УДК 519.7

ВЫЧИСЛЕНИЕ ДЕТЕРМИНАНТА И ПРОИЗВЕДЕНИЯ МАТРИЦ В СТРУКТУРЕ КЛЕТОЧНОГО АВТОМАТА

В. С. Кожевников*, И. В. Матюшкин**

**Московский физико-технический институт (Национальный исследовательский университет), г. Долгопрудный, Россия*
***Национальный исследовательский университет «Московский институт электронной техники», г. Москва, Россия*

Приведено формальное описание двумерных клеточных автоматов, реализующих умножение матриц и нахождение определителя. Все алгоритмы даны для границ с замыканием, шаблона окрестности Неймана и закрытых вычислений (т. е. данные не вводятся в процессе расчёта). Отдельно реализовано умножение матрицы на вектор-столбец. Для вычисления определителя предложены два алгоритма, основанные на методе Гаусса. Один имеет линейную сложность и использует управляющий флаг с пятью состояниями, однако применим не ко всякой матрице. Другой работает для любой матрицы, но имеет квадратичную сложность и его управляющий флаг принимает одиннадцать состояний.

Ключевые слова: *клеточный автомат, определитель, детерминант, умножение матриц, параллельные вычисления.*

DOI 10.17223/20710410/46/8

COMPUTATION OF A DETERMINANT AND A MATRIX PRODUCT IN CELLULAR AUTOMATA

V. S. Kozhevnikov*, I. V. Matyushkin**

**Moscow Institute of Physics and Technology, Dolgoprudny, Russia*
***National Research University of Electronic Technology, Moscow, Russia*

E-mail: vladislavkozhevnikov@gmail.com, imatyushkin@niime.ru

In the paper, possible implementations in cellular automata of matrix–vector multiplication, matrix multiplication, and determinant computation are described. The algorithms are formalised in terms of a cellular automaton model using a two-dimensional field with closed boundaries and von Neumann neighbourhood. The proposed algorithms are notable for isolated calculations (meaning that no data is entered during the computation process), which is a feature of a classical cellular automaton as opposed to a systolic array. Matrix multiplication is implemented for two square matrices as well as specifically for a matrix and a column vector, the latter implementation using less control flag states and therefore less additional memory. The matrix multiplication algorithm adapts Katona's scheme for matrix multiplication in a systolic array to an isolated cellular automaton. For calculation of a determinant, two algorithms based on Gaussian elimination are proposed. One has linear complexity and uses a control flag with only 5 states, being, however, inapplicable to an arbitrary matrix. The other one is modified to seek the largest leading element during row reduction, which makes its complexity quadratic and its control flags assume 11 states but allows the algorithm to be applied to an arbitrary matrix and be more numerically stable.

Keywords: *cellular automaton, determinant, matrix multiplication, parallel computing.*

Введение

Несмотря на то, что в настоящее время фокус исследований по массивному вычислительному параллелизму и нетрадиционным парадигмам компьютинга сместился с однородных вычислительных систем (ОВС), популярных в 60–80-х гг. XX века [1, 2], на нейронные сети и нейроморфные системы, исследование клеточно-автоматного (КА) параллелизма, взятого в классической парадигме вычислений, предполагающей явно заданный алгоритм решения задачи, по-прежнему представляет теоретический интерес. Данная работа продолжает цикл публикаций по клеточно-автоматным алгоритмам типовых матричных операций. Ранее были даны точные КА-формулировки, решающие очень простые задачи: унарной поэлементной операции [3], сортировки строк [4], отражения матрицы относительно центральной вертикали или горизонтали, а также главной диагонали, т. е. транспонирования [3]. В поисках более простого КА для задачи транспонирования мы нашли алгоритм, проходящий некоторые псевдослучайные пермутации матрицы [5], в том числе поворот на 90 и 180° (отражение от центра).

В традиционном курсе линейной алгебры перечисленные преобразования, не затрагивающие элементы матрицы, считаются самоочевидными, но для ОВС (или клеточно-автоматного вычислительного узла) они достаточно важны. Ещё большей важностью обладают операции, меняющие элементы матрицы, и прежде всего — произведение матриц, вычисление определителя и обратной матрицы. К ним условно причислим нахождение обычного произведения натуральных чисел по схеме Атрубина [4], где каждый сомножитель задается вектором его цифр. Решение первых двух задач средствами КА является целью настоящей работы. Способы параллельной реализации этих операций в многопроцессорной системе достаточно известны [6]. В гораздо меньшей степени исследовалось КА-распараллеливание, располагающее более скудными средствами (прежде всего, это запрет на «дальнодействие», т. е. удалённые ячейки памяти не могут обмениваться данными даже через центральный процессорный элемент). Впервые мы дадим точную КА-формулировку для вычисления определителя матрицы.

1. Обозначения и предварительные замечания

В работе используются следующие обозначения для элементов двумерного клеточного автомата, имеющего структуру квадратной решётки размера $n \times n$ на торе, т. е. с замкнутыми границами. Состояние клетки определяется конечным набором компонент $\left\langle a^{(1)}, \ldots, a^{(K)} \right\rangle$, каждая из которых принимает значения в своём множестве состояний $D^{(k)}, k \in \{1, \ldots, K\}$. Удобно неформально разбивать по семантике компоненты на три группы: данные, флаги управления, стек. Глобальную конфигурацию КА, взятую по отдельной компоненте, назовём *слоем*. Клетки автомата индексируются как матричные элементы, первый индекс является номером строки, а второй — столбца. Компоненты клетки снабжаются теми же индексами, что и данная клетка. То есть $a_{ij}^{(k)}$ — компонента $a^{(k)}$ клетки, расположенной в i-й строке и j-м столбце. Используется относительная индексация для обозначения соседних клеток. Если выбрана центральная клетка окрестности, то индексы у её компонент опускаются, пишется просто $a^{(k)}$. Компоненты соседей обозначаются стрелками, например, $a_{\rightarrow}^{(k)}$ отвечает ближайшей клетке справа, $a_{\uparrow}^{(k)}$ — клетке сверху по отношению к центральной. Локальная функ-

ция перехода (ЛФП) задаётся таблицей условий и соответствующих переходов. Мы не требуем, чтобы условия были взаимоисключающими, поэтому в процессе применения ЛФП последовательно проверяются все условия. Эти условия, перечисленные в соответствующей таблице, не обязательно исчерпывают все возможные конфигурации окрестности. Если окрестность клетки имеет конфигурацию, отсутствующую в таблице, то её состояние не меняется. Иногда такие (тождественные) переходы всё же включаются в таблицу и обозначаются словом idem. В приведённых алгоритмах ЛФП для граничных ячеек не требует отдельной спецификации.

Обычно в практике моделирования КА-расчёт прерывают либо внешним образом по достижении некого числа итераций или по какому-то условию, типичному для численных методов, либо внутренним образом при равенстве текущей и прошлой глобальных конфигураций. В теории алгоритмов для останова обязательно нужны синтаксические средства (стоп-символ для машины Тьюринга, заключительная формула подстановки в нормальных алгорифмах Маркова и т. п.). При реализации КА-алгоритмов имеются две альтернативы: останов по idem и останов по стоп-значению. Первая альтернатива означает, что в каждой ячейке КА действует ЛФП типа idem (обычно контролируемая стоп-значением «заморозки» одного из флагов); она довольно естественна, но неудобна при техническом исполнении ОВС. Вторая альтернатива предполагает, что одна, любая или выделенная, ячейка КА приходит в стоп-состояние. Она предполагает другое написание ЛФП, чем в первом случае, и удобна с технической точки зрения. В основном мы будем использовать останов по idem.

Для нас принципиально разделять вычисления на закрытые и открытые. Вычисление назовём закрытым, если всё необходимое для него имеется в вычислителе до начала вычисления, а его результаты выводятся вовне после завершения. Иными словами, КА как математический объект не имеет средств ввода/вывода. Обычно классические КА в теории предстают закрытыми объектами, хотя открытым КА посвящена отдельная глава в классической работе [7]. Будем считать, что КА сконфигурирован до расчёта, т. е. вычисления закрытые. С точки зрения ОВС более удобны открытые, что характерно для схем Атрубина и Катоны. Здесь данные вводятся в процессе исполнения алгоритма. Несмотря на то, что мы в значительной мере используем идеи подобных алгоритмов, новизна работы состоит в их адаптации к закрытым вычислениям.

2. Умножение матриц

2.1. Традиционный параллелизм матричного умножения

В отличие от последовательных алгоритмов умножения матриц, которые используют всевозможные аналитические приёмы от несложных матричных преобразований [8] до инструментария тензорного анализа и теории групп [9] для сокращения (хотя бы асимптотического) числа необходимых арифметических операций, существующие параллельные алгоритмы работают практически по определению матричного произведения. Наиболее простая классификация условно разделяет их на два типа: ленточный и блочный.

В основе *ленточного* метода лежит разбиение матриц (или одной матрицы) на ленты. Под *лентой* понимается строка (столбец) или несколько смежных строк (столбцов). Пусть требуется найти произведение $AB = C$ квадратных матриц порядка n. Естественным представляется разбить матрицу A на строки, а B — на столбцы. Тогда n^2 процессоров могут вычислить элементы произведения C, параллельно перемножая соответствующие строки и столбцы. Если процессоров меньше, чем элементов C, то A

и B разбиваются на ленты, содержащие по несколько строк и столбцов, а процессоры находят уже не отдельные элементы C, а подматрицы.

Блочный метод, являющийся, пожалуй, наиболее распространённым, можно считать обобщением ленточного. Матрицы разбиваются на *блоки*, то есть подматрицы. Например, имея дело с квадратными матрицами A и B порядка n, кратного некоторому натуральному числу q (в частности, $q = n$), их можно разбить на q^2 равных по объёму блоков:

$$AB = \begin{pmatrix} A_{11} & \dots & A_{1q} \\ \vdots & \ddots & \vdots \\ A_{q1} & \dots & A_{qq} \end{pmatrix} \begin{pmatrix} B_{11} & \dots & B_{1q} \\ \vdots & \ddots & \vdots \\ B_{q1} & \dots & B_{qq} \end{pmatrix} = \begin{pmatrix} C_{11} & \dots & C_{1q} \\ \vdots & \ddots & \vdots \\ C_{q1} & \dots & C_{qq} \end{pmatrix} = C.$$

Тогда полученные блочные матрицы перемножаются аналогично числовым:

$$C_{ij} = \sum_{k=1}^{q} A_{ik} B_{kj}.$$

Здесь C_{ij} — блоки произведения C, каждый из которых может быть вычислен отдельным процессором. В пределах одного процессора могут использоваться любые методы последовательного умножения матриц для более эффективного нахождения произведения блоков (например, алгоритм Штрассена [10]).

Из конкретных алгоритмов, реализующих блочный метод, широко известны схемы Фокса и Кэннона [6]. Они подразумевают использование сети процессоров с распределённой памятью, что близко к идее КА. Существуют алгоритмы, реализующие матричное умножение на систолических массивах процессоров [10, 11], а также его частный случай — умножение матрицы на вектор [12], которые тоже следует отнести скорее к блочным, чем к ленточным.

Что касается времени вычисления произведения, то для двумерной решётки процессоров, сообщающихся лишь с соседями в своей окрестности Неймана, минимально возможное время вычисления не менее $\Theta(n)$, где n — порядок перемножаемых квадратных матриц, вне зависимости от числа процессоров [13]. Имеется достаточное количество примеров алгоритмов с такой сложностью, перемножающих две [10, 11, 14] и даже три матрицы [15]. Для сетей с более сложной топологией и количеством процессоров, превышающим $\Theta(n^2)$, предложены алгоритмы, производящие умножение за время $\Theta(\log_2 n)$ [14]. Алгоритмы с меньшим временем исполнения нам не известны. КА накладывает жёсткое ограничение на структуру сети, предполагая наличие решётки.

2.2. Алгоритм 1. Умножение матрицы на столбец

Сначала рассмотрим задачу умножения матрицы на столбец. Выбранный способ в некотором смысле является ленточным, что отличает его от применяемого нами далее. Состояние ячейки КА является трёхкомпонентным (табл. 1), R — некоторое кольцо, например $\mathbb{R}$ или $\mathbb{C}$.

Таблица 1

Компоненты автомата, умножающего матрицу на столбец

Компонента	Множество значений	Назначение
a	Кольцо R	Хранение матрицы
b	Кольцо R	Хранение столбца
s	$\{0, 1, 2\}$	Управление

Запишем начальные условия:

1) $s_{1j} = 1$, $j \in \{1, \ldots, n-1\}$; остальные s_{ij} равны 0;
2) $b_{1j} = B_j$, $j \in \{1, \ldots, n\}$; остальные b_{ij} произвольные;
3) $\|a_{ij}\| = A$.

Здесь $B_{n\times 1}$ — столбец, на который умножается матрица $A_{m\times n}$, то есть матрица записывается в автомат естественным образом, а столбец — транспонированным в первую строку. На каждом шаге столбец B копируется вниз, пока не будет записан в каждой строке. Как только столбец достигает строки, в ней начинается процесс умножения и суммирования. Сумма вычисляется последовательно, начиная с конца строки, и в итоге записывается в её начало. Таким образом, ответ будет располагаться в первом столбце компоненты a. В табл. 2 представлена ЛФП, реализующая описанный алгоритм; условия, разделённые вертикальной чертой, задают конъюнкцию ($\wedge$), а горизонтальной — дизъюнкцию ($\vee$).

Таблица 2

ЛФП автомата, умножающего матрицу на столбец

№ п/п	Условие		Правило перехода
1	$s = 0$	$s_{\leftarrow} = 0$	$s := s_{\uparrow}$
		$s_{\leftarrow} = 1$	$s := 2$
	$(s = 1) \wedge (s_{\rightarrow} \neq 1)$		
2	$s = 0$		$b := b_{\uparrow}$
	$s = 1$	$s_{\rightarrow} = 0$	$a := a \cdot b + a_{\rightarrow} \cdot b_{\rightarrow}$
		$s_{\rightarrow} = 2$	$a := a \cdot b + a_{\rightarrow}$
3	$s = 2$		idem

В табл. 3, где иллюстрируется работа КА на примере умножения матрицы размера 4×5 на соответствующий столбец, t — номер шага, $t = 0$ — начальная конфигурация в алгоритме 1; в столбце, соответствующем флагу s, цветом выделены «замороженные» клетки; в столбце компоненты a различными цветами выделены клетки, содержащие частичную сумму (например, 12), полную сумму (например, 15) и окончательный результат (например, 19); в столбце компоненты b цветом выделены клетки, содержащие элементы столбца B, а клетки с незначащим содержимым заштрихованы.

Строки, состоящие из нулей, ещё не содержат столбца B. Как только в флаге s строки появляются единицы, в ней начинается умножение и суммирование. Значение $s = 2$ является остановочным, то есть при переходе в состояние с $s = 2$ клетка «замораживается» (обозначено словом idem в табл. 2). Автомат завершает работу на шаге $t = m+n-2$. Алгоритм работает лишь для $n \geqslant 2$, то есть столбец B должен содержать более одного элемента.

Таблица 3

Эволюция трёх компонент $\langle a, b, s\rangle$ КА при умножении матрицы размера 4×5

t	s					a					b					t	s					a					b				
0	1	1	1	1	0	5	1	3	1	0	4	2	3	0	2	4	2	2	2	2	2	31	11	9	0	0	4	2	3	0	0
	0	0	0	0	0	2	3	0	1	4	0	0	0	0	0		1	2	2	2	2	2	14	8	8	4	4	2	3	0	0
	0	0	0	0	0	-1	0	-1	2	-1	0	0	0	0	0		1	1	2	2	2	-1	0	-5	-2	-1	4	2	3	0	0
	0	0	0	0	0	0	3	1	-1	5	0	0	0	0	0		1	1	1	2	2	0	3	1	10	5	4	2	3	0	0
1	1	1	1	2	2	5	1	3	0	0	4	2	3	0	0	5	2	2	2	2	2	31	11	9	0	0	4	2	3	0	0
	1	1	1	1	0	2	3	0	1	4	4	2	3	0	2		2	2	2	2	2	22	14	8	8	4	4	2	3	0	0
	0	0	0	0	0	-1	0	-1	2	-1	0	0	0	0	0		1	2	2	2	2	-1	-5	-5	-2	-1	4	2	3	0	0
	0	0	0	0	0	0	3	1	-1	5	0	0	0	0	0		1	1	2	2	2	0	3	13	10	5	4	2	3	0	0
2	1	1	2	2	2	5	1	9	0	0	4	2	3	0	0	6	2	2	2	2	2	31	11	9	0	0	4	2	3	0	0
	1	1	1	2	2	2	3	0	8	4	4	2	3	0	0		2	2	2	2	2	22	14	8	8	4	4	2	3	0	0
	1	1	1	1	0	-1	0	-1	2	-1	4	2	3	0	2		2	2	2	2	2	-9	-5	-5	-2	-1	4	2	3	0	0
	0	0	0	0	0	0	3	1	-1	5	0	0	0	0	0		1	2	2	2	2	0	19	13	10	5	4	2	3	0	0
3	1	2	2	2	2	5	11	9	0	0	4	2	3	0	0	7	2	2	2	2	2	31	11	9	0	0	4	2	3	0	0
	1	1	2	2	2	2	3	8	8	4	4	2	3	0	0		2	2	2	2	2	22	14	8	8	4	4	2	3	0	0
	1	1	1	2	2	-1	0	-1	-2	-1	4	2	3	0	0		2	2	2	2	2	-9	-5	-5	-2	-1	4	2	3	0	0
	1	1	1	1	0	0	3	1	-1	5	4	2	3	0	2		2	2	2	2	2	19	19	13	10	5	4	2	3	0	0

2.3. Алгоритм 2. Умножение матриц в два этапа

Приведённый здесь алгоритм умножения квадратных матриц использует идею [11] (рис. 1). Элементы матриц A и B поступают в поле КА с дискретным запаздыванием (отображено сдвигом строк/столбцов) во время расчёта.

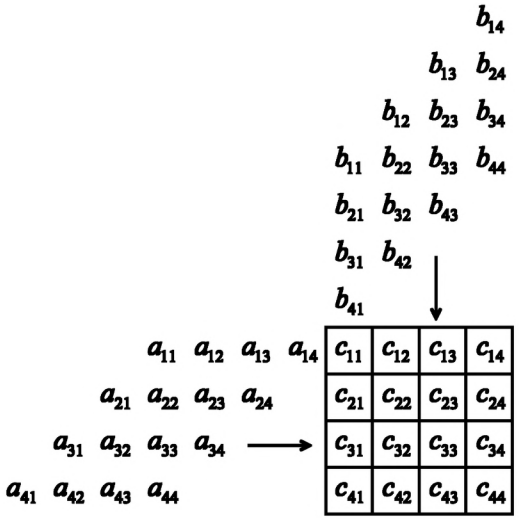


Рис. 1. Схема умножения Катоны

Двумерный клеточный автомат хранит элементы перемножаемых матриц A и B, а также их произведения C. Сначала происходит относительный сдвиг строк матрица A и столбцов матрицы B, а затем они (столбцы и строки) перемещаются так, что на каждом шаге в каждой клетке остаётся лишь перемножать содержащиеся в ней элементы A и B и складывать произведение с элементом C в той же клетке. Иными словами, производится поклеточное накопление суммы по формуле $c := c + a \cdot b$, перемещения же частичных сумм, как в алгоритме 1, не происходит, что является прин-

ципиальной особенностью алгоритма Катоны. В отличие от оригинального алгоритма Катоны, предлагаемый алгоритм ориентирован на закрытые вычисления. Структура данных КА показана в табл. 4.

Таблица 4

Компоненты автомата, перемножающего матрицы в два этапа

Компонента	Множество значений	Назначение
a	Кольцо R	Хранит первый множитель, матрицу A
b	Кольцо R	Хранит второй множитель, матрицу B
c	Кольцо R	Хранит результат произведения
s	$\{0,1\}$	Флаг синхронизации
v	$\{0,1\}$	Флаг вертикального перемещения
h	$\{0,1\}$	Флаг горизонтального перемещения

Этап 1. Смещение матричных элементов. Матрица A: строки с флагом $h = 1$ двигаются вправо. Флаг распространяется из первой строки вниз. Матрица B: столбцы с флагом $v = 1$ двигаются вниз. Флаг распространяется из первого столбца вправо. Соответствующая ЛФП представлена в табл. 5.

Этап 2. Перемножение. При указанных правилах перехода после смещения строки матрицы A и столбцы матрицы B будут продолжать двигаться, но уже синхронно, смещённые друг относительно друга. Если, начиная с шага $t = n$, добавить правило 3, приведённое в табл. 6, то автомат произведёт умножение за следующие n шагов (начиная с шага $t = n$).

Таблица 5

ЛФП на первом этапе

№ п/п	Условие	Правило перехода
1	$h_{\uparrow} = 1$	$h := 1$
	$h = 1$	$a := a_{\leftarrow}$
2	$v_{\leftarrow} = 1$	$v := 1$
	$v = 1$	$b := b_{\uparrow}$

Таблица 6

ЛФП для флага s

№ п/п	Условие	Правило перехода
3	$t \geqslant n$	$c := c + a \cdot b$
4	Всегда	$s := s_{\uparrow}$

Синхронизация. Автомат следует остановить через $2n - 1$ шагов с момента запуска. Останов на шаге $t = 2n - 1$ и изменение правил с шага $t = n$ происходит по значению. Для этой цели вводится флаг синхронизации $s \in \{0,1\}$, изначально равный 1 лишь в левой верхней клетке $(1,1)$ и двигающийся вниз. При достижении им левой нижней клетки $(n,1)$ добавляется правило 3 (табл. 6) для всех клеток одновременно, а при повторном её достижении автомат останавливается. Выпишем полностью начальные условия алгоритма 2:

$$\forall j \in \{1,\ldots,n\}\,(h_{1j} = 1) \ \wedge \ \forall i \neq 1 \forall j \in \{1,\ldots,n\}\,(h_{ij} = 0) \ \wedge \ \forall i \in \{1,\ldots,n\}\,(v_{i1} = 1),$$
$$\forall j \neq 1 \forall i \in \{1,\ldots,n\}\,(v_{ij} = 0) \ \wedge \ s_{11} = 1,$$
$$\forall (i,j) \neq (1,1)\,(s_{ij} = 0) \ \wedge \ \forall i,j \in \{1,\ldots,n\}\,(c_{ij} = 0).$$

2.4. Последовательное выполнение КА-алгоритмов и проблема синхронизации

На примере алгоритма 2 мы встретились с проблемой последовательного сочленения КА, когда логика решения задачи требует её декомпозиции на несколько этапов и соответственно на несколько простых КА со своими ЛФП. Для последовательных вычислений она почти незаметна, так как объединяются множества инструкций и данных. Имеет ли формализм КА внутренние средства реализации последовательного выполнения двух КА над общим полем данных?

Абстрактный КА с двумя компонентами можно представить четверкой $\Xi = \langle \mathrm{T}, S = A \times B, LTF \rangle$. Здесь T — топология поля и шаблона окрестности; S — множество состояний ячейки (для определённости состоящее из двух компонент); LTF — локальная функция перехода, т. е. $\begin{pmatrix} a^{t+1} \\ b^{t+1} \end{pmatrix} = \begin{pmatrix} f\left(a^t, b^t\right) \\ g\left(a^t, b^t\right) \end{pmatrix}$, где t — номер шага, $a \in A$, $b \in B$. Пусть есть два КА Ξ_1, Ξ_2 с общей топологией $\mathrm{T} \equiv \mathrm{T}_1 = \mathrm{T}_2$, перекрывающимися состояниями $S_1 = A \times B$, $S_2 = B \times C$ и различными функциями перехода $LTF_1 = \begin{pmatrix} f_1 \\ g_1 \end{pmatrix}, LTF_2 = \begin{pmatrix} g_2 \\ h_2 \end{pmatrix}$, которые должны функционировать последовательно, будучи связаны конечной и начальной конфигурациями для произвольной ячейки поля неким правилом переноса $\begin{pmatrix} b^{t_2=0} \\ c^{t_2=0} \end{pmatrix} = TR \begin{pmatrix} a^{t_1=t_F} \\ b^{t_1=t_F} \end{pmatrix}$ в момент времени t_F. Каждый автомат работает в своем времени — t_1 и t_2 соответственно. Требуется построить третий КА, эмулирующий их последовательную работу в едином времени t. Обозначим его как $\Xi = \Xi_1 \oplus \Xi_2$.

Естественно сразу взять $S = A \times B \times C$, а ЛФП представить в виде

$$LTF = \begin{cases} LTF_1, & t \leqslant t_F, \\ LTF_2, & t > t_F \end{cases}$$

или в покомпонентном виде

$$(a, b, c)^{t+1} = \begin{cases} \left(f_1\left(a^t, b^t\right), g_1\left(a^t, b^t\right), \mathrm{idem}\right), & t \leqslant t_F, \\ \left(\mathrm{idem}, g_2\left(b^t, c^t\right), f_1\left(b^t, c^t\right)\right), & t > t_F. \end{cases}$$

Однако такой вид недопустим, поскольку идеология КА запрещает при формализации ЛФП ссылаться на абсолютные величины — номер итерации, индексы ячейки в поле и т. п. Отдельная ячейка «не знает» о глобальных событиях наподобие ситуации останова. В алгоритме 2 нам пришлось ввести флаг синхронизации, чтобы реализовать условный останов по значению первого КА и воспользоваться им для останова второго КА. Даже в этой ситуации требуется внешнее вмешательство, чтобы синтезировать ЛФП с управлением по флагу, полагая

$$LTF = \begin{cases} LTF_1, & s = 0, \\ LTF_2, & s = 1. \end{cases}$$

Как синхронизировать ячейки КА по управляющему флагу, если он изменился только в одной ячейке? Проблема известна как задача синхронизации стрелков (Firing Squad Synchronization Problem, FSSP) [16]. Однако заметим, что при синтезе Ξ нам придётся идти на затраты памяти в виде дополнительных управлений U, и тогда $S = A \times B \times \times C \times U$, и на затраты времени на синхронизацию.

Если применить для алгоритма 2 результат, полученный в [17], как использующий наименьшее среди известных решений количество состояний, то флаг s должен будет принимать значения в множестве $\{\omega, \lambda, \alpha, \beta, \gamma\}$. Также потребуется новый флаг f, идентичный s, так что $U \leftrightarrow \begin{pmatrix} s \\ f \end{pmatrix}$. Синхронизация производится независимо в каждой строке. Правила перехода для флагов s и f (будем называть их соответственно s-правилами и f-правилами) заимствованы из [17, рис. 15] с той разницей, что отсутствует фиктивное состояние, используемое для задания правил перехода граничных клеток, так как мы используем КА с замкнутыми границами. Отсутствует также состояние выстрела. Синхронизация считается выполненной, если все клетки одновременно переходят в состояние ω, причём до этого паттерн $\omega\omega\omega$ не встречается.

Правила перехода для флагов s и f не отличаются, однако начальные состояния должны быть подобраны так, чтобы флаг s производил синхронизацию за $n-1$ шагов, а f — за $2n-1$ шагов. Итак, в новом варианте автомата в правила 1, 2 и 3 добавляется условие синхронизации, правило 4 заменяется s-правилами и добавляются f-правила (табл. 7).

Таблица 7

ЛФП синхронизированного КА

№ п/п	Условие	Правило перехода
1	$h_{\uparrow} = 1$	$h := 1$
	$\left(\overline{f_{\leftarrow} f f_{\rightarrow}} \neq \omega\omega\omega\right) \wedge (h = 1)$	$a := a_{\leftarrow}$
2	$v_{\leftarrow} = 1$	$v := 1$
	$\left(\overline{f_{\leftarrow} f f_{\rightarrow}} \neq \omega\omega\omega\right) \wedge (v = 1)$	$b := b_{\uparrow}$
3	$\left(\overline{f_{\leftarrow} f f_{\rightarrow}} \neq \omega\omega\omega\right) \wedge \left(\overline{s_{\leftarrow} s s_{\rightarrow}} = \omega\omega\omega\right)$	$c := c + a \cdot b$
4	s-правила	
5	f-правила	

Для флагов h, v и c начальные условия остаются прежними. Основная сложность заключается в задании начальных условий для флагов синхронизации s и f.

На примере алгоритма 2 мы видим, что накладные затраты на последовательное соединение КА могут быть сравнимы с основными затратами и даже превосходить их.

2.5. Алгоритм 3. Эффективный алгоритм умножения матриц

Оказывается, что можно обойтись вовсе без синхронизации. Для этого умножение происходит не одновременно во всех клетках и лишь на втором этапе, а в определённой последовательности, начиная с первого же шага. При этом используется всего один вспомогательный флаг $s \in \{0, r, d, 1\}$, имеющий четыре состояния. Границы двух этапов теперь несколько меняются и вообще становятся условными.

На первом этапе алгоритма, занимающем $n-2$ шага, процесс изменения флага s эквивалентен работе введённых ранее флагов h и v, если считать $s = 0 \Leftrightarrow h = v = 0$, $s = 1 \Leftrightarrow h = v = 1$, $s = r \Leftrightarrow (h = 0) \wedge (v = 1)$ и $s = d \Leftrightarrow (h = 1) \wedge (v = 0)$. На следующем этапе переходы усложняются. На первом этапе умножение производится лишь в клетках с $s = 1$, на втором — во всех клетках с $s \neq 0$. При $s = 0$ клетка находится в состоянии покоя, то есть она не производит никаких операций над матричными элементами. Подчеркнём, что всё это достигается применением одной и той же ЛФП (табл. 8) на обоих условных этапах.

Таблица 8

ЛФП алгоритма 3 умножения матриц

№ п/п	Условие			Правило перехода
1	$s=0$	$s_{\uparrow}=r$	$s_{\leftarrow}=0$	$s:=r$
			$s_{\leftarrow}=d$	$s:=1$
		$(s_{\uparrow}=0)\wedge(s_{\leftarrow}=d)$		$s:=d$
	$s=r$	$(s_{\leftarrow}=r)\wedge(s_{\uparrow}=1)$		$s:=1$
		$s_{\leftarrow}=1$	$(s_{\rightarrow}=r)\wedge(s_{\uparrow}\neq 1)$	
			$(s_{\rightarrow}=1)\vee(s_{\uparrow}=1)$	$s:=d$
		$s_{\leftarrow}=0$		$s:=0$
	$s=d$	$(s_{\uparrow}=d)\wedge(s_{\leftarrow}=1)$		$s:=1$
		$s_{\uparrow}=1$	$(s_{\downarrow}=d)\wedge(s_{\leftarrow}\neq 1)$	
			$(s_{\downarrow}=1)\vee(s_{\leftarrow}=1)$	$s:=r$
		$s_{\uparrow}=0$		$s:=0$
	$s=1$	$(s_{\uparrow}=r)\wedge(s_{\leftarrow}=d)$		$s:=0$
		$(s_{\uparrow}=0)\vee(s_{\leftarrow}=0)$		
		$(s_{\uparrow}=1)\wedge(s_{\leftarrow}=d)$		$s:=d$
		$(s_{\uparrow}=r)\wedge(s_{\leftarrow}=1)$		$s:=r$
2	$(s=r)\vee(s=1)$			$a:=a_{\leftarrow}$
	$(s=d)\wedge(s_{\leftarrow}=1)$			
3	$(s=d)\vee(s=1)$			$b:=b_{\uparrow}$
	$(s=r)\wedge(s_{\uparrow}=1)$			
4	$s=1$			$c:=c+a\cdot b$
	$(s=r)\wedge(s_{\uparrow}=1)$			
	$(s=d)\wedge(s_{\leftarrow}=1)$			
5	$s=s_{\rightarrow}=s_{\downarrow}=s_{\leftarrow}=s_{\uparrow}=0$			idem

Запишем начальные условия:

$$s_{11}=1\ \wedge\ \forall j\in\{2,\ldots,n\}\,(s_{1j}=r)\ \wedge\ \forall i\in\{2,\ldots,n\}\,(s_{i1}=d);$$

остальные $s_{ij}=0$;

$\|a_{ij}\|=A,\ \|b_{ij}\|=B$ — первый и второй сомножители;

c_{ij} произвольные.

В табл. 9, где представлена динамика КА для $n=5$, t — номер шага, $t=0$ — начальная конфигурация, в столбце флага s цветом выделены «замороженные» клетки (например, 0), а в столбце компоненты c — окончательный результат (например, 5).

В алгоритме 3 останов происходит по idem, которому соответствует нулевая конфигурация окрестности (фон Неймана), то есть $s=0$ в самой клетке и у всех её соседей. Автомат завершает работу за $(2n-1)$ шагов.

Таблица 9

Эволюция компонент $\langle a, b, c, s\rangle$ КА, работающего по алгоритму 3 при умножении матриц порядка $n = 5$

Этап 1					Этап 2				
t	s	a	b	c	t	s	a	b	c
0	1 r r r r	3 1 1 0 -1	1 -2 -1 2 -2	0 0 0 0 0	4	1 1 1 1 d	1 1 0 -1 3	2 -1 -1 -1 -2	3 -5 -6 6 0
	d 0 0 0 0	-2 0 -1 -1 2	2 -2 0 -2 -1	0 0 0 0 0		1 1 1 1 d	-1 -1 2 -2 0	-1 4 3 2 -1	-7 2 2 0 0
	d 0 0 0 0	3 1 3 1 0	-1 -1 2 1 1	0 0 0 0 0		1 1 1 1 d	1 0 3 1 3	3 -1 -1 -2 1	3 -8 6 3 0
	d 0 0 0 0	0 2 4 3 0	3 4 -1 -1 -1	0 0 0 0 0		1 1 1 1 d	0 0 2 4 3	-1 -2 0 1 -1	0 -4 8 -3 0
	d 0 0 0 0	-1 1 -1 -2 1	-1 -1 3 -1 1	0 0 0 0 0		r r r r 1	-1 1 -1 -2 1	1 -2 2 -1 1	0 0 0 0 0
1	1 1 r r r	-1 3 1 1 0	-1 -2 -1 2 -2	3 0 0 0 0	5	0 r r r r	3 1 1 0 -1	1 -2 2 -1 1	5 -6 -6 7 -6
	1 1 r r r	-2 0 -1 -1 2	1 -2 0 -2 -1	0 0 0 0 0		d 1 1 1 1	0 -1 -1 2 -2	2 -1 -1 -1 -2	-6 -2 8 -4 0
	d d 0 0 0	3 1 3 1 0	2 -1 2 1 1	0 0 0 0 0		d 1 1 1 1	3 1 0 3 1	-1 4 3 2 -1	6 -8 3 1 3
	d d 0 0 0	0 2 4 3 0	-1 4 -1 -1 -1	0 0 0 0 0		d 1 1 1 1	3 0 0 2 4	3 -1 -1 -2 1	0 -4 8 1 -3
	d d 0 0 0	-1 1 -1 -2 1	3 -1 3 -1 1	0 0 0 0 0		d 1 1 1 1	1 -1 1 -1 -2	-1 -2 0 1 -1	-1 -2 -2 2 1
2	1 1 1 r r	0 -1 3 1 1	3 -1 -1 2 -2	4 -6 0 0 0	6	0 0 1 1 1	3 3 1 1 0	1 -2 0 1 -1	5 -8 -4 7 -7
	1 1 1 r r	2 -2 0 -1 -1	-1 -2 0 -2 -1	-2 0 0 0 0		0 0 r r r	-2 0 -1 -1 2	1 -2 2 -1 1	-6 -1 9 -6 4
	1 1 1 r r	3 1 3 1 0	1 -2 2 1 1	0 0 0 0 0		1 d 1 1 1	1 3 1 0 3	2 -1 -1 -1 -2	3 -4 3 7 2
	d d d 0 0	0 2 4 3 0	2 -1 -1 -1 -1	0 0 0 0 0		1 d 1 1 1	4 3 0 0 2	-1 4 3 2 -1	9 -4 8 -3 1
	d d d 0 0	-1 1 -1 -2 1	-1 4 3 -1 1	0 0 0 0 0		1 d 1 1 1	-2 1 -1 1 -1	3 -1 -1 -2 1	-2 0 -2 1 3
3	1 1 1 1 r	1 0 -1 3 1	-1 4 3 2 -2	4 -5 -3 0 0	7	0 0 0 1 1	3 3 3 1 1	1 -2 -1 -2 1	5 -8 -4 8 -7
	1 1 1 1 r	-1 2 -2 0 -1	3 -1 -1 -2 -1	-4 4 0 0 0		0 0 0 1 1	-2 0 0 -1 -1	1 -2 0 1 -1	-6 -1 7 -5 6
	1 1 1 1 r	0 3 1 3 1	-1 -2 0 1 1	3 -2 6 0 0		0 0 0 r r	3 1 3 1 0	1 -2 2 -1 1	5 -7 2 7 -4
	1 1 1 1 r	0 2 4 3 0	1 -2 2 -1 -1	0 0 0 0 0		1 1 d 1 1	2 4 3 0 0	2 -1 -1 -1 -2	5 8 8 -3 -1
	d d d d 0	-1 1 -1 -2 1	2 -1 -1 -1 1	0 0 0 0 0		1 1 d 1 1	-1 -2 1 -1 1	-1 4 3 2 -1	-8 -1 -1 -1 2
					8	0 0 0 0 1	3 3 3 3 1	1 -2 -1 2 -1	5 -8 -4 6 -6
						0 0 0 0 1	-2 0 0 0 -1	1 -2 0 -2 1	-6 -1 7 -6 7
						0 0 0 0 1	3 1 3 3 1	1 -2 2 1 -1	5 -7 2 6 -4
						0 0 0 0 r	0 2 4 3 0	1 -2 2 -1 1	9 4 5 -3 -1
						1 1 1 d 1	1 -1 -2 1 -1	2 -1 -1 -1 -2	-7 -9 2 -3 1
					9	0 0 0 0 0	3 3 3 3 3	1 -2 -1 2 -2	5 -8 -4 6 -7
						0 0 0 0 0	-2 0 0 0 0	1 -2 0 -2 -1	-6 -1 7 -6 6
						0 0 0 0 0	3 1 3 3 3	1 -2 2 1 1	5 -7 2 6 -5
						0 0 0 0 0	0 2 4 3 3	1 -2 2 -1 -1	9 4 5 -3 -1
						0 0 0 0 0	-1 1 -1 -2 1	1 -2 2 -1 1	-5 -8 4 -4 3

3. Вычисление определителя матрицы

Как известно, определителем квадратной матрицы $\|a_{ij}\|_{i,j=1}^{n}$ называется сумма следующего вида:

$$\det\|a_{ij}\| = \sum_{i_1,\ldots,i_n=1}^{n} \varepsilon_{i_1,\ldots,i_n} a_{1i_1} \ldots a_{ni_n},$$

где $\varepsilon_{i_1,\ldots,i_n}$ — символ Леви-Чивиты.

Приведённая формула известна как формула полного разложения [18], которая содержит $n!$ однородных произведений. Существуют и другие равносильные определения, например основанное на разложении Лапласа:

$$\det\|a_{ij}\| = \sum_{j=1}^{n} a_{ij} A_{ij},$$

где A_{ij} — алгебраическое дополнение элемента a_{ij}, i — произвольный номер от 1 до n. Эта формула имеет рекурсивный характер, отсылая к определителям меньшего порядка и также предполагая чередование знаков. Наличие рекурсии ставит под сомнение возможность КА-реализации.

Перечислим некоторые элементарные свойства определителя, которые потребуются в дальнейшем:

— при перестановке двух строк (столбцов) матрицы её определитель меняет знак;
— при прибавлении к строке (столбцу) матрицы линейной комбинации остальных строк (столбцов) её определитель не меняется.

3.1. Вычислительный параллелизм для нахождения детерминанта

Наиболее простым является вычисление по определению, например по обобщённому правилу Саррюса, основанному на формуле полного разложения [19]. Разумеется, такой подход далёк от оптимального, поскольку имеет факториальную вычислительную сложность.

Также известен своей простотой метод Гаусса [18], использующий элементарные преобразования матрицы для приведения её к треугольному виду. Сложность его выполнения на одиночном процессоре составляет $\Theta(n^3)$ (n — порядок матрицы) [20]. Однако помимо того, что он плохо поддаётся распараллеливанию, он включает в себя деление, что делает его неприменимым над произвольными кольцами.

Алгоритмы, ориентированные на параллельные вычисления, способны находить детерминант за полилогарифмическое время [20]. При нижней оценке вычислительной сложности $\Theta(\log_2 n)$ существуют алгоритмы со сложностью $\Theta\left((\log_2 n)^2\right)$ [21]. При этом многие алгоритмы не используют деления, что значительно расширяет круг их применения. Так, алгоритм, описанный в [22], применим к матрицам над произвольным коммутативным кольцом с единицей. Тем не менее подобные алгоритмы требуют полиномиального количества процессоров $p = p(n)$. В различных работах приводятся следующие оценки: $p = \mathrm{O}(n^6)$ [20], $p = \mathrm{O}(n^4)$ [21], $p = \mathrm{O}(n^{3{,}496})$ [22]. Ясно, что использование такого количества вычислительных элементов непрактично. Кроме того, такие алгоритмы представляют процессоры нелокально связанными, что противоречит классическому определению КА.

Таким образом, существующие в теории быстрые алгоритмы не замещают более медленные аналоги с полиномиальной сложностью. В частности, упомянутый метод Гаусса широко применяется в современных вычислительных системах [23 – 26].

3.2. Алгоритм 4. Простой алгоритм вычисления определителя

Описанный ниже двумерный клеточный автомат (табл. 10) вычисляет определитель матрицы за линейное время (относительно порядка матрицы $n > 1$) методом Гаусса. Предполагается, что в процессе не возникает строк с нулевым ведущим элементом. Здесь F — некоторое поле, например $\mathbb{R}$ или $\mathbb{C}$.

Таблица 10

Компоненты клетки автомата, вычисляющего определитель методом Гаусса

Компонента	Множество значений	Назначение
s	$\{0, 1, 2, 3, 4\}$	Управляющий флаг
a	Поле F	Хранение матрицы
b	Поле F	Копирование элементов первой строки вниз
c	Поле F	Копирование элементов первого столбца вправо

Запишем начальные условия:

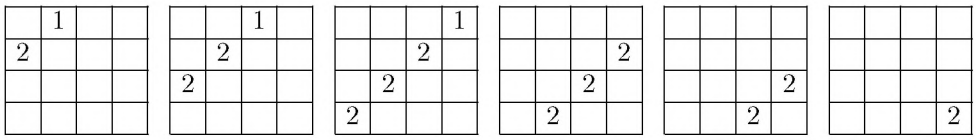
$$s_{11} = 1;\ \text{остальные } s_{ij} = 0,$$
$$\|a_{ij}\| = A \text{ — рассматриваемая матрица,}$$
b и c произвольные.

Флаг s определяет все переходы: $s = 0$ отвечает состоянию покоя (начальному состоянию большинства клеток), $s = 3$ — состоянию «заморозки», $s = 1$ обозначает клетку строки, вычитаемой из нижних строк (с множителем согласно методу Гаусса), $s = 2$ — клетку, переносящую элемент вычитаемой строки, делённый на ведущий элемент (хранится в компоненте b), и первый элемент данной строки (хранится в компоненте c), $s = 4$ — вспомогательное значение, позволяющее определить начало очередного этапа метода Гаусса.

Под *этапом метода Гаусса* понимается вычитание строки, умноженной на соответствующие коэффициенты, из нижележащих строк, в результате которого элементы, находящиеся под ведущим (первым ненулевым) элементом вычитаемой строки, обнуляются. Один этап осуществляется последовательностью конфигураций флага s, приведённой в табл. 11.

Таблица 11

Последовательность конфигураций флага s, соответствующая одному этапу (первому) метода Гаусса для $n = 4$



Однако не имеет смысла ожидать завершения одного этапа для начала следующего, поэтому алгоритм работает по принципу конвейера. Запуск каждого следующего этапа происходит на третий шаг после начала предыдущего. Полная последовательность конфигураций до остановки всех компонент автомата для $n = 5$ представлена в табл. 12. В ней цветом выделены клетки с флагами, принявшими окончательное значение (например, 0), и клетки, в которых компонента b содержит частичную сумму определителя (например, 1), а также итог вычислений (-8).

В табл. 13 приведены ЛФП алгоритма вычисления детерминанта. Условия накладывают ограничение только на флаг s. Они записаны в графическом виде. В каждом условии перечислены допустимые паттерны окрестностей. Для более сложных условий используются логические символы, знак дизъюнкции опускается. В пределах одного паттерна используются символы «|» в качестве «или» и «¬» в качестве «не». Как видно из табл. 13, эволюция флага s происходит независимо от компонент памяти a, b и c, так как их значения не фигурируют в условиях перехода. Останов происходит естественным образом, когда флаг s во всех клетках принимает значение 3, соответствующее тождественному переходу. Время работы алгоритма составляет $(3n - 4)$ шагов.

Недостатком приведённого алгоритма является то, что он неприменим для произвольной матрицы. Более того, его применимость можно выяснить лишь в процессе вычисления. Эта проблема устраняется выбором главного, то есть наибольшего по модулю элемента в столбце перед каждым этапом метода Гаусса. Состояние клетки в соответствующей модификации алгоритма имеет четыре компоненты $\langle s, a, b, c \rangle$,

аналогично простому алгоритму. Однако флаг синхронизации s принимает значения в множестве $\{0, \#, -1, -2, -3, -4, -5, 1, 2, 3, 4\}$. ЛФП представлена в табл. 14.

Запишем начальные условия:

$$s_{11} = -1;\ \text{остальные } s_{ij} = 0,$$
$$\|a_{ij}\| = A \text{ — рассматриваемая матрица,}$$
b и c произвольные.

Алгоритм с выбором главного элемента имеет уже не линейную, а квадратичную сложность, а именно n^2 шагов. Как и алгоритм 4, он имеет три компоненты памяти (одна для данных, две другие используются как буферные) и один флаг. Однако компонента флага имеет не пять состояний, а одиннадцать.

Таблица 12

Эволюция компонент в алгоритме 4 для матрицы порядка $n = 5$

t	s	a	b	c	t	s	a	b	c
0	1 0 0 0 0	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0	6	3 3 3 3 3	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0
	0 0 0 0 0	1 2 2 4 2	0 0 0 0 0	0 0 0 0 0		3 3 3 3 1	0 -1 -2 1 -1	1 3 4 3 3	1 1 1 1 1
	0 0 0 0 0	2 4 2 4 2	0 0 0 0 0	0 0 0 0 0		3 3 0 2 2	0 0 -2 -4 -4	1 -1 -2 1 3	2 2 2 2 2
	0 0 0 0 0	1 1 2 5 1	0 0 0 0 0	0 0 0 0 0		3 4 2 2 0	0 0 2 2 1	1 -1 -2 3 0	1 2 2 1 0
	0 0 0 0 0	1 4 2 2 5	0 0 0 0 0	0 0 0 0 0		3 2 2 0 0	0 0 -2 2 5	1 -1 4 0 0	1 -1 1 0 0
1	4 1 0 0 0	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0	7	3 3 3 3 3	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0
	2 0 0 0 0	0 2 2 4 2	1 0 0 0 0	1 0 0 0 0		3 3 3 3 3	0 -1 -2 1 -1	1 3 4 3 3	1 1 1 1 1
	0 0 0 0 0	2 4 2 4 2	0 0 0 0 0	0 0 0 0 0		3 3 4 1 2	0 0 -2 -4 -2	1 -1 -2 1 -1	2 2 2 2 2
	0 0 0 0 0	1 1 2 5 1	0 0 0 0 0	0 0 0 0 0		3 3 2 2 2	0 0 0 0 -2	1 -1 -2 1 3	1 2 -1 2 1
	0 0 0 0 0	1 4 2 2 5	0 0 0 0 0	0 0 0 0 0		3 3 2 2 0	0 0 -4 -1 5	1 -1 -2 3 0	1 -1 -1 1 0
2	3 3 1 0 0	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0	8	3 3 3 3 3	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0
	4 2 0 0 0	0 -1 2 4 2	1 3 0 0 0	1 1 0 0 0		3 3 3 3 3	0 -1 -2 1 -1	1 3 4 3 3	1 1 1 1 1
	2 0 0 0 0	0 4 2 4 2	1 0 0 0 0	2 0 0 0 0		3 3 3 3 1	0 0 -2 -4 -2	1 -1 -2 1 -1	2 2 2 2 2
	0 0 0 0 0	1 1 2 5 1	0 0 0 0 0	0 0 0 0 0		3 3 4 2 2	0 0 0 -4 0	1 -1 -2 -4 -1	1 2 -1 -1 2
	0 0 0 0 0	1 4 2 2 5	0 0 0 0 0	0 0 0 0 0		3 3 2 2 2	0 0 0 0 2	1 -1 2 1 3	1 -1 2 -1 1
3	3 3 3 1 0	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0	9	3 3 3 3 3	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0
	3 0 2 0 0	0 -1 -2 4 2	1 3 4 0 0	1 1 1 0 0		3 3 3 3 3	0 -1 -2 1 -1	1 3 4 3 3	1 1 1 1 1
	4 2 0 0 0	0 -2 2 4 2	1 3 0 0 0	2 2 0 0 0		3 3 3 3 3	0 0 -2 -4 -2	1 -1 -2 1 -1	2 2 2 2 2
	2 0 0 0 0	0 1 2 5 1	1 0 0 0 0	1 0 0 0 0		3 3 3 0 2	0 0 0 -4 -2	1 -1 -2 -4 -2	1 2 -1 -1 -1
	0 0 0 0 0	1 4 2 2 5	0 0 0 0 0	0 0 0 0 0		3 3 3 2 2	0 0 0 8 1	1 -1 2 -4 -1	1 -1 2 2 -1
4	3 3 3 3 1	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0	10	3 3 3 3 3	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0
	3 4 1 2 0	0 -1 -2 1 2	1 3 4 3 0	1 1 1 1 0		3 3 3 3 3	0 -1 -2 1 -1	1 3 4 3 3	1 1 1 1 1
	3 2 2 0 0	0 0 -6 4 2	1 -1 4 0 0	2 2 2 0 0		3 3 3 3 3	0 0 -2 -4 -2	1 -1 -2 1 -1	2 2 2 2 2
	4 2 0 0 0	0 -2 2 5 1	1 3 0 0 0	1 1 0 0 0		3 3 3 4 1	0 0 0 -4 -2	1 -1 -2 -4 -2	1 2 -1 -1 -1
	2 0 0 0 0	0 4 2 2 5	1 0 0 0 0	1 0 0 0 0		3 3 3 2 2	0 0 0 0 5	1 -1 2 -8 -2	1 -1 2 -2 2
5	3 3 3 3 3	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0	11	3 3 3 3 3	1 3 4 3 3	0 0 0 0 0	0 0 0 0 0
	3 3 3 1 2	0 -1 -2 1 -1	1 3 4 3 3	1 1 1 1 1		3 3 3 3 3	0 -1 -2 1 -1	1 3 4 3 3	1 1 1 1 1
	3 4 2 2 0	0 0 -2 -2 2	1 -1 -2 3 0	2 2 2 2 0		3 3 3 3 3	0 0 -2 -4 -2	1 -1 -2 1 -1	2 2 2 2 2
	3 2 2 0 0	0 0 -2 5 1	1 -1 4 0 0	1 2 1 0 0		3 3 3 3 3	0 0 0 -4 -2	1 -1 -2 -4 -2	1 2 -1 -1 -1
	3 2 0 0 0	0 1 2 2 5	1 3 0 0 0	1 1 0 0 0		3 3 3 3 3	0 0 0 0 1	1 -1 2 -8 -8	1 -1 2 -2 2

Таблица 13

ЛФП простого алгоритма вычисления детерминанта

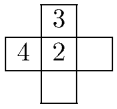
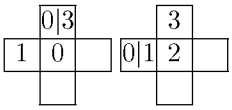
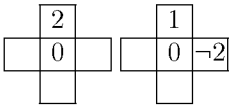
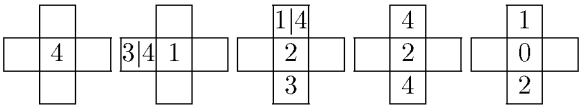
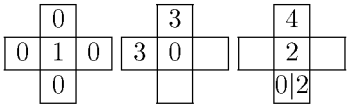
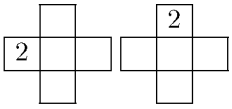
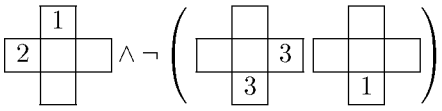
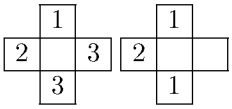
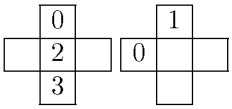
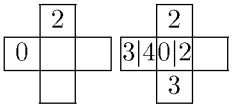
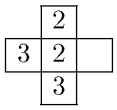
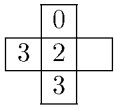
№ п/п	Условие перехода	Правило перехода
1		$s := 0$
		$s := 1$
		$s := 2$
		$s := 3$
		$s := 4$
2		$a := a - c_{\leftarrow} \cdot b_{\uparrow}$, $b := b_{\uparrow}$, $c := c_{\leftarrow}$
		$a := a - c_{\leftarrow} \cdot a_{\uparrow}$, $b := a_{\uparrow}$, $c := c_{\leftarrow}$
		$a := a - c_{\leftarrow} \cdot a_{\uparrow}$, $b := b_{\leftarrow} \cdot (a - c_{\leftarrow} \cdot a_{\uparrow})$
		$a := 0$, $c := a/a_{\uparrow}$, $b := a_{\uparrow}$
		$a := 0$, $c := a/b_{\uparrow}$, $b := b_{\uparrow}$
		$a := 0$, $c := a/b_{\uparrow}$, $b := b_{\leftarrow} \cdot b_{\uparrow}$
		$a := 0$, $c := a/a_{\uparrow}$, $b := b_{\leftarrow} \cdot a_{\uparrow}$

Таблица 14

ЛФП алгоритма нахождения детерминанта с выбором главного элемента

Условие перехода		Правило перехода
$(s=-1)\wedge(s_\uparrow=0)\wedge(\lvert a\rvert>\lvert a_\uparrow\rvert)$		$s:=-4$
$(s=-1)\wedge(s_\uparrow=0)\wedge\neg(\lvert a\rvert>\lvert a_\uparrow\rvert)$		$s:=-3$
$(s=-1)\wedge(s_\uparrow=\#)$		$s:=1$
$(s=-2)\wedge(s_\uparrow\in\{0,-3\})\wedge(\lvert a\rvert>\lvert a_\uparrow\rvert)$		$s:=-4$
$(s=-2)\wedge(s_\uparrow\in\{0,-3,-4\})\wedge\neg(\lvert a\rvert>\lvert a_\uparrow\rvert)$		$s:=0$
$(s=0)\wedge(s_\uparrow\in\{0,-3\})\wedge(s_\downarrow\in\{-1,-2\})$ $\wedge(s_\leftarrow\notin\{-4,-5,1,2,3,4\})$		$s:=-2$
$(s=0)\wedge(s_\uparrow=\#)\wedge(s_\downarrow\in\{-1,-2\})$ $\wedge(s_\leftarrow\notin\{-4,-5,1,2,3,4\})$		$s:=1$
$(s=0)\wedge(s_\uparrow\notin\{1,3,4\})\wedge(s_\leftarrow\in\{-4,-5\})$		$s:=-5$
$(s=0)\wedge(s_\uparrow\notin\{1,3,4\})\wedge(s_\leftarrow\in\{1,2\})$		$s:=2$
$(s=0)\wedge(s_\uparrow\notin\{1,3\})\wedge(s_\leftarrow\in\{3,4\})\wedge(s_\downarrow\notin\{\#,2\})$		$s:=4$
$(s=0)\wedge(s_\uparrow\in\{1,3\})$		$s:=3$
$(s=0)\wedge(s_\uparrow=4)\wedge(s_\leftarrow=3)\wedge(s_\downarrow=\#)$		$s:=-1$
$(s=0)\wedge(s_\uparrow=2)\wedge(s_\leftarrow=3)\wedge(s_\downarrow\in\{\#,2\})$		$s:=\#$
$(s=-3)\wedge(s_\uparrow\notin\{3,1\})\wedge(s_\downarrow=-2)$		$s:=1$
$(s=-3)\wedge(s_\uparrow\in\{3,1\})$		$s:=3$
$(s=-4)\wedge(s_\uparrow\neq1)\wedge(s_\downarrow\neq-2)$		$s:=-3$
$(s=-4)\wedge(s_\uparrow\neq1)\wedge(s_\downarrow=-2)$		$s:=1$
$(s=-4)\wedge(s_\uparrow=1)$		$s:=3$
$(s=-5)\wedge(s_\leftarrow\notin\{3,4\})$		$s:=0$
$(s=-5)\wedge(s_\leftarrow\in\{3,4\})\wedge\neg((s_\uparrow=2)\wedge(s_\downarrow\in\{\#,2\}))$		$s:=4$
$(s=-5)\wedge(s_\leftarrow=3)\wedge(s_\uparrow=2)\wedge(s_\downarrow\in\{\#,2\})$		$s:=\#$
$s=1$		$s:=\#$
$s\in\{2,3\}$		$s:=\#$
$s=4$		$s:=0$
$(s\in\{-1,-2\})\wedge(s_\leftarrow\neq1)\wedge(s_\uparrow\in\{0,-3,-4\})\wedge(\lvert a\rvert>\lvert a_\uparrow\rvert)$		$a:=-a_\uparrow$
$(s=0)\wedge(s_\leftarrow\notin\{1,2\})\wedge(s_\uparrow\notin\{1,2,3,4\})\wedge(s_\downarrow=-5)$		$a:=a_\downarrow$
$(s=0)\wedge(s_\leftarrow\in\{1,2\})$		$b:=a$
$(s=0)\wedge(s_\leftarrow\notin\{1,2\})\wedge(s_\uparrow=1)$	Всегда	$a:=0,\ b:=a_\uparrow$
	$a_\uparrow=0$	$c:=0$
	$a_\uparrow\neq0$	$c:=a/a_\uparrow$
$(s=0)\wedge(s_\leftarrow\notin\{1,2\})\wedge(s_\uparrow=3)$	Всегда	$a:=0,\ b:=b_\uparrow$
	$b_\uparrow=0$	$c:=0$
	$b_\uparrow\neq0$	$c:=a/b_\uparrow$
$(s=0)\wedge(s_\leftarrow\notin\{1,2\})\wedge(s_\uparrow\in\{2,4\})$ $\wedge\neg((s_\uparrow=2)\wedge(s_\downarrow\in\{\#,2\}))$		$a:=a-b_\uparrow\cdot c_\leftarrow,$ $b:=b_\uparrow,\ c:=c_\leftarrow$
$(s=0)\wedge(s_\leftarrow=3)\wedge(s_\uparrow=2)\wedge(s_\downarrow\in\{\#,2\})$		$a:=a-b_\uparrow\cdot c_\leftarrow,$ $b:=b_\leftarrow\cdot(a-b_\uparrow\cdot c_\leftarrow)$
$(s=0)\wedge(s_\leftarrow\notin\{1,2\})\wedge(s_\uparrow\notin\{1,2,3,4\})$ $\wedge(s_\downarrow\in\{-1,-2\})\wedge(\lvert a_\downarrow\rvert>\lvert a\rvert)$		$a:=a_\downarrow$

Окончание табл. 14

Условие перехода		Правило перехода
$(s = -3) \wedge (s_\uparrow \notin \{1, 3\}) \wedge (s_\downarrow \in \{-1, -2\}) \wedge (\lvert a_\downarrow \rvert > \lvert a \rvert)$		$a := a_\downarrow$
$(s = -3) \wedge (s_\uparrow = 1) \wedge \neg((s_\downarrow = \#) \wedge (s_\leftarrow = \#))$	Всегда	$a := 0,\ b := a_\uparrow$
	$a_\uparrow = 0$	$c := 0$
	$a_\uparrow \neq 0$	$c := a/a_\uparrow$
$(s = -3) \wedge (s_\uparrow = 1) \wedge (s_\downarrow = \#) \wedge (s_\leftarrow = \#)$	Всегда	$a := 0,\ b := a_\uparrow \cdot b_\leftarrow$
	$a_\uparrow = 0$	$c := 0$
	$a_\uparrow \neq 0$	$c := a/a_\uparrow$
$(s = -3) \wedge (s_\uparrow = 3) \wedge \neg((s_\downarrow = \#) \wedge (s_\leftarrow = \#))$	Всегда	$a := 0,\ b := b_\uparrow$
	$b_\uparrow = 0$	$c := 0$
	$b_\uparrow \neq 0$	$c := a/b_\uparrow$
$(s = -3) \wedge (s_\uparrow = 3) \wedge (s_\downarrow = \#) \wedge (s_\leftarrow = \#)$	Всегда	$a := 0,\ b := b_\uparrow \cdot b_\leftarrow$
	$b_\uparrow = 0$	$c := 0$
	$b_\uparrow \neq 0$	$c := a/b_\uparrow$
$(s = -4) \wedge (s_\uparrow \neq 1) \wedge (s_\downarrow \in \{-1, -2\}) \wedge (\lvert a_\downarrow \rvert > \lvert a \rvert)$		$a := a_\downarrow$
$(s = -4) \wedge (s_\uparrow = 1) \wedge \neg((s_\downarrow = \#) \wedge (s_\leftarrow = \#))$	Всегда	$a := 0,\ b := a_\uparrow$
	$a_\uparrow = 0$	$c := 0$
	$a_\uparrow \neq 0$	$c := a/a_\uparrow$
$(s = -4) \wedge (s_\uparrow = 1) \wedge (s_\downarrow = \#) \wedge (s_\leftarrow = \#)$	Всегда	$a := 0,\ b := a_\uparrow \cdot b_\leftarrow$
	$a_\uparrow = 0$	$c := 0$
	$a_\uparrow \neq 0$	$c := a/a_\uparrow$
$(s = -5) \wedge (s_\uparrow \neq 2)$		$a := -a_\uparrow$
$(s = -5) \wedge (s_\uparrow = 2) \wedge \neg((s_\uparrow = 2) \wedge (s_\downarrow \in \{\#, 2\}))$		$a := -a_\uparrow - a \cdot c_\leftarrow,$ $b := a,\ c := c_\leftarrow$
$(s = -5) \wedge (s_\uparrow = 2) \wedge (s_\leftarrow = 3) \wedge (s_\downarrow \in \{\#, 2\})$		$a := -a_\uparrow - a \cdot c_\leftarrow,$ $b := b_\leftarrow \cdot (-a_\uparrow - a \cdot c_\leftarrow)$
$(s = 2) \wedge (s_\downarrow = -5)$		$a := a_\downarrow$

Заключение

Из предложенных алгоритмов обратим внимание на алгоритм 3 матричного умножения и модификацию алгоритма вычисления определителя как наиболее универсальные. Так как алгоритм 3 достигает асимптотически минимальной сложности $\Theta(n)$ для топологии двумерной решётки [13], задачу умножения матриц на клеточном автомате можно считать решённой. Иначе обстоит дело с нахождением детерминанта. Как уже отмечено, модифицированный алгоритм не является оптимальным с точки зрения временны́х затрат, а простой алгоритм не определён для некоторых матриц. Более того, данные алгоритмы используют операцию деления, что ограничивает их общность. Хотелось бы найти алгоритм, опирающийся на формулу полного разложения, преобразованную вынесением общих множителей таким образом, чтобы количество слагаемых было минимально. Подобный подход был бы применим для матрицы с элементами в произвольном кольце. Выразим надежду, что такой КА-алгоритм существует и его можно найти.

ЛИТЕРАТУРА

1. *Legendi T. et al.* Megacell machine // Parallel Computing. 1988. V. 8. No. 1–3. P. 195–199.
2. *Kramer P. P. G. and van Leeuwen J.* Systolic computation and VLSI // Foundations of Computer Science IV. P. 1. / eds. J. W. de Bakker and J. van Leeuven. Amsterdam, 1983. P. 75–103.
3. *Матюшкин И. В., Заплетина М. А.* Отражение и транспонирование данных в матрице клеточно-автоматного вычислителя // Изв. вузов. Электроника. 2019. Т. 24. № 1. С. 51–63.
4. *Матюшкин И. В., Жемерикин А. В., Заплетина М. А.* Клеточно-автоматные алгоритмы сортировки строк и умножения целых чисел по схеме Атрубина // Изв. вузов. Электроника. 2016. Т. 21. № 6. С. 557–565.
5. *Матюшкин И. В., Кожевников В. С.* Клеточно-автоматные алгоритмы пермутации матриц // Труды МФТИ. 2019. Т. 11. № 1. С. 39–52.
6. *Гергель В. П.* Теория и практика параллельных вычислений: учеб. пособие. М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. 423 с.
7. *Кудрявцев В. Б., Подколзин А. С., Болотов А. А.* Основы теории однородных структур. М.: Наука, 1990. 296 с.
8. *Strassen V.* Gaussian elimination is not optimal // Numer. Math. 1969. V. 13. No. 4. P. 354–356.
9. *Cohn H., Kleinberg R., Szegedy B., and Umans C.* Group-theoretic algorithms for matrix multiplication // Proc. Ann. IEEE Symp. FOCS. 2005. P. 379–388.
10. *Фельдман Л. П., Назарова И. А., Хорошилов А. В.* Параллельные блочные алгоритмы умножения матриц для мультикомпьютеров с распределенной памятью // Наукові праці Донецького національного технічного університету. Серія «Інформатика, кібернетика та обчислювальна техніка». 2007. Вып. 8(120). С. 297–308.
11. *Katona E.* Cellular algorithms for binary matrix operations // LNCS. 1981. V. 111. P. 203–216.
12. *Stojanovic N. M., Milovanovic I. Z., Stojcev M. K., and Milovanovic E. I.* Matrix-vector multiplication on a fixed size unidirectional systolic array // 8th Intern. Conf. on Telecommunications in Modern Satellite, Cable and Broadcasting Services, Nis, Serbia. 2007. P. 457–460.
13. *Gentleman W. M.* Some complexity results for matrix computations on parallel processors // J. ACM. 1978. V. 25. No. 1. P. 112–115.
14. *Dekel E., Nassimi D., and Sahni S.* Parallel matrix and graph algorithms // SIAM J. Computing. 1981. V. 10. No. 4. P. 657–675.
15. *Moraga C.* On a case of symbiosis between systolic arrays // Integration. 1984. V. 2. No. 3. P. 243–253.
16. *Umeo H.* Firing squad synchronization algorithms for two-dimensional cellular automata // J. Cellular Automata. 2009. V. 4. No. 1. P. 1–20.
17. *Mazoyer J.* A six-state minimal time solution to the firing squad synchronization problem // Theor. Comput. Sci. 1987. V. 50. No. 2. P. 183–238.
18. *Беклемишев Д. В.* Курс аналитической геометрии и линейной алгебры: учебник для вузов. 10-е изд., испр. М.: Физматлит, 2005. 304 с.
19. *Аршон С.* Обобщенное правило Саррюса // Матем. сб. 1935. Т. 42. № 1. С. 121–128.
20. *Mahajan M. and Vinay V.* Determinant: Combinatorics, Algorithms, and Complexity. Chicago J. Theor. Comput. Sci., 1997.
21. *Csanky L.* Fast parallel inversion algorithms // SIAM J. Computing. 1976. V. 5. No. 4. P. 818–823.

22. *Berkowitz S. J.* On computing the determinant in small parallel time using a small number of processors // Inform. Processing Lett. 1984. V. 18. No. 3. P. 147–150.
23. *Beliakov G. and Matiyasevich Y.* A parallel algorithm for calculation of determinants and minors using arbitrary precision arithmetic // BIT Numerical Math. 2015. V. 56. No. 1. P. 33–50.
24. *Marco A. and Martínez J.-J.* Parallel computation of determinants of matrices with polynomial entries // J. Symbolic Computation. 2004. V. 37. No. 6. P. 749–760.
25. *Almalki S., Alzahrani S., and Alabdullatif A.* New parallel algorithms for finding determinants of $N \times N$ matrices // Proc. World Congress on Computer and Information Technology (WCCIT), 2013. P. 1–6.
26. `www.mathworks.com/help/matlab/ref/det.html` — The MathWorks, Inc., 2019.

REFERENCES

1. *Legendi T. et al.* Megacell machine. Parallel Computing, 1988, vol. 8, no. 1–3, pp. 195–199.
2. *Kramer P. P. G. and van Leeuwen J.* Systolic computation and VLSI. Foundations of Computer Science IV, part 1 (eds. J. W. de Bakker and J. van Leeuven). Amsterdam, 1983, pp. 75–103.
3. *Matushkin I. V. and Zapletina M. A.* Otrazheniye i transponirovaniye dannykh v matritse kletochno-avtomatnogo vychislitelya [Data reflection and transposition in the matrix of a cellular automata computator]. Proc. Universities. Electronics, 2019, vol. 24, no. 1, pp. 51–63. (in Russian)
4. *Matushkin I. V., Zhemerikin A. V., and Zapletina M. A.* Kletochno-avtomatnyye algoritmy sortirovki strok i umnozheniya tselykh chisel po skheme Atrubina [Cellular Automata Algorithms for String Sorting and Integer Multiplication by Atrubin's Scheme]. Proc. Universities. Electronics, 2016, vol. 21, no. 6, pp. 557–565. (in Russian)
5. *Matyushkin I. V. and Kozhevnikov V. S.* Kletochno-avtomatnyye algoritmy permutatsii matrits [Cellular automata algorithms for matrix permutations]. Proc. MIPT, 2019, vol. 11, no. 1, pp. 39–52. (in Russian)
6. *Gergel V. P.* Teoriya i praktika parallel'nykh vychisleniy [Theory and Practice of Parallel Computing]. Moscow, Internet-Universitet Informatsionnykh Tekhnologiy; BINOM Publ., 2007. 423 p. (in Russian)
7. *Kudryavtsev V. B., Podkolzin A. S., and Bolotov A. A.* Osnovy teorii odnorodnykh struktur [Basics of the Theory of Homogeneous Structures]. Moscow, Nauka Publ., 1990. 296 p. (in Russian)
8. *Strassen V.* Gaussian elimination is not optimal. Numer. Math., 1969, vol. 13, no. 4, pp. 354–356.
9. *Cohn H., Kleinberg R., Szegedy B., and Umans C.* Group-theoretic algorithms for matrix multiplication. Proc. Ann. IEEE Symp. FOCS, 2005, pp. 379–388.
10. *Feldman L. P., Nazarova I. A., and Horoshilov A. V.* Parallel'nyye blochnyye algoritmy umnozheniya matrits dlya mul'tikomp'yuterov s raspredelennoy pamyat'yu [Parallel block algorithms of matrix multiplication for computer systems with distributed memories]. Naukovi pratsi Donets'kogo natsional'nogo tekhnichnogo universitetu, seriya "Informatika, kibernetika ta obchislyuval'na tekhnika", 2007, vol. 8, no. 120, pp. 297–308. (in Russian)
11. *Katona E.* Cellular algorithms for binary matrix operations. LNCS, 1981, vol. 111, pp. 203–216.
12. *Stojanovic N. M., Milovanovic I. Z., Stojcev M. K., and Milovanovic E. I.* Matrix-vector multiplication on a fixed size unidirectional systolic array. 8th Intern. Conf. on Telecommunications in Modern Satellite, Cable and Broadcasting Services, Nis, Serbia, 2007, pp. 457–460.

13. *Gentleman W. M.* Some complexity results for matrix computations on parallel processors. J. ACM, 1978, vol. 25, no. 1, pp. 112–115.
14. *Dekel E., Nassimi D., and Sahni S.* Parallel matrix and graph algorithms. SIAM J. Computing, 1981, vol. 10, no. 4, pp. 657–675.
15. *Moraga C.* On a case of symbiosis between systolic arrays. Integration, 1984, vol. 2, no. 3, pp. 243–253.
16. *Umeo H.* Firing squad synchronization algorithms for two-dimensional cellular automata. J. Cellular Automata, 2009, vol. 4, no. 1, pp. 1–20.
17. *Mazoyer J.* A six-state minimal time solution to the firing squad synchronization problem. Theor. Comput. Sci., 1987, vol. 50, no. 2, pp. 183–238.
18. *Beklemishev D. V.* Kurs analiticheskoy geometrii i lineynoy algebry [A course of analytic geometry and linear algebra]. Moscow, Fizmatlit Publ., 2005. 304 p. (in Russian)
19. *Arshon S.* Obobshchennoye pravilo Sarryusa [Generalized Sarrus' rule]. Matematicheskiy Sbornik, 1935, vol. 42, no. 1, pp. 121–128. (in Russian)
20. *Mahajan M. and Vinay V.* Determinant: Combinatorics, Algorithms, and Complexity. Chicago J. Theor. Comput. Sci., 1997.
21. *Csanky L.* Fast parallel inversion algorithms. SIAM J. Computing, 1976, vol. 5, no. 4, pp. 818–823.
22. *Berkowitz S. J.* On computing the determinant in small parallel time using a small number of processors. Inform. Processing Lett., 1984, vol. 18, no. 3, pp. 147–150.
23. *Beliakov G. and Matiyasevich Y.* A parallel algorithm for calculation of determinants and minors using arbitrary precision arithmetic. BIT Numerical Math., 2015, vol. 56, no. 1, pp. 33–50.
24. *Marco A. and Martínez J.-J.* Parallel computation of determinants of matrices with polynomial entries. J. Symbolic Computation, 2004, vol. 37, no. 6, pp. 749–760.
25. *Almalki S., Alzahrani S., and Alabdullatif A.* New parallel algorithms for finding determinants of $N \times N$ matrices. Proc. World Congress on Computer and Information Technology (WCCIT), 2013, pp. 1–6.
26. `www.mathworks.com/help/matlab/ref/det.html` — The MathWorks, Inc., 2019.