяти. При этом 10% свободной памяти делится поровну между стеками, а 90% — пропорционально росту стеков с момента предыдущего перераспределения. В принципе этот алгоритм можно применить и для работы с деками.

Заметим, однако, что алгоритмы, подобные алгоритму Гарвика рассчитаны на то, что структуры данных продолжают в будущем вести себя также, как и в наблюдаемом настоящем. Но где гарантия, что дек, в который было больше всего включений, как раз и не начнет убывать? Мы же предполагаем, что предварительный статистический анализ исследуемой системы позволит выявить такие промежутки времени, когда вероятности операций практически не меняются. Тогда на каждом таком участке мы можем управлять деками оптимально. Например, правомерно предположить, что в начале работы программы вероятности включения в деки больше, чем вероятности исключения, затем они примерно равны, а в конце работы — больше вероятности исключения, так как обычно работа программы начинается и заканчивается пустым деком. Для использования на практике такая модель должна применяться несколько раз с разными матрицами переходов, в том случае, если статистические исследования выявили промежутки времени, на которых вероятности операций с деками практически не меняются (т.к. мы рассматриваем однородную цепь Маркова, когда вероятности не зависят от времени [14]). В [20] предложен модифицированный вариант алгоритма Гарвика, где при переполнении какого-либо стека производится локальное перераспределение части стеков, расположенных рядом с переполнившимся стеком. Такой подход должен быть полезен при многопотоковой работе с деками.

В модели мы рассматриваем случайное блуждание, начинающиеся из начала координат (в начале работы деки пустые). Нами также анализировались модели, в которых решалась задача не начального разделения памяти, а оптимального перераспределения памяти после переполнения [21]. Там начальной точкой блуждания является

та, в которой мы переполнялись, или точка полученная из нее за счет оптимального перехвата. Такую модель можно применять, если мы будем рассматривать участки времени с разными вероятностями операций с деками.

Заметим, что в некоторых конкретных задачах мы иногда могли бы получить верхние оценки размеров деков и, выделив каждому деку максимальный объем памяти, организовать работу без переполнения памяти. Но в этом случае при большом числе потоков суммарный объем верхних оценок может превзойти размер доступной памяти для деков и, кроме того, все деки не могут достигать верхних оценок одновременно. Поэтому мы в данной работе решаем задачу оптимального разделения памяти для деков, предполагая, что даже при известных верхних оценках мы не сможем организовать работу с деками без перераспределения.

В дальнейшем мы планируем разработку своего менеджера памяти для работы с work-stealing деками, основанного на предложенных нами моделях.

6. Заключение

Разработаны алгоритмы и программы на языке C для построения матриц переходных вероятностей P при произвольных значениях m, s, o и вероятностей операций; нахождения оптимального разбиения памяти между деками, в зависимости от вероятностных характеристик деков (эта задача была апробирована в [**22**]); нахождения оптимального количества элементов для перехвата. Критерий оптимальности — максимальное среднее время работы.

Для решения поставленных задач использовался аппарат управляемых случайных блужданий, поглощающих цепей Маркова, система LAPACK. Вычисления производились с помощью кластера КарНЦ РАН.

Список литературы

[1] R. D. Blumofe, C. E. Leiserson. “Scheduling Multithreaded Computations by Work Stealing”, *Journal of the ACM*, **46**:5 (1999), pp. 720–748. ↑ 84

[2] M. Herlihy, N. Shavit. *The Art of Multiprocessor Programming*, Elsevier, 2008, 536 p. ↑ 84

[3] D. Knuth. *The Art of Computer Programming*. V. 1, Addison-Wesley, 2005, v+134 p. ↑ 84,96

[4] D. Hendler, N. Shavit. “Non-blocking Steal-half Work Queues”, ACM Symposium on Principles of Distributed Computing (PODC 2002) (July 21–24, 2002, Monterey, California), pp. 280–289. ↑ 84

[5] A. V. Sokolov, A. V. Drac. “The Linked List Representation of n LIFO-Stacks and/or FIFO-Queues in the Single-Level Memory”, *Information Processing Letters*, **13**:19–21 (2013), pp. 832–835. ↑ 84

[6] Е. А. Аксенова, А. А. Лазутина, А. В. Соколов. «Об оптимальных методах представления динамических структур данных», *Обозрение прикладной и промышленной математики*, **10**:2 (2003), с. 375–376. ↑ 84

[7] D. Hendler, Y. Lev, M. Moir, N. Shavit. “A Dynamic-Sized Nonblocking Work Stealing Deque”, *Distributed Computing*, **18**:3 (2006), pp. 189–207. ↑ 84,91

[8] M. Mitzenmacher. “Analyses of Load Stealing Models Based on Differential Equations”, Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA’ 98) (Puerto Vallarta, Mexico, June 28–July 2, 1998), pp. 212–221. ↑ 84

[9] D. Chase, Y. Lev. “Dynamic Circular Work-Stealing Deque”, ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2005) (Las Vegas, Nevada, USA, July 18–20, 2005), pp. 21–28. ↑ 84,91,96

[10] E. A. Aksenova, A. A. Lazutina, A. V. Sokolov. “Study of a Non-Markovian Stack Management Model in a Two-Level Memory”, *Programming and Computer Software*, **30**:1 (2004), pp. 25–33. ↑ 84

[11] E. A. Aksenova, A. V. Sokolov. “The Optimal Implementation of Two FIFO-queues in Single-Level Memory”, *Applied Mathematics*, **2**:10 (2011), pp. 1297–1302. ↑ 84

[12] А. В. Соколов, А. В. Драц. «Моделирование некоторых методов представления n FIFO очередей в памяти одного уровня», *Эвристические алгоритмы и распределенные вычисления*, **1**:1 (2014), с. 40–52. ↑ 84

[13] А. В. Калачев. *Многоядерные процессоры*, Бином, М., 2010. ↑ 85

[14] Дж. Кемени, Дж. Снелл. *Конечные цепи Маркова*, Наука, М., 1960, 272 с. ↑ 87,96

[15] N. S. Arora, R. D. Blumofe, C. G. Plaxton. “Thread Scheduling for Multiprogrammed Multiprocessors”, ACM Symposium on Parallel Algorithms and Architectures (SPAA’ 98) (Puerto Vallarta, Mexico, June 28–July 2, 1998), pp. 119–129. ↑ 91

[16] N. M. Le, A. Pop, A. Cohen, F. Z. Nardelli. “Correct and efficient work-stealing for weak memory model”, ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP’ 13) (Shenzhen, China, February 23–27, 2013), pp. 69–80. ↑ 92

[17] Р. И. Кучумов. «Реализация и анализ work-stealing планировщика задач», *Стохастическая оптимизация в информатике*, **12**:1 (2016), с. 20–39. ↑ 92,94

[18] G. Paoloni. *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures*, White Paper 324264-001, Intel, 2010, URL: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf ↑ 93

[19] М. А. Посыпкин, И. Х. Сигал. «Комбинированный параллельный алгоритм решения задачи о ранце», IV Международная конференция «Параллельные вычисления и задачи управления» (Москва, Россия, 27–29 октября 2008 г.), с. 177–189. ↑ 93

[20] D. Y. Yeh, T. Munakata. "Dynamic initial allocation and local reallocation procedures for multiple stacks", *Communications of the ACM*, **29**:2 (1986), pp. 134–141. ↑ 96

[21] Е. А. Барковский, А. В. Соколов. «Вероятностная модель для задачи оптимального управления work-stealing деками при различных стратегиях перехвата работы», IX Международная Петрозаводская конференция «Вероятностные методы в дискретной математике» (Петрозаводск, Россия, 30 мая–3 июня 2016 г.), с. 11–13. ↑ 96

[22] A. Sokolov, E. Barkovsky. "The Mathematical Model and The Problem of Optimal Partitioning of Shared Memory for Work-Stealing Deques", *Parallel Computing Technologies*, 13th International Conference PaCT 2015 (Petrozavodsk, Russia, August 31–September 4, 2015), Lecture Notes in Computer Science, vol. **9251**, Springer International Publishing, 2015, pp. 102–106. ↑ 85,97

Пример ссылки на эту публикацию:

Е. А. Барковский, Р. И. Кучумов, А. В. Соколов. «Оптимальное управление двумя work-stealing деками в общей памяти при различных стратегиях перехвата работы», *Программные системы: теория и приложения*, 2017, **8**:1(32), с. 83–103.

URL: http://psta.psiras.ru/read/psta2017_1_83-103.pdf

Об авторах:

Евгений Александрович Барковский

В 2015 г. закончил аспирантуру Института прикладных математических исследований Карельского научного центра РАН (лаборатория информационных компьютерных технологий). Автор 14 научных публикаций, посвященных параллельным динамическим структурам данных. Область научных интересов: задачи оптимального управления параллельными динамическими структурами данных, work-stealing балансировщики.

e-mail: barkevgen@gmail.com

Руслан Ильдусович Кучумов

Магистрант СПбГУ факультета прикладной математики процессов управления. Автор публикаций про параллельные алгоритмы хеширования и про work-stealing балансировщики задач. Область научных интересов: прикладная математика и информатика, параллельные и распределенные алгоритмы, балансировщики задач, балансировка нагрузки.

e-mail: kuchumovri@gmail.com

Андрей Владимирович Соколов

Профессор, ведущий научный сотрудник Института прикладных математических исследований Карельского научного центра РАН. Является автором более 100 научных публикаций. Область научных интересов: оптимальное управление динамическими структурами данных, оптимальное динамическое распределение нестраничной памяти, управляемые случайные блуждания, цепи Маркова, параллельные вычисления, динамические work-stealing балансировщики.

e-mail: sokavs@gmail.com

Eugene Barkovksy, Ruslan Kuchumov, Andrew Sokolov. *Optimal control of two deques in shared memory with various work-stealing strategies.*

Abstract. In parallel load balancers based on work-stealing strategy each processor has its own task deque. One end of the deque is used by the owner to add and retrieve tasks, and the second — by the other processors to steal tasks.

The aim of this research is to construct and analyze mathematical models of the process of work with two cyclic deques located in the shared memory. The parameters of these models are the probabilities of operations (serial or parallel) at each step of discrete time. Mathematical models are built as random walks on an integer lattice in the plane. On the basis of models, problems of optimal partition of memory were solved for various work-stealing strategies. As the criterion of optimality we consider the maximum mean time to the memory overflow.

Statistical studies to assess the probabilities of operations were carried out for multiple types of tasks. For this purpose, as a part of our RFBR grant, work-stealing balancer was constructed. Obtained probabilities were used in numerical experiments to analyze the developed models.

To solve these problems apparatus of controlled random walks, absorbing Markov chains and LAPACK system were used. Calculations were made using cluster KRC RAS. (*In Russian*).

Key words and phrases: work-stealing schedulers, work-stealing deques, data structures, Markov chains, random walks.

References

[1] R. D. Blumofe, C. E. Leiserson. “Scheduling Multithreaded Computations by Work Stealing”, *Journal of the ACM*, **46**:5 (1999), pp. 720–748.

[2] M. Herlihy, N. Shavit. *The Art of Multiprocessor Programming*, Elsevier, 2008, 536 p.

[3] D. Knuth. *The Art of Computer Programming*. V. 1, Addison-Wesley, 2005, v+134 p.

[4] D. Hendler, N. Shavit. “Non-blocking Steal-half Work Queues”, ACM Symposium on Principles of Distributed Computing (PODC 2002) (July 21–24, 2002, Monterey, California), pp. 280–289.

[5] A. V. Sokolov, A. V. Drac. “The Linked List Representation of n LIFO-Stacks and/or FIFO-Queues in the Single-Level Memory”, *Information Processing Letters*, **13**:19–21 (2013), pp. 832–835.

[6] Ye. A. Aksenova, A. A. Lazutina, A. V. Sokolov. “Ob optimal’nykh metodakh predstavleniya dinamicheskikh struktur dannykh”, *Obozreniye prikladnoy i promyshlennoy matematiki*, **10**:2 (2003), pp. 375–376 (in Russian).

[7] D. Hendler, Y. Lev, M. Moir, N. Shavit. “A Dynamic-Sized Nonblocking Work Stealing Deque”, *Distributed Computing*, **18**:3 (2006), pp. 189–207.

[8] M. Mitzenmacher. “Analyses of Load Stealing Models Based on Differential Equations”, ACM Symposium on Parallel Algorithms and Architectures (SPAA’ 98) (Puerto Vallarta, Mexico, June 28–July 2, 1998), pp. 212–221.

This work was supported by grant RFBR No 15-01-03404-a.

[9] D. Chase, Y. Lev. “Dynamic Circular Work-Stealing Deque”, ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2005) (Las Vegas, Nevada, USA, July 18–20, 2005), pp. 21–28.

[10] E. A. Aksenova, A. A. Lazutina, A. V. Sokolov. “Study of a Non-Markovian Stack Management Model in a Two-Level Memory”, *Programming and Computer Software*, **30**:1 (2004), pp. 25–33.

[11] E. A. Aksenova, A. V. Sokolov. “The Optimal Implementation of Two FIFO-queues in Single-Level Memory”, *Applied Mathematics*, **2**:10 (2011), pp. 1297–1302.

[12] A. V. Sokolov, A. V. Drats. “Simulation of some methods of representation of n FIFO-queues in the single-level memory”, *Evristicheskiye algoritmy i raspredelennyye vychisleniya*, **1**:1 (2014), pp. 40–52 (in Russian).

[13] A. V. Kalachev. *Multi-core processors*, Binom, M., 2010 (in Russian).

[14] J. G. Kemeny, J. L. Snell. *Finite Markov Chains*, Van Nostrand, Princeton, NJ, 1960, 210 p.

[15] N. S. Arora, R. D. Blumofe, C. G. Plaxton. “Thread Scheduling for Multiprogrammed Multiprocessors”, ACM Symposium on Parallel Algorithms and Architectures (SPAA’ 98) (Puerto Vallarta, Mexico, June 28–July 2, 1998), pp. 119–129.

[16] N. M. Le, A. Pop, A. Cohen, F. Z. Nardelli. “Correct and efficient work-stealing for weak memory model”, ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP’ 13) (Shenzhen, China, February 23–27, 2013), pp. 69–80.

[17] R. I. Kuchumov. “Implementation and Analysis of Work-stealing Task Scheduler”, *Stokhasticheskaya optimizatsiya v informatike*, **12**:1 (2016), pp. 20–39 (in Russian).

[18] G. Paoloni. *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures*, White Paper 324264-001, Intel, 2010, URL: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf

[19] M. A. Posypkin, I. Kh. Sigal. “Parallel Combined Algorithm for Solving Knapack Problems”, IV Mezhdunarodnaya konferentsiya “Parallel’nyye vychisleniya i zadachi upravleniya” (Moskva, Rossiya, 27–29 oktyabrya 2008 g.), pp. 177–189 (in Russian).

[20] D. Y. Yeh, T. Munakata. “Dynamic initial allocation and local reallocation procedures for multiple stacks”, *Communications of the ACM*, **29**:2 (1986), pp. 134–141.

[21] Ye. A. Barkovskiy, A. V. Sokolov. “Veroyatnostnaya model’ dlya zadachi optimal’nogo upravleniya work-stealing dekami pri razlichnykh strategiyakh perekhvata raboty”, IX Mezhdunarodnaya Petrozavodskaya konferentsiya “Veroyatnostnyye metody v diskretnoy matematike” (Petrozavodsk, Rossiya, 30 maya–3 iyunya 2016 g.), pp. 11–13 (in Russian).

[22] A. Sokolov, E. Barkovsky. “The Mathematical Model and The Problem of Optimal Partitioning of Shared Memory for Work-Stealing Deques”, *Parallel Computing Technologies*, 13th International Conference PaCT 2015 (Petrozavodsk, Russia, August 31–September 4, 2015), Lecture Notes in Computer Science, vol. **9251**, Springer International Publishing, 2015, pp. 102–106.

Sample citation of this publication:

Eugene Barkovksy, Ruslan Kuchumov, Andrew Sokolov. "Optimal control of two deques in shared memory with various work-stealing strategies", *Program systems: Theory and applications*, 2017, **8**:1(32), pp. 83–103. (*In Russian*).

URL: http://psta.psiras.ru/read/psta2017_1_83-103.pdf