УДК 519.7+519.683

Алгоритмы параллельных вычислений в формализации клеточных автоматов: сортировка строк и умножение чисел по схеме Атрубина

И.В.Матюшкин

Институт проблем проектирования в микроэлектронике РАН

Национальный исследовательский университет «МИЭТ»

АО «НИИМЭ и Микрон», imatyushkin@ippm.ru

Аннотация — Актуальной задачей последних лет является вычислительный параллелизм, в том числе на уровне алгоритмов. Переформулировка алгоритма на язык клеточных автоматов (КА) – один из путей решения этой задачи. В данной статье показаны алгоритмы одномерной и двумерной сортировки, а также алгоритм произведения двух чисел.

Ключевые слова — клеточные автоматы, одномерная сортировка, двумерная сортировка, КА-произведение двух целых чисел.

I. Введение

С появлением многоядерных процессоров (и других нетрадиционных архитектур) актуальность прибрела задача адаптации вычислительных алгоритмов к их структуре, подразумевающая значительную степень параллелизма обработки информации. Один из путей решения этой задачи состоит в использовании языка клеточных автоматов (КА) при формулировке параллельного алгоритма. Далее будут рассмотрены три типовых задачи, встречающихся в практике: сортировка символов, сортировка строк (будем называть её двумерной сортировкой) и умножение чисел в системе с основанием N по способу Атрубина (1965).

Заметим, что при проведении поиска литературы за обозримый период нами не найдено чёткой формулировки КА-алгоритма сортировки. Алгоритм Атрубина был сформулирован для систолического массива процессоров и использует открытость системы (данные вводят через поток в буфере, а результаты тоже выводятся в потоки). Таким образом, рафинированная КА-формулировка также отсутствует в литературе. При переформулировке нам удалось улучшить алгоритм Атрубина по числу используемых регистров на ячейку (элементарный процессор): с 5 до 4 регистров (компонент).

II. Одномерная сортировка

В общей формулировке требуется упорядочить массив однородных объектов, для которых задано бинарное отношение сравнения. Без ограничения общности будем считать, что объект изоморфен символу, а символы упорядочены лексикографически, т.е. $\forall a,b \in \mathrm{A}\ \ (a \le b), \le$ есть отношение нестрогого порядка (соответствующая функция возвращает 0 или 1).

Простейшее КА-решение, использующее блочный механизм Марголуса, основано на идее «всплытия» пузырьком. Блочный автомат Марголуса представим, как известно, через классический синхронный КА. Сразу заметим, что более развитые методы сортировки, стремясь к минимизации проходов по массиву, в той или иной мере используют глобальную информацию, что делает их сомнительными для использования в КА.

Исходное предположение: ячейка КА может сравнивать два символа, каждый сортируемый символ первоначально «вписан» в ячейку. Количество ячеек – N, ячейки нумеруются индексом $0 \le i \le N-1$. Состояние ячейки представлено двумя компонентами $\langle S, F \rangle$: S – целое число (буква), F – булево. Соседство: 3 ячейки (или 2 для граничных): i -я – центральная, $i-1$, $i+1$ – боковые. Границы не замкнутые.

Начальные состояния для каждой из ячеек:

- S : случайное целое число/буква (или введенное пользователем);
- F : $(i \mathrm{div} 2) \bmod 2$, где «div» – целочисленное деление, “mod” – остаток от деления по модулю 2.

Правила перехода:

$$S_i := (F_i = F_{i+1}) * ((S_i > S_{i+1}) * S_{i+1} + (S_i \le S_{i+1}) * S_i) +$$
$$+ (F_i = F_{i-1}) * ((S_i < S_{i-1}) * S_{i-1} + (S_i \ge S_{i-1}) * S_i),$$
$$F_i := F_{i-1}.$$

Правила для граничных ячеек:

- Для 0-й ячейки:

$$S_0 := (F_0 = F_1) * \big((S_0 > S_1) * S_1 + (S_0 \le S_1) * S_0\big),$$
$$F_0 := 1 - F_1.$$

- Для ($N-1$)-й ячейки:

$$S_0 := (F_{N-1} = F_{N-2}) * (S_{N-1} < S_{N-2}) * S_{N-2},$$
$$F_{N-1} := F_{N-2}.$$

На рис. 1 показана диаграмма для такого КА, сортирующего массив символов. Когда состояние автомата перестанет изменяться, тогда наступает останов. При строгом понимании вычислимости нужно технически обеспечить это требование, что на самом деле может представлять неожиданную сложность. Сложность сортировки пузырьком, как известно, составляет $O(N^2)$. Описанный выше КА дает эмпирически $O(N)$, точнее даже $0.8N$ при $N < 100$. Примечательно, что для наихудшего случая, когда наименьший элемент находится в конце поля, за время его миграции почти полностью завершаются все перестройки. На рисунке 1 легко различимы траектории движения (всплытия) подобных элементов. Нужно осознавать, что в отличие от классической модели вычислений мы не можем использовать центральный процессорный элемент, мы лишены возможности переносить данные между удаленными регистрами, и потому феномен таких «движений» (с целью переноса данных) суть «родимое пятно» вычислимости в КА.

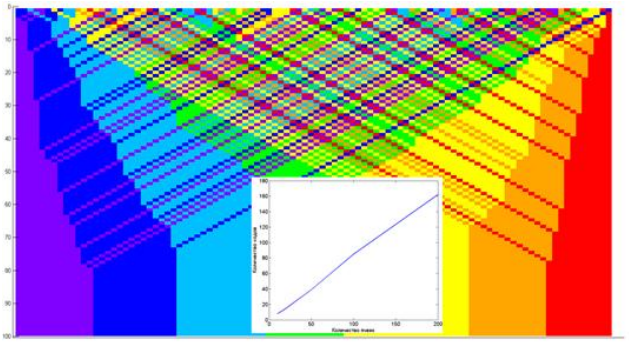

Рис. 1. Диаграмма «время-пространство» (ось времени направлена вниз) для КА сортировки. Поле КА есть массив из N = 100 ячеек, цвет отражает состояние ячейки (7 значений). Начальное состояние случайное. В диаграмму вклеена эмпирическая зависимость числа ходов до останова от размера поля N

III. Двумерная сортировка

Часто сортируемый объект обладает сложной структурой, например, некоторого слова в алфавите A . Алгоритм двумерной сортировки опирается на предыдущий алгоритм и используется для сортировки слов, записанных в некотором массиве из N строк (каждое слово не более M знаков). Пустые ячейки/клетки заполняем пробелами. Состояние ячейки КА имеет три компонента: символ данных $A \in A$, а также 2 флага с тремя состояниями. Флаг сравнения $W \in \{0,1,2\}$ со следующей семантикой: 0 – клетки не сравнивались (по компоненту A), 1 – символы сравнивалась и не равны, 2 – символы сравнивалась и равны. Флаг блочности $K \in \{1,2\}$ разделяет наши слова на блоки (как и в КА Марголуса на разбиении) и выделяет активный столбец, т. е. сравнение разрешено только между ячейками с одинаковым значением этого флага. Ячейка КА имеет окрестность фон Неймана. В начальном состоянии поле КА состоит из букв (и пробелов; пробел в отношении порядка предшествует любому символу алфавита) со всеми нулевыми флагами, кроме первого столбца. В первом столбце стоят флаги блочности и активности так, чтобы разбить слова по парам.

Идея алгоритма состоит в следующем. Назовём фронтом колонку с ненулевыми K ; в каждый момент времени фронт единственен и перемещается (вправо) от столбца с первыми до столбца с последними буквами слов. При переходе с последнего столбца на первый флаг K изменяется так, чтобы сместить активные блоки и тем самым завершить попарное сравнение слов. Фронт по флагу W следует (слева) за фронтом по флагу K . От значения флага W зависит, нужно нам сравнивать символы или нет. Если символы не равны, то ячейка получает значение флага $W = 1$. По результату сравнения мы меняем буквы. Если клетка слева уже имеет $W = 1$, то нет смысла производить дальнейшее сравнение, тогда значение флага W наследуется.

Таким образом, начальное состояние определяется формулой:

$$W(i,j) = 0, A(i,j) \in A, K(i,0) = (i \mathrm{div} 2) \bmod 2 + 1,$$
$$0 \le i < N, 0 \le j < M.$$

Целочисленные операции определяются так, например: $7 div 2 = 3$, $7 mod 2 = 1$. Знак «:=» означает присвоение, и величины слева от него берутся в момент времени ($t+1$).

Локальная функция перехода:

Вначале зададим переход для флага блочности.

$$K(i,j) := \begin{cases} K(i,j-1), & j \ne 0; \\ K(i,N-1) \bmod 2 + 1, & \\ (j=0) \wedge (i \bmod 2 = 0) \wedge (K(i,N-1) \ne 0); & \\ K(i,N-1), & \\ (j=0) \wedge (i \bmod 2 = 1) \wedge (K(i,N-1) \ne 0); & \\ K(i,j) & \text{otherwise.} \end{cases}$$

Последние три формулы, т.к. происходит обращение к абсолютному (для ячейки) индексу i , необходимо трактовать как спецификацию граничной ЛФП для ячеек четных и нечетных номеров, расположенных в самом левом столбце. Именно в силу граничности не происходит нарушения идеологии (локальности и однородности) КА.

В зависимости от флага K определены изменения других компонентов. Для $K(i,j) = 0$ – ничего не

делать. Для $K \in \{1,2\}$ пусть

$$i' := \begin{cases} i+1, & K(i+1,j) = K(i,j); \\ i-1, & K(i-1,j) = K(i,j). \end{cases}$$

Если $i' = \varnothing$, то ничего не делать. В противном случае

$$W(i,j) := \begin{cases} 0, & (W(i,j) \neq 0); \\ 1, & (A(i,j) \neq A(i',j)) \vee (W(i,j-1) = 1); \\ 2, & A(i,j) = A(i',j). \end{cases}$$

$$A(i,j) := \begin{cases} A(i',j), & (W(i,j) = W(i',j) = 1) \wedge ((i > i') \wedge \\ \wedge (A(i,j) < A(i',j))) \vee ((A(i,j) > A(i',j)) \wedge \\ \wedge (i < i')); \\ A(i,j), & W(i,j) \neq W(i',j). \end{cases}$$

При моделировании (см. табл.1) использовалась оригинальная программа SoftCAM. Первая цифра индекса возле буквы – это флаг K, а вторая – W.

Таблица 1

Пример работы алгоритма одномерной сортировки

Инициализация КА. 1-я итерация алгоритма двумерной сортировки

t^{10}	a^{00}	b^{00}	l^{00}	e^{00}	00
t^{10}	o^{00}	o^{00}	l^{00}	00	00
b^{20}	r^{00}	a^{00}	i^{00}	n^{00}	00
b^{20}	r^{00}	e^{00}	d^{00}	00	00
h^{10}	a^{00}	t^{00}	00	00	00
a^{10}	p^{00}	p^{00}	l^{00}	e^{00}	00
c^{20}	u^{00}	s^{00}	t^{00}	o^{00}	m^{00}
j^{20}	o^{00}	k^{00}	e^{00}	00	00
m^{10}	o^{00}	n^{00}	k^{00}	e^{00}	y^{00}
b^{10}	a^{00}	t^{00}	m^{00}	a^{00}	n^{00}

2-я итерация алгоритма двумерной сортировки

t^{02}	a^{10}	b^{00}	l^{00}	e^{00}	00
t^{02}	o^{10}	o^{00}	l^{00}	00	00
b^{02}	r^{20}	a^{00}	i^{00}	n^{00}	00
b^{02}	r^{20}	e^{00}	d^{00}	00	00
h^{01}	a^{10}	t^{00}	00	00	00
a^{01}	p^{10}	p^{00}	l^{00}	e^{00}	00
c^{01}	u^{20}	s^{00}	t^{00}	o^{00}	m^{00}
j^{01}	o^{20}	k^{00}	e^{00}	00	00
m^{01}	o^{10}	n^{00}	k^{00}	e^{00}	y^{00}
b^{01}	a^{10}	t^{00}	m^{00}	a^{00}	n^{00}

7-я итерация алгоритма двумерной сортировки

t^{20}	a^{00}	b^{00}	l^{00}	e^{00}	00
t^{10}	o^{00}	o^{00}	l^{00}	00	00
b^{10}	r^{00}	a^{00}	i^{00}	n^{00}	00
b^{20}	r^{00}	e^{00}	d^{00}	00	00
a^{20}	p^{00}	p^{00}	l^{00}	e^{00}	00
h^{10}	a^{00}	t^{00}	00	00	00
c^{10}	u^{00}	s^{00}	t^{00}	o^{00}	m^{00}
j^{20}	o^{00}	k^{00}	e^{00}	00	00
b^{20}	a^{00}	t^{00}	m^{00}	a^{00}	n^{00}
m^{10}	o^{00}	n^{00}	k^{00}	e^{00}	y^{00}

54-я итерация алгоритма двумерной сортировки. Завершение алгоритма

a^{10}	p^{00}	p^{00}	l^{00}	e^{00}	00
b^{10}	a^{00}	t^{00}	m^{00}	a^{00}	n^{00}
b^{20}	r^{00}	a^{00}	i^{00}	n^{00}	00
b^{20}	r^{00}	e^{00}	d^{00}	00	00
c^{10}	u^{00}	s^{00}	t^{00}	o^{00}	00
h^{10}	a^{00}	t^{00}	00	00	00
j^{20}	o^{00}	k^{00}	e^{00}	00	00
m^{20}	o^{00}	n^{00}	k^{00}	e^{00}	y^{00}
t^{10}	a^{00}	b^{00}	l^{00}	e^{00}	00
t^{10}	o^{00}	o^{00}	l^{00}	00	00

Результаты моделирования показывают корректность работы предложенного алгоритма, сложность которого имеет порядок $N*M$.

IV. Клеточный автомат умножения целых чисел

Атрубин показал возможность параллельного умножения скромными средствами одномерных структур (итерированного массива). Опишем модификацию его алгоритма вначале неформально. Но и в таком виде наблюдается сходство с алгоритмами параллельных подстановок и формализм алгорифмов Маркова.

Первоначально в КА записано последовательно 2 числа одной системы счисления. Эти числа разбиваются по разрядам, каждая клетка КА хранит цифру (digit) одного разряда (вторая строка). Кроме чисел-операндов состояние ячейки хранит идентификатор (первая строка), который показывает, к какому операнду относится цифра (a – для первого, b – для второго). Если цифра с идентификатором «a» стоит слева от цифры с идентификатором «b», то производится умножение этих двух чисел, а результат

записывается в стек (третья строка). После в данных клетках меняются местами идентификаторы и цифры. Все результаты умножения складываются, и общий итог записывается в четвёртую строку. Данный алгоритм работает, пока операнды полностью не поменяются местами. После завершения алгоритма в четвёртой строке содержится результат. В таблице 2 показаны все шаги выполнения алгоритма на примере 963*8402 в десятичной системе. Ячейки-столбцы, активные на следующем шагу, показаны серым фоном.

Таблица 2

*Выполнение параллельного умножения чисел по предлагаемому алгоритму в десятичной системе счисления (пример: 963*8402=8091126)*

Инициализация КА

a	a	a	b	b	b	b
9	6	3	8	4	0	2
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Состояние КА после выполнения 1-го шага

a	a	b	a	b	b	b
9	6	8	3	4	0	2
0	0	2	4	0	0	0
0	0	2	4	0	0	0

Состояние КА после выполнения 2-го шага

a	b	a	b	a	b	b
9	8	6	4	3	0	2
0	4	8	1	2	0	0
0	5	0	5	2	0	0

Состояние КА после выполнения 3-го шага

b	a	b	a	b	a	b
8	9	4	6	0	3	2
7	2	2	4	0	0	0
7	7	2	9	2	0	0

Состояние КА после выполнения 4-го шага

b	b	a	b	a	b	a
8	4	9	0	6	2	3
0	3	6	0	0	0	6
8	0	8	9	2	0	6

Состояние КА после выполнения 5-го шага

b	b	b	a	b	a	A
8	4	0	9	2	6	3
0	0	0	0	1	2	0
8	0	8	9	3	2	6

Состояние КА после выполнения 6-го шага. Завершение алгоритма

b	b	b	b	a	a	a
8	4	0	2	9	6	3
0	0	0	1	8	0	0

8	**0**	**9**	**1**	**1**	**2**	**6**

Итак, дадим точную формулировку КА умножения. Пусть первое число содержит n знаков $a_1a_2\ldots a_n$, второе – m знаков $b_1b_2\ldots b_m$, а основание системы $N>1$. КА синхронный, одномерный, с радиусом окрестности 1 (для граничных ячеек ½, т.е. не учитываем «пропавших» соседей). Поля КА индексируется слева направо $0\le i<n+m$. Состояние ячейки определяется 4-хкомпонентной структурой:

$$\text{state}=\begin{pmatrix}\alpha\in\{a,b\}\\ d\in\{0,..,N-1\}\\ D\in\{0,1\}\\ R\in\{0,..,N-1\}\end{pmatrix}=\begin{Bmatrix}\text{флаг принадлежности}\\ \text{цифра числа}\\ \text{перенос в старший знак}\\ \text{цифра ответа}\end{Bmatrix}.$$

В начальной конфигурации числа записываются одно за другим, начиная со старшего разряда, т.е.

$$\alpha_i=\begin{cases}a, 0\le i\\ b, n\le i<n+m\end{cases},\quad d_i=\begin{cases}a_{i+1}, 0\le i\\ b_{i-n+1}, n\le i<n+m\end{cases}.$$

Остальные компоненты равны нулю, т.е. $\forall i\ D_i=0, R_i=0$.

Для внутренних ячеек опишем ЛФП, которая в зависимости от выполнения тех или иных условий меняет вид – ЛФП1, ЛФП2, ЛФП3 (см. табл. 3). Если ни одно из условий не выполнено, состояние ячейки КА сохраняется. Для граничных ячеек ЛФП наследуются, если ввести фиктивного соседа и положить $\text{state}_{-1}/\text{state}_{n+m}\equiv(\alpha_{0/n+m-1},0,0,0)$. Первый переход (ЛФП1) призван корректно провести сложение в ответе и учесть то, что «держим в уме». Два других срабатывают «рядом» в один такт – каждый для своей «колонки» компонентов.

Таблица 3

Условия и переходы состояний КА

№ЛФП	Условия наступления	Переход
1	$\overline{2\vee 3}$	$X\to R_i+D_{i+1}$, $R_i:=X\bmod N,\quad D_i:=X\text{div}N.$
2	$\alpha_i=a,$ $\alpha_{i+1}=b$	$X\to R_i+D_{i+1}+(d_i\cdot d_{i+1})\text{div}N.$ $R_i:=X\bmod N,\quad D_i:=X\text{div}N.$ $\alpha_i:=b,\quad d_i:=d_{i+1}.$
3	$\alpha_i=b,$ $\alpha_{i-1}=a$	$X\to R_i+D_{i+1}+(d_i\cdot d_{i-1})\bmod N.$ $R_i:=X\bmod N,\quad D_i:=X\text{div}N.$ $\alpha_i:=a,\quad d_i:=d_{i-1}.$

БЛАГОДАРНОСТИ

Выражаю благодарность студенту 4-го курса МИЭТ Жемерикину А.В. за базовую идею, послужившую основой для модификации алгоритма Атрубина.

ПОДДЕРЖКА

Работа выполнена в рамках НИР «Исследование перспективных моделей вычислений и реализующих их архитектур высокопроизводительных информационно-вычислительных комплексов нового поколения» по программе фундаментальных исследований ОНИТ РАН «Архитектурно-программные решения и обеспечение безопасности суперкомпьютерных информационно-вычислительных комплексов новых поколений».

ЛИТЕРАТУРА

[1] A.L. Stempkovsky, P.A. Vlasov, G.V. Kozin. Algorithmic Environment for VLSI Design on Cellular Automata // Proceedings of a Joint Symposium : Information Processing and Software, Systems Design Automation, Academy of Sciences of the USSR, Siemens AG, FRG, Moscow, June 5/6, 1990, Springer-Verlag, pp.308-312.

[2] Bidlo, M., Vasicek, Z., Slany, K. Sorting Network Development Using Cellular Automata. - Evolvable Systems: From Biology to Hardware. - 9th International Conference, ICES 2010, York, UK, September 6-8, 2010, Proceedings, LNCS 6274. London: Springer London, 2010. - pp. 85-96.

[3] Fisher, P.C. Generation on primes by one dimentional real time iterative array. - J. ACM 12, 1965, pp.388-394.

[4] Heen O. Efficient constant speed-up for one dimensional cellular automata calculators. - Parallel Computing, 1997. pp. 1663.

[5] Atrubin A.J. A one-dimensional real-time iterative multiplier - IEEE Transactions on Electronic Computers EC-14, 1965. pp. 394-399.

[6] Even, S., Litman, A. A systematic design and explanation of the Atrubin multiplier. - Sequences II. Methods in Communication, Security, and Computer Science. – Springer, New York, 1993, pp. 189-202.

Algorithms of the parallel computations in the formalization of cellular automata: the sorting of strings and the multiplication of numbers by Atrubin's method

I. V. Matyushkin

Institute for Design Problems in Microelectronics, Russia, Moscow

National Research University of Electronic Technology, Russia, Moscow

Molecular Electronics Research Institute, Russia, Moscow

imatyushkin@ippm.ru

Keywords — parallel computations, cellular automata, algorithm, multiplication, sorting, Atrubin's multiplier

ABSTRACT

Clear statements in the language of cellular automata (CA) are given for the algorithms of parallel sorting. We have not found such statements in literature during visible time in view of relative elementary nature of tasks. Sorting by the array of symbols (we call it one-dimensional) is supported by bubble method and famous Margolus automaton. Sorting by the array of strings, which demand the regulation of words (we call it two-dimensional, hence every symbol is written in one CA cell), is supported by one-dimensional sorting. The complexity of algorithms is linear. CA of one-dimensional sorting by the state of a cell use the structure, which is called "bit-symbol". CA of two-dimensional sorting use the structure, which is called "trit-trit-symbol", i.e., two flags (blockiness and comparison), each of them have three states, are joined to data symbol.

We improve the algorithm of two natural numbers multiplication (in numerical system, base of which is N) that was first formulated by Atrubin in 1965 for binary notation. This algorithm demands five registers of processor, because it was primarily adapted to the systolic array of processors. In spite of adaption to the CA language, the introduced algorithm demands four components instead of five, namely the structure, which is called "bit-bit-symbol-symbol". Our algorithm differs from Atrubin's multiplier in the following aspects: 1) in initial state the numbers are considered as they are introduced in the data string (second component); 2) the numbers of one data move through the numbers of another; 3) answer string is dynamically changing, and corresponds to output signal of Atrubin processors.

REFERENCES

[1] A.L. Stempkovsky, P.A. Vlasov, G.V. Kozin. Algorithmic Environment for VLSI Design on Cellular Automata // Proceedings of a Joint Symposium : Information Processing and Software, Systems Design Automation, Academy of Sciences of the USSR, Siemens AG, FRG, Moscow, June 5/6, 1990, Springer-Verlag, pp.308-312.

[2] Bidlo, M., Vasicek, Z., Slany, K. Sorting Network Development Using Cellular Automata. - *Evolvable Systems: From Biology to Hardware. - 9th International Conference, ICES 2010, York, UK, September 6-8, 2010, Proceedings, LNCS 6274. London: Springer London*, 2010. pp. 85-96.

[3] Fisher, P.C. Generation on primes by one dimentional real time iterative array. *J. ACM 12*, 1965, pp.388-394.

[4] Heen O. Efficient constant speed-up for one dimensional cellular automata calculators. *Parallel Computing*, 1997. pp. 1663.

[5] Atrubin A.J. A one-dimensional real-time iterative multiplier. *IEEE Transactions on Electronic Computers EC-14*, 1965. pp. 394-399.

[6] Even, S., Litman, A. A systematic design and explanation of the Atrubin multiplier. *Sequences II. Methods in Communication, Security, and Computer Science*. *Springer, New York*, 1993, pp. 189-202.