P11-2003-184

Е. П. Акишина, В. В. Иванов*, Б. Ф. Костенко

ИЗУЧЕНИЕ СТРУКТУР СИЛЬНОГО ВЫГОРАНИЯ ДВУОКИСИ УРАНА С ПОМОЩЬЮ КЛЕТОЧНЫХ АВТОМАТОВ: АЛГОРИТМЫ И ПРОГРАММЫ

*Объединенный институт ядерных исследований, Дубна,
и Международный Сольвеевский иснтитут физики и химии,
Брюссель

1. Введение

RIM-структуры, образующиеся в результате сильного выгорания двуокиси урана UO_2, привлекают большое внимание в основном из-за их возможного катастрофического влияния на работу современных ядерных электростанций. Изучение таких структур представляет также и научный интерес, так как механизм их формирования и основные характеристики процесса к настоящему времени до конца неясны (см., в частности, [1]–[5] и имеющиеся там ссылки).

В [6] нами был предложен новый подход к моделированию этих процессов, а также к обработке микрофотографий RIM-структур на основе клеточных автоматов (КА). В настоящей работе описаны КА-алгоритмы, использовавшиеся в [6]. В разделе 2 сформулирована концепция клеточного автомата. В следующем разделе изложена процедура преобразования микрофотографий в рабочее поле КА. В разделе 4 представлены алгоритмы КА для извлечения количественных характеристик структур поверхности. Раздел 5 посвящен КА-алгоритмам моделирования пространственно-временной динамики структуры поверхности в процессе выгорания UO_2, а также в результате ее химического травления.

2. Концепция клеточного автомата

Клеточные автоматы возникли в результате многочисленных попыток создать простую математическую модель для описания сложных биологических структур и процессов [7, 8].

Стандартный клеточный автомат представляет собой простую динамическую систему, поведение которой определяется локальными связями между ее элементами (клетками). Схематично КА может быть представлен в виде периодической пространственной решетки, которая построена и функционирует по следующим правилам:

1. Определяются клетки и их возможные дискретные состояния; обычно каждая клетка может находиться в одном из двух состояний, 0 или 1 (существуют КА с большим числом состояний).

2. Определяются взаимодействия между клетками; как правило, каждая клетка может взаимодействовать только с ближайшими соседними клетками (см. рис. 1).

3. Фиксируются "законы", определяющие эволюцию КА. Состояние каждой клетки меняется согласно фиксированным правилам перехода, которые диктуются самой изучаемой проблемой и часто имеют простую функциональную форму; результат перехода зависит от состояния клетки и состояния ее соседей.

4. КА - синхронизированная система, в которой все клетки меняют свои состояния одновременно.

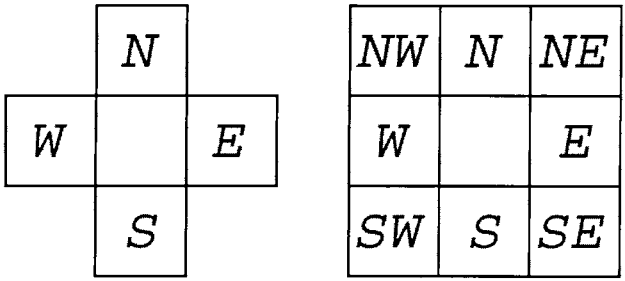

Рис. 1: Соседство фон Неймана (слева) и Мура (справа)

Понятие соседа для решетки, состоящей из квадратов, может быть определено в смысле фон Неймана или Мура. В первом случае принимаются во внимание только соседи Northern (N), Southern (S), Westerly (W) и Easterly (E). В модели Мура дополнительно учитываются и другие соседи, которые соприкасаются с центральной клеткой: NW, NE, SW и SE (рис. 1).

Несмотря на свою простоту, модели КА оказались весьма эффективными для описания различных сложных структур и процессов в физике и биологии [7, 8]. Клеточные автоматы также используются для генерации и обработки цифровых изображений, проектирования компьютерного оборудования и др. [9].

3. Преобразование микрофотографий в цифровой вид

В данной работе использовались микрофотографии поверхности UO_2 (с увеличением в 1250 раз) как "протравленных", так и "отполированных" образцов [6] для степеней выгорания 16.2, 42.6, 54.8 и 65.0 GWd/tM, что отвечает пребыванию топлива в реакторе в течение 323, 953, 1266 и 1642 дней, соответственно. Обычно микрофотографии "отполированной" поверхности используются для оценки пористости материала, а "протравленой" — для визуализации трещин и границ зерен.

Отсканированные микрофотографии обрезались до необходимого размера. В результате были получены цифровые изображения поверхностей образцов, отвечающих разным степеням выгорания топлива. На рис. 2 и 3 представлены микрофотографии "отполированных" образцов для степеней выгорания 16.2 и 65.0 GWd/tM, соответственно.

Далее эти изображения переводились в черно-белые с помощью программы Photoshop путем ручной регулировки яркости и контрастности. Качество данной процедуры можно оценить сравнив рис. 2 и 4 с рис. 3 и 5.

Наконец, черно-белые изображения переводились в цифровые ASCII-файлы с помощью программы Convert [10]: черным и белым точкам (клеткам) на компьютерном экране присваивались, соответственно, значения 0 и 1.

Результирующие цифровые файлы представляют собой рабочие поля КА, используемые в последующей обработке.

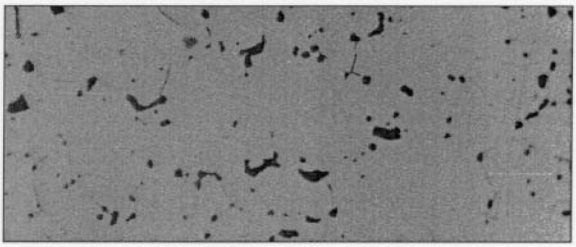

Рис. 2: Отсканированная микрофотография поверхности "отполированного" образца для степени выгорания 16.2 GWd/tM

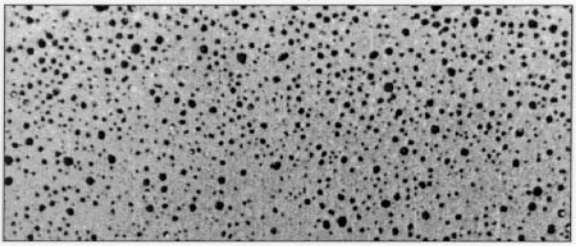

Рис. 3: Отсканированная микрофотография поверхности "отполированного" образца для степени выгорания 65.0 GWd/tM

Рис. 4: Черно-белое представление рис. 2

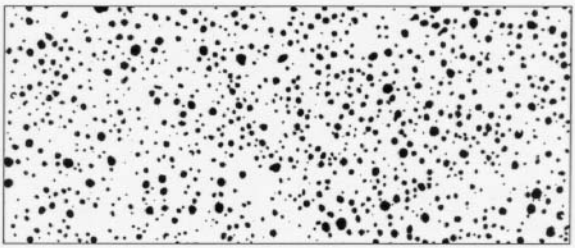

Рис. 5: Черно-белое представление рис. 3

4. Алгоритмы обработки микрофотографий

Для обработки изображений с микрофотографий соответствующие цифровые файлы представлялись в виде матриц, в которых каждый элемент отвечал одной клетке рабочего поля КА. Разработанные алгоритмы использовались для:

- определения количественных характеристик черных пятен;
- отбора пятен с заданным числом клеток и вычисления длин их границ;
- изучения статистической независимости отдельных частей микрофотографий.

4.1. Определение количественных характеристик пятен

Для количественного описания динамики пятен необходимо знать [6]:
а) число клеток в каждом пятне - $count$;
б) координаты центра пятна - (cx, cy);
в) средний радиус пятна - $radius$;
г) максимальный радиус пятна - $radiusmax$.

При вычислении среднего радиуса предполагается, что пятно имеет форму круга и соответствующий радиус вычисляется по формуле $radius = \sqrt{(count/\pi)}$. Максимальный радиус пятна - это максимальное расстояние от центра пятна до его граничной клетки (клетка считается граничной, если она имеет 1, 2 или 3 белых соседей). Для определения количественных характеристик пятен разработан следующий алгоритм.

Алгоритм

1. Микрофотография представляется матрицей $A(n \times m)$, где количество ячеек в каждой строке равняется m, а количество строк - n.
2. Переменной $number$, которая используется для подсчета числа пятен, присваивается значение 0.
3. Матрица $B(n \times m)$ содержит только одно пятно, которое обрабатывается в данный момент.
4. Переменной cur присваивается значение 1. Данная переменная показывает количество клеток в так называемом виртуальном "мешке" (промежуточный массив, в который помещаются только клетки анализируемого пятна), а также номер текущей клетки пятна.
5. Переменной $count$, которая используется для подсчета числа клеток в пятне, присваивается значение 0.
6. Переменной $sumx$, которая используется для подсчета суммы x-координат клеток в пятне, присваивается значение 0.
7. Переменной $sumy$, которая используется для подсчета суммы y-координат клеток в пятне, присваивается значение 0.

8. Всем элементам матрицы B присваивается значение 1.

9. Программа сканирует матрицу A по рядам, начиная с верхнего левого угла, до тех пор пока не будет найдена черная клетка.

10. Текущая клетка помещается в "мешок", для чего ее (i, j)-координаты запоминаются в матрице $Baln(Bal)$ (координата i) и в матрицу $Balm(Bal)$ (координата j).

11. Координаты (x, y) текущей клетки извлекаются из "мешка": $x = Baln(cur)$, $y = Balm(cur)$.

12. Если "мешок" не пустой ($cur \neq 0$), то из него извлекается верхний элемент: $cur = cur - 1$.

13. Проверяется, является ли текущий элемент черным (данная проверка необходима, так как черная клетка может быть помещена в "мешок" несколько раз из-за наличия у нее нескольких черных соседей).

14. Число клеток в пятне увеличивается на 1: $count = count + 1$. Текущая клетка в матрице A заменяется на белую для того, чтобы не учитывать ее повторно, а элементу $B(x, y)$ присваивается значение 0.

15. Сумма координат $sumx$ увеличивается на x, а сумма координат $sumy$ увеличивается на y.

16. Проверяются соседи текущей клетки. Координаты (i,j) всех ее черных соседей запоминаются в матрицах $Baln(Bal)$ и $Balm(Bal)$.

17. Пункты 10, 11, 12, 13, 14, 15 и 16 повторяются до тех пор, пока "мешок" не станет пустым. Это означает, что подсчитаны все клетки текущего пятна.

18. Значения переменных *number* и *count* записываются в выходной файл "*coord*".

19. Значения переменных cx и cy записываются в выходной файл "*coord*": $cx = sumx/count + 1$, $cy = sumy/count + 1$.

20. Вычисляется средний радиус пятна $radius = \sqrt{(count/\pi)}$ и записывается в выходной файл "*coord*".

21. Переменным i и j присваиваются значения 0.

22. Переменной $radiusmax$ присваивается значение 0.

23. Переменной *neib* (число черных соседей клетки) присваивается значение 0.

24. Программа сканирует матрицу B по рядам начиная с верхнего левого угла до тех пор, пока не будет найдена черная клетка.

25. Подсчитывается число черных соседей *neib* для этой клетки.

26. Вычисляется расстояние *radius* между граничной клеткой с координатами (kk, ll) и центром пятна (cx, cy).

27. Если вычисленное расстояние *radius* равно 0, то пятно содержит только одну клетку. Радиус таких пятен принимается равным 0.7.

28. Если *radius* больше, чем *radiusmax*, то текущее значение *radiusmax* заменяется на *radius*.

29. Пункты 23 - 28 повторяются до тех пор, пока не будет обработана матрица B.

30. Значение переменной *radiusmax* записывается в выходной файл "*coord*".

31. Значение переменной *number* увеличивается на 1.

32. Процедура повторяется начиная с пункта 4, до тех пор, пока не будет обработана вся матрица A.

Input

**.dat* - цифровой ASCII-файл.

Output

details - файл содержит число клеток в каждом пятне, координаты центров пятен, средние значения радиусов пятен и максимальные радиусы пятен.

Программа *Bubble-details.cpp* на языке C приведена в Приложении I.1.

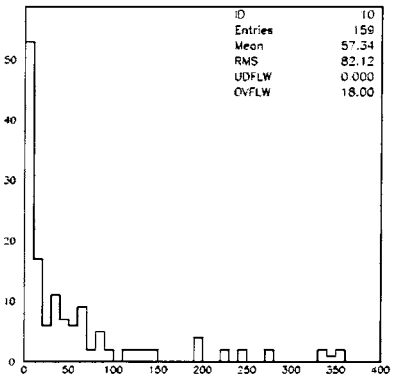

Рис. 6: Распределение числа клеток в пятнах для степени выгорания 16.2 GWd/tM

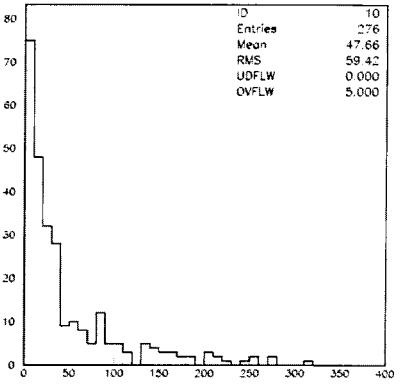

Рис. 7: Распределение числа клеток в пятнах для степени выгорания 42.6 GWd/tM

На рис. 6 и 7 приведены распределения чисел клеток в пятнах на поверхности фиксированного размера, отвечающие степеням выгорания 16.2 и 42.6, а на рис. 8

Рис. 8: Черно-белое представление микрофотографии при выгорании 16.2 GWd/tM

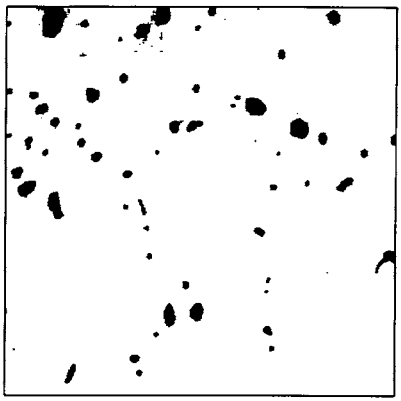

Рис. 9: Черно-белое представление микрофотографии при выгорании 42.6 GWd/tM

и 9 представлены соответствующие им черно-белые изображения. На рис. 10 и 11 приведены распределения чисел клеток для степеней выгорания 54.8 и 65.0 GWd/tM, а на рис. 12 и 13 представлены отвечающие им черно-белые изображения.

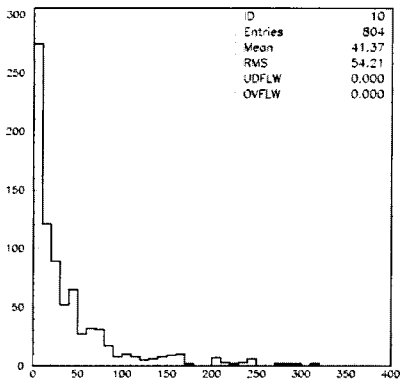

Рис. 10: Распределение числа клеток в пятнах для степени выгорания 54.8 GWd/tM

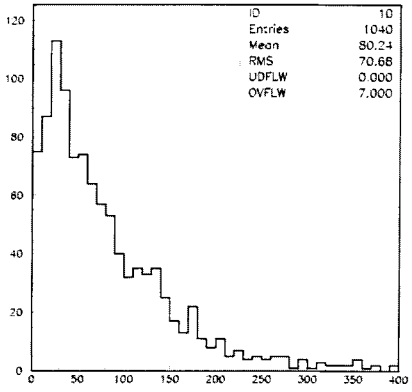

Рис. 11: Распределение числа клеток в порах для степени выгорания 65.0 GWd/tM

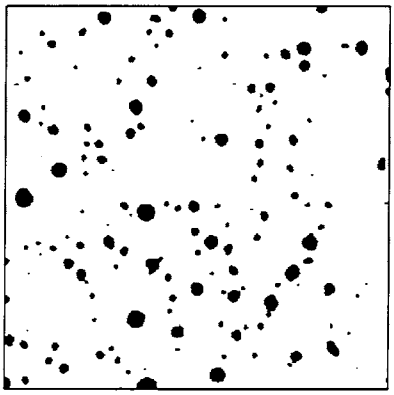

Рис. 12: Черно-белое представление микрофотографии при выгорании 54.8 GWd/tM

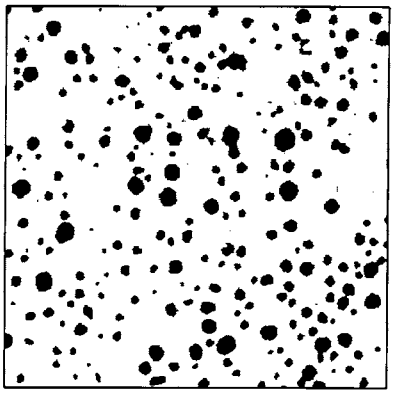

Рис. 13: Черно-белое представление микрофотографии при выгорании 65.0 GWd/tM

Эти распределения и рисунки построены с помощью пакета [11], и поэтому для нашего случая термин **Entries** означает полное число пятен, занесенных в гистограмму, а **OVFLW** - число больших пятен, размеры которых превышают 400 клеток. Представленные распределения демонстрируют характерную эволюцию поверхностного слоя топлива с течением времени (см. также [6]):

1) с ростом степени выгорания от 16.2 до 65.0 GWd/tM наблюдается систематический рост числа пятен;

2) с ростом степени выгорания от 16.2 до 54.8 GWd/tM происходит уничтожение пятен большого размера (их "рассыпание" на более мелкие пятна), а также идет образование новых пятен маленького размера;

3) при переходе от степени выгорания 54.8 к 65.0 GWd/tM наблюдается агрегация мелких пятен в пятна большего размера; при этом в распределении наблюдается характерный пик, отвечающий наиболее вероятному размеру пятна ($\sim$ 40–50); распределение на рис. 11 неплохо аппроксимируется лог-нормальным законом [12].

На рис. 14–17 представлены распределения средних радиусов пятен для степеней выгорания 16.2, 42.6, 54.8 и 65.0 GWd/tM соответственно. Эти распределения демонстрируют эволюцию радиусов пятен с ростом степени выгорания топлива, которая коррелирует с динамикой размеров пятен.

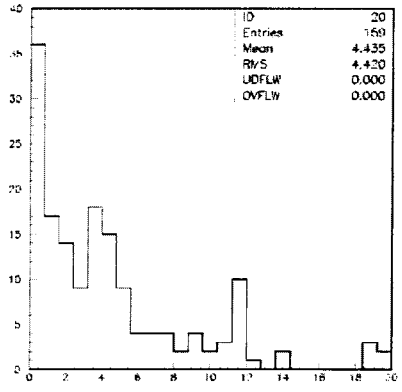

Рис. 14: Распределение средних радиусов пятен для степени выгорания 16.2 GWd/tM

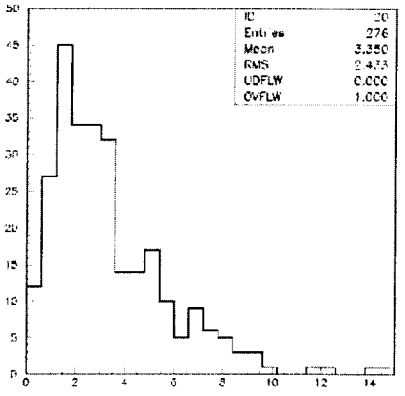

Рис. 15: Распределение средних радиусов пятен для степени выгорания 42.6 GWd/tM

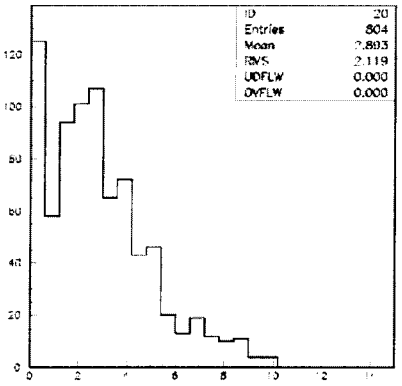

Рис. 16: Распределение средних радиусов пятен для степени выгорания 54.8 GWd/tM

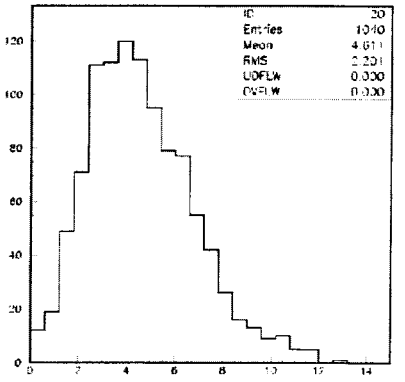

Рис. 17: Распределение средних радиусов пятен для степени выгорания 65.0 GWd/tM

4.2. Отбор пятен с заданным числом клеток

При изучении флуктуаций форм пятен необходимо различать пятна с заданным числом клеток и иметь возможность их раздельного графического

представления. Разработанный нами алгоритм позволяет отбирать пятна, содержащие заданное количество клеток и записывать их в текстовый файл по 3 образа в одном ряду. Для этих пятен также подсчитывается полная длина границы. Под длиной границы здесь понимается сумма белых клеток, которые имеют одного, двух или трех черных соседей. Основу данного алгоритма составляет описанный выше алгоритм *Bubble-details*, в качестве входной информации используется файл *bubbles* (смотри раздел **4.1**).

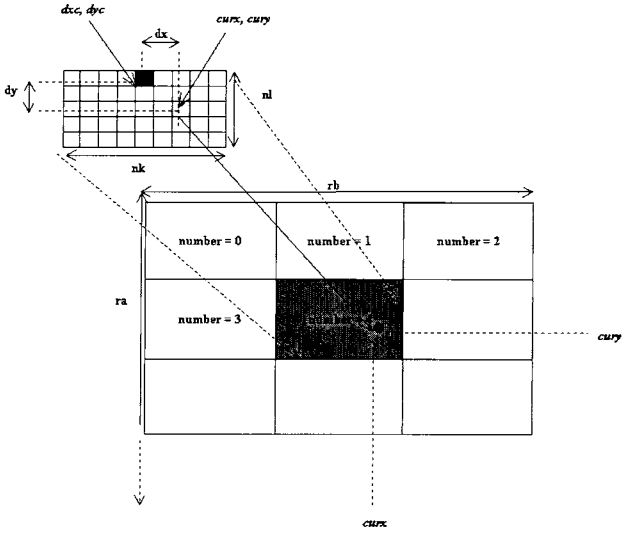

Рис. 18: Фрагмент матрицы C (внизу) и структуры блока (наверху)

Алгоритм

1. Микрофотография представляется матрицей $A(n \times m)$, где количество ячеек в каждой строке равняется m, а количество строк - n.

2. Задается значение переменной *specified*, определяющей количество клеток в пятнах, которые необходимо отобрать.

3. Определяется матрица $C(Cn \times Cm)$, которая используется для запоминания пятен с заданым числом клеток: *Cn = periodx*nk, Cm = 10*nl.* C делится на блоки заданного размера *nk*nl*, который зависит от размера отбираемых пятен. Из блоков формируется массив рядов, каждый из которых содержит по *periodx=3* образа. Начало блока имеет координаты *[dxc, dyc]*. Фрагмент матрицы C, включающий блоки 0,1,2,..., приведен на рис. 18. Структура блока, содержащего *nk*nl* клеток, показана в верхней части рис. 18.

4. Файл *bubbles* копируется в матрицу *count_in_box*.

5. Переменной *number*, используемой для подсчета всех пятен, присваивается значение 0.

6. Переменной *number2*, используемой для подсчета пятен с заданным числом клеток, присваивается значение 0.

7. Программа сканирует матрицу A по рядам, начиная с верхнего левого угла, до тех пор, пока не будет найдена черная клетка.

8. Переменной *cur* присваивается значение 1. Данная переменная показывает количество клеток в так называемом "мешке" и определяет текущую клетку.

9. Переменной *count*, которая подсчитывает количество клеток в пятне, присваивается значение 0.

10. Текущая клетка помещается в "мешок", для этого сохраняются ее координаты: в матрице Baln[*Bal*] *i*-я координата, в матрице Balm[*Bal*] *j*-я координата.

11. Переменной $firstx$ присваивается значение i, а переменной $firsty$ присваивается значение j.

12. Переменным $sumx$ и $sumy$ присваиваются значения 0.

13. Координаты (x, y) текущей клетки извлекаются из "мешка": x = Baln[*cur*], y = Balm[*cur*].

14. Если "мешок" не пустой (*cur*$\neq$ 0), то из него извлекается верхний элемент: *cur = cur - 1.*

15. Проверяется, является ли текущий элемент черным (данная проверка необходима, так как черная клетка может быть помещена в "мешок" несколько раз из-за наличия нескольких черных соседей).

16. Число клеток в пятне увеличивается на 1: *count = count + 1.* Текущая клетка в матрице A меняется на белую, для того чтобы не учитывать ее повторно.

17. Подсчитываются величины *dx, dy, ra, rb, curx, cury* (смотри рис. 18):
dx = x - firstx; dy = y - firsty;
*ra = number2/periodx; rb = number2 - ra*periodx;*
- координаты блока в матрице C, где *ra* принимает значения 0,1,2, а *rb* - это номер строки;
*curx = dy + ra*nl + dxc + 1; cury = dx + rb*nk + dyc;*
- координаты клетки пятна в матрице C.

18. Число клеток в текущем пятне сравнивается с переменной *specified.* Если они совпадают, то клетке в матрице C с координатами *[curx,cury]* присваивается значение *number2 + 1.*

19. Проверяются соседи текущей клетки. Координаты [*i,j*] всех ее черных соседей запоминаются в матрицах Baln[*Bal*] и Balm[*Bal*].

20. Величина *number* увеличивается на 1.

21. Величина *number2* увеличивается на 1, если число клеток в текущем пятне совпадает со значением $specified$.

22. Переменные i и j устанавливаются равными нулю.

23. Пункты 5 - 22 повторяются до тех пор, пока "мешок" не станет пустым. Пустой мешок означает, что подсчитаны все клетки текущего пятна.

24. Переменная *color*, которая показывает номер текущего пятна в матрице C, устанавливается равной нулю.

25. Алгоритм сканирует матрицу C до тех пор, пока он не находит первую клетку, принадлежащую пятну с номером *color*.

26. Алгоритм проверяет все клетки анализируемого пятна на наличие белых соседей. При этом подсчитывается сумма белых клеток, которые имеют одного, двух или трех черных соседей (переменная *edgelength*).

27. Переменная *edgelength* записывается в начало выходного файла.

28. Пункты 25 - 27 повторяются до тех пор, пока не будет обработана вся матрица C.

29. Переменная *color* увеличивается на 1.

30. Пункты 25 - 29 повторяются до тех пор, пока переменная *color* не станет равной *number*2.

Input

**.dat* - цифровой ASCII-файл.

bubbles - файл с числом клеток в каждом пузырьке.

Output

Move - файл, содержащий "шапку" и матрицу C.

Программа *Move.cpp* на языке C приведена в Приложении I.2.

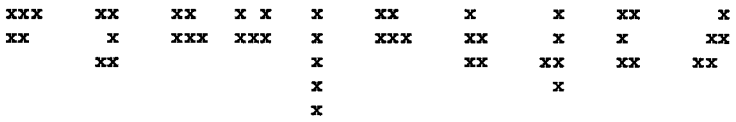

Рис. 19: Пример геометрических форм пятен, содержащих по 5 клеток

На рис. 19, в качестве примера, приведены геометрические образы пятен, содержащих по 5 клеток. Из рисунка видно, что число отобранных пятен равно 10. При этом число различных геометрических образов составляет 6, что заметно меньше числа отобраных пятен.

4.3. Изучение статистической независимости отдельных частей микрофотографий

Для исследования статистической независимости отдельных частей микрофотографий (смотри [6]) проверялось выполнение закона Гиббса для изображения. Для этого анализируемое изображение разбивалось на 400 квадратов, каждый из которых содержал 40 × 40 клеток (смотри рис. 20). В каждом квадрате

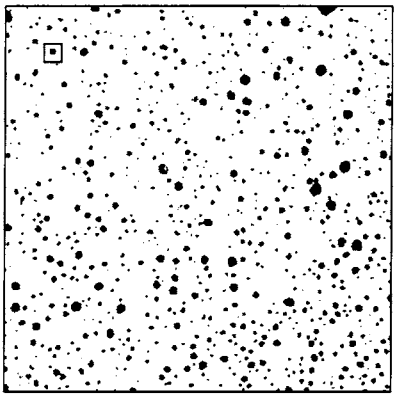

Рис. 20: Исходный образец для спепени выгорания 65.0 GWd/tM, использованный в данном исследовании; размер маленького квадрата - 40 × 40 клеток

подсчитывалось полное число черных клеток N и общее число граничных клеток L. Под граничной клеткой здесь понимается черная клетка, имеющая 1, 2 или 3 белых соседей.

Алгоритм

1. Микрография представляется матрицей $A(n \times m)$, где количество ячеек в каждой строке равняется m, а число строк - n. Матрица A делится на квадраты, каждый из которых содержит 40 × 40 клеток.

2. Переменная *boxnumber*, которая показывает номер текущего квадрата, устанавливается равной нулю.

3. Переменная *black*, которая показывает число черных клеток в квадрате, устанавливается равной нулю.

4. Переменная *edge*, которая показывает число граничных клеток в квадрате, устанавливается равной нулю (черная клетка является граничной, если она содержит 1,2 или 3 белых соседа).

5. Переменная *edgemin*, которая показывает минимально возможную границу пятна, устанавливается равной нулю.

6. Сканируется текущий квадрат матрицы A.

7. Подсчитывается величина *black* для текущего квадрата.

8. Подсчитывается величина *edge* для текущего квадрата.

9. Подсчитывается величина *edgemin* для текущего квадрата: $edgemin = \sqrt{4 \cdot \pi \cdot black}$.

10. Вычисляется величина $difference$, которая равна разности величин *edge* и *edgemin*.

11. Переменная *boxnumber* увеличивается на 1.

12. Значение переменной *black* записывается в выходной файл.

13. Значение переменной *edge* записывается в выходной файл.

14. Значение переменной $difference$ записывается в выходной файл.

15. Пункты 3 - 14 повторяются до тех пор, пока *boxnumber* не станет равным 399.

Input

**.dat* - цифровой ASCII-файл.

Output

lmin - файл, содержащий число черных клеток и число граничных клеток для каждого квадрата.

Программа Independent.cpp на языке C приведена в Приложении I.3.

На рис. 21 приведены распределения количества черных клеток N и длины границы L, а также 2-мерные (N, L)-зависимости, полученные в результате обработки изображения, представленного на рис. 20. Из этих распределений видно, что они не отвечают закону Гиббса (смотри [6])

$$w_{L,N} = A \exp(\mu N - E_{L,N})/kT,$$

где μ - это химический потенциал, а $E_{L,N}$ - энергия отдельного квадрата, которая является монотонной функцией от L.

Таким образом, на основе проведенного анализа нельзя сделать вывода о статистической независимости отдельных макроскопических частей микрофотографий. Этот вывод совпадает с результатом фрактального анализа микрофотографий, полученным в работе [13].

5. Алгоритмы моделирования процессов формирования структур

В настоящее время нет полного понимания микрохимии процессов травления. В работе [6] было показано, что процесс травления может быть достаточно реалистично воспроизведен в рамках модели клеточного автомата со специально заданными "правилами голосования". В общем случае "правила голосования"

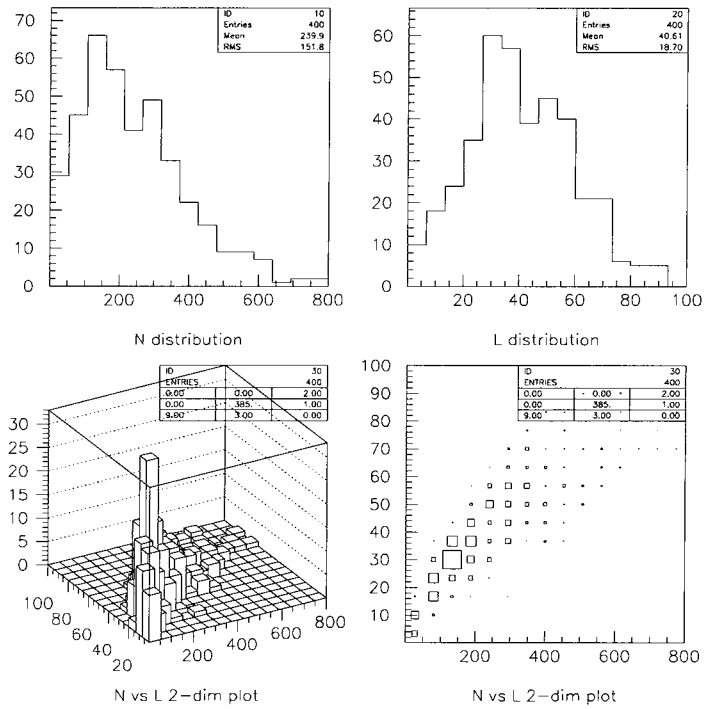

Рис. 21: Распределения числа черных клеток N и длины границы L, а также соответствующие 2-мерные (N, L)-зависимости, полученные в результате обработки изображения, представленного на рис. 20

позволяют присвоить конкретной клетке КА значение "0" или "1" в соответствии с ее похожестью на своих соседей ("популярностью" среди них) [17].

Кроме того, в [6] исследовалась динамика пятен (пор) в зависимости от температуры образца UO_2 в рамках модели Изинга [15].

Для решения этих задач были разработаны следующие алгоритмы:

- для моделирования процесса восстановления (антитравления);
- для моделирования травления;
- для описания процесса формирования пор (на основе модели Изинга).

5.1. Моделирование процесса восстановления

Для оценки эффективности применения КА для решения задач указанного типа мы представляем здесь КА, который моделирует процесс восстановления

структуры поверхности, подвергшейся процедуре травления. Этот КА трансформирует цвета в соответствии со следующим "правилом голосования":

- для белой клетки цвет остается без изменений;
- для черной клетки цвет меняется на белый, если 3 или 4 ее соседа белые, и остается без изменений в противном случае.

Интуитивная связь этого простого правила с процессом восстановления состоит в том, что при этом постепенно уничтожаются клетки, расположенные на границе черных пятен.

Алгоритм, реализующий процесс "антитравления", приведен ниже.

Алгоритм

1. Микрография представляется матрицей $A(n \times m)$, где количество ячеек в каждой строке равняется m, а количество строк - n.
2. Матрица $F(n \times m)$ используется для запоминания результирующего изображения; здесь m - это количество ячеек в каждой строке, а n - количество строк.
3. Переменной i присваивается значение 0.
4. Переменной j присваивается значение 0.
5. Элемент *A[i,j]* копируется в *F[i,j]*.
6. Элемент *F[i,j]* проверяется на цвет: если он черный, то переход на пункт 7, в противном случае — переход на пункт 10.
7. Переменной *neighbors*, которая показывает число белых соседей у текущей клетки *[i,j]*, присваивается значение 0.
8. Подсчитывается величина *neighbors*.
9. Элементу *F[i,j]* присваивается значение 1, если величина *neighbors* = 3 или 4.
10. Значение переменной j увеличивается на 1.
11. Пункты 5–10 повторяются до тех пор, пока $j < m$.
12. Значение переменной i увеличивается на 1.
13. Пункты 4–12 повторяются до тех пор, пока $i < n$.
14. Матрица F записывается в выходной файл.

Input

**.dat* - цифровой ASCII-файл.

Output

antietching - файл, содержащий выходное изображение.

Программа AntiEtching.cpp на языке C приведена в Приложении II.1.

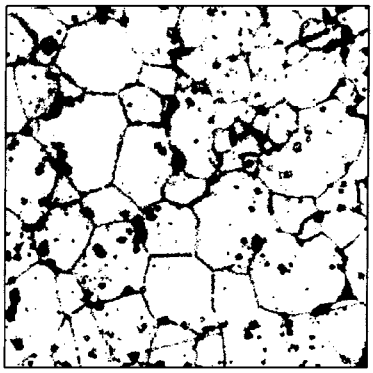

Рис. 22: Исходный протравленный образец, который использовался для моделирования процесса "антитравления"

Рисунок 22 показывает исходный протравленный образец, который использовался для моделирования процесса восстановления. Рисунки 23 и 24 демонстрируют результат работы КА после 2 и 5 итераций соответственно. Рисунок 24 может быть интерпретирован как микрофотография отполированного образца до применения процедуры травления.

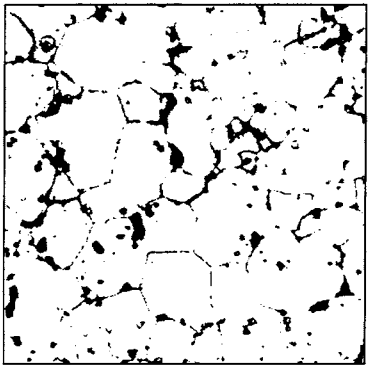

Рис. 23: Результат моделирования антитравления с помощью КА: после 2-х итераций

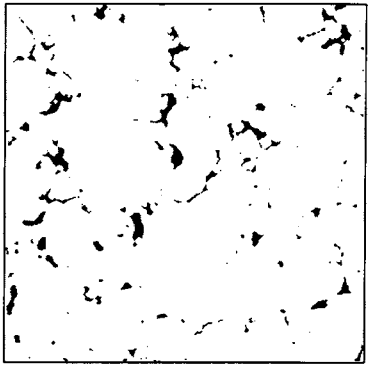

Рис. 24: Результат моделирования антитравления с помощью КА: после 5-ти итераций

5.2. Моделирование процесса травления

Процесс травления является обратным тому процессу, который был описан выше. При травлении постепенно удаляются белые клетки, прилегающие к

границам черных пятен. Соответствующий КА трансформирует цвета клеток согласно следующему "правилу голосования":

- цвет черной клетки остается без изменений,
- цвет белой клетки не меняется, если все 4 соседа белые; в случае 4, 3, 2, 1 черных соседей цвет белой клетки меняется на черный с вероятностями 1, 1, 0.25, 0.1 соответственно.

Алгоритм КА, моделирующий процесс травления, приведен ниже.

Алгоритм

1. Микрофотография представляется матрицей $A(n \times m)$, где количество ячеек в каждой строке равняется m, а количество строк - n.
2. Матрица $F(n \times m)$ используется для запоминания результирующего изображения; здесь m - это количество ячеек в каждой строке, а n - количество строк.
3. Переменной i присваивается значение 0.
4. Переменной j присваивается значение 0.
5. Элемент *A[i,j]* копируется в *F[i,j]*.
6. Элемент *F[i,j]* проверяется на цвет: если он белый, то переходим на пункт 7, в противном случае переходим на пункт 10.
7. Переменной *neighbors*, которая показывает число белых соседей у текущей клетки *[i,j]*, присваивается значение 0.
8. Подсчитывается величина *neighbors*.
9. Элементу *F[i,j]* присваивается значение 0, если величина *neighbors* равна 0, 1, 2 или 3 (с вероятностями 1, 1, 0.25, 0.1 соответственно).
10. Значение переменной j увеличивается на 1.
11. Пункты 5–10 повторяются до тех пор, пока $j < m$.
12. Значение переменной i увеличивается на 1.
13. Пункты 4–12 повторяются до тех пор, пока $i < n$.
14. Матрица F записывается в выходной файл.

Input

**.dat* - цифровой ASCII-файл.

Output

etching - файл, содержащий выходное изображение.

Программа Etching.cpp на языке C приведена в Приложении II.2.

На рис. 25 приведен исходный образец, который использовался для моделирования процесса травления. Кроме пор он содержит искусственно

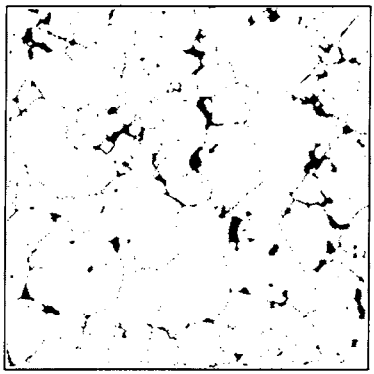

Рис. 25: Исходный образец, который использовался для моделирования процесса травления

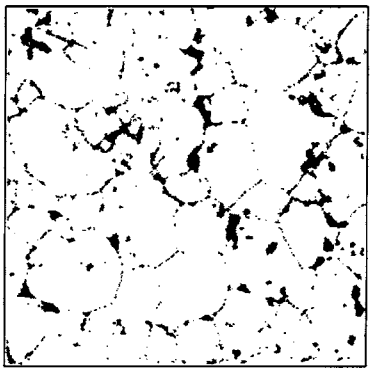

Рис. 26: Результат моделирования процесса травления с помощью КА: после 10 итераций

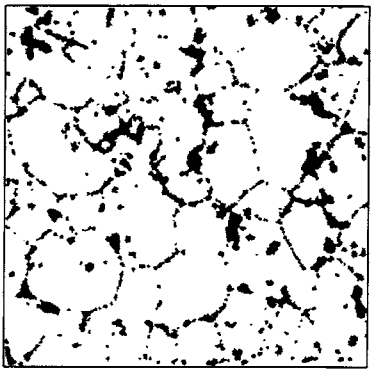

Рис. 27: Результат моделирования процесса травления с помощью КА: после 20 итераций

нанесенные точки, отвечающие первичным дефектам образующимся около трещин. Разрушения такого типа ответственны за процесс химического травления.

Исходное изображение подвергалось с помощью данного алгоритма обработке несколько раз. На рис. 26 и 27 представлены результаты применения данного КА для 10 и 20 итераций алгоритма соответственно.

5.3. Моделирование процесса формирования пузырьков на основе модели Изинга

В первом приближении смесь пора-UO_2 может быть рассмотрена как бинарная система, образованная двумя типами "молекул". Известно, что такие

системы могут быть описаны в рамках модели Изинга [15]. В этой модели переменные задаются в узлах решетки и принимают два значения, 1 и -1. Взаимодействие между узлами происходит как в КА. Для нашего случая удобно сопоставить черным и белым клеткам значения 0 и 1, при этом взаимодействуют ближайшие в смысле фон Неймана соседи.

Ниже приводится алгоритм КА, с помощью которого моделировалась эволюция изображения, вызванная изменением температуры образца [6].

Алгоритм

1. Микрофотография представляется матрицей $A(n \times m)$, где количество ячеек в каждой строке равняется m, а количество строк - n.

2. Матрица A делится на квадратные блоки размером 3×3 клеток.

3. Сканируется матрица A, по одной клетке из каждого блока. Порядок сканирования клеток во всех блоках матрицы показан на рис. 28 номерами 1, 2, ..., 9. Вначале сканируются все клетки матрицы, обозначенные номером 1, затем все клетки под номером 2 и т.д. Такой порядок позволяет для любой черной клетки не менять своего состояния дважды в процессе одного сканирования. Сканирование выполняется до тех пор, пока не найдена черная клетка.

4. Затем вычисляется вероятность $pvap = q^{4-NWH}$, где NWH - это число белых соседей, а q - заданная температура (см. детали, относящиеся к определению вероятности $pvap$ в [6].

5. С помощью датчика случайных чисел вычисляется величина $rnd1$.

6. Проверяется соотношение $rnd1 < pvap$. Если неравенство выполняется, то выполняется переход к пункту 8, в противном случае - переход к пункту 3.

7. С помощью датчика случайных чисел вычисляется величина $rnd2$.

8. Новое положение черной клетки определяется на основе величины NWH и вероятности $rnd2$.

9. Продолжается сканирование матрицы A, начиная с пункта 3, до тех пор, пока вся матрица не будет обработана.

10. Пункты 3-10 повторяются заданное число (100 в нашем случае) раз.

Input

*.*dat* - цифровой ASCII-файл.
rand - файл с набором случайных чисел.

Output

output - файл, содержащий выходное изображение.

Программа Live.pl на языке Perl приведена в Приложении II.3.

8	4	2
5	1	7
3	6	9

Рис. 28: Порядок сканирования в блоке матрицы A

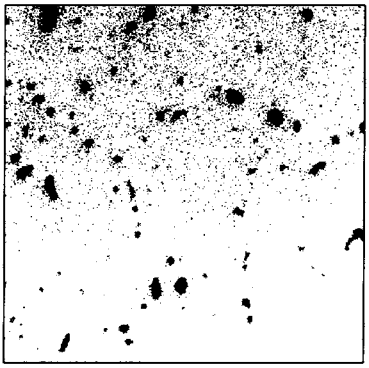

Рис. 29: Исходный образец, который использовался для моделирования формирования пор

На рис. 29 приведен образец, который использовался для моделирования процесса формирования пор.

Рисунки 30 и 31 показывают зависимость процесса фомирования пор от изменения температуры. Отчетливо видна коагуляция пор при низкой температуре (рис. 30) и уничтожение больших пор за счет испарения при высокой температуре (рис. 31).

6. Заключение

В заключение следует отметить, что, несмотря на огромную практическую значимость создания реалистичной математической модели процесса выгорания урана в ядерных реакторах современных атомных электростанций, позволяющей, в частности, прогнозировать аварийные ситуации, связанные с возможностью взрыва вещества за счет большого внутреннего давления продуктов деления, такая модель не разработана вплоть до настоящего времени.

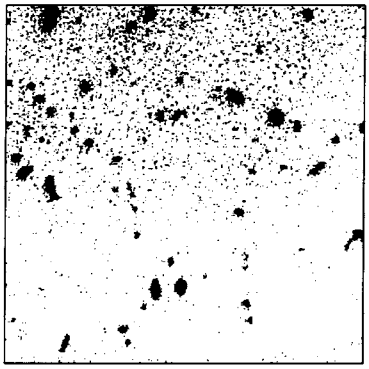

Рис. 30: Результат моделирования эволюции пор с помощью КА Изинга: P = 0.1

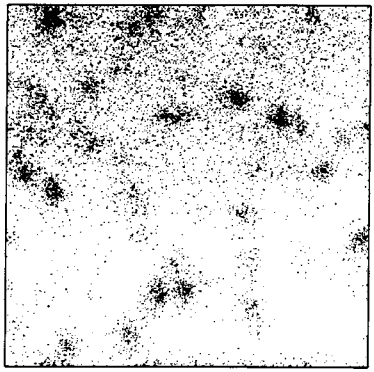

Рис. 31: Результат моделирования эволюции пор с помощью КА Изинга: P = 0.5

В данной работе предложен новый подход для моделирования динамики процесса эволюции поверхности уранового топлива на основе клеточных автоматов. Эффективность такого подхода продемонстрирована на примере клеточного автомата для формирования пятен из продуктов деления урана в процессе его выгорания (клеточный автомат Изинга), а также клеточного автомата, моделирующего процесс химического травления поверхности топлива.

С помощью клеточных автоматов были извлечены важные характеристики процесса выгорания урана, такие как эволюция размеров пятен, их количества и форм. Кроме того, исследовалась статистическая независимость отдельных частей поверхности топлива. Данное исследование показало, что нельзя сделать вывода о статистической независимости отдельных частей поверхности топлива.

Таким образом, применение клеточных автоматов в обработке изображений поверхности топлива и моделировании процессов ее эволюции продемонстрировали перспективность предложеного нами подхода. Вместе с тем, следует отметить, что нами сделаны лишь первые шаги в направлении создания реалистичной модели процесса выгорания урана.

Приложение I.1

(Bubbles-details.cpp)

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <math.h>

const An = 800; // y-dimension of A matrix
const Am = 800; // x-dimension of A matrix
const Bal = 1000; // dimension of Baln, Balm - matrixes

int A[An][Am]; // matrix contains input image
int B[An][Am]; // matrix contains output image
int Baln[Bal]; // matrix with x-coordinates of cells in the "bag"
int Balm[Bal]; // matrix with y-coordinates of cells in the "bag"

void main(void){
    int i, j, cur, x, y, number, xs, ys, kk, ll;
    long sumx, sumy, ots;
    int cx, cy; // coordinates' center of the current bubble
    long count; // number of cells in the bubble
    int neib; // count of cell neibours
    double radius, radiusmax; // bubble radius

    ifstream f_input("center800.dat"); // name of input file
    ofstream f_output("details.dat"); // name of output file

    for(i = 0; i < An; i++){
       for(j = 0; j < Am; j++){
          f_input >> A[i][j];
       }
    }

    i = 0;
    number = 0;
    while (i < An){
       j = 0;
          while (j < Am){
             if (A[i][j] == 0){

                cur = 1;
                count = 0;
                Baln[cur] = i;
                Balm[cur] = j;
                sumx = 0;
                sumy = 0;
```

```
for(kk = 0; kk < An; kk++){
   for(ll = 0; ll < Am; ll++){
      B[kk][ll] = 1;
    }
 }
 while ( cur > 0){
    x = Baln[cur];
    y = Balm[cur];
    cur = cur - 1;

    if (A[x][y] == 0){
       count++;
       A[x][y] = 8;
       B[x][y] = 0;

       sumx = sumx + x;
       sumy = sumy + y;

       //nw:
       xs = x - 1;
       if (xs < 0) xs = An - 1;
       ys = y;

       if (A[xs][ys] == 0){
          cur++;
          Baln[cur] = xs;
          Balm[cur] = ys;
       }

       //nn:
       xs = x;
       ys = y - 1;
       if (ys < 0) ys = Am - 1;
       if (A[xs][ys] == 0){
          cur++;
          Baln[cur] = xs;
          Balm[cur] = ys;
       }

       //ne:
       xs = x + 1;
       if (xs > An - 1) xs = 0;
       ys = y;
       if (A[xs][ys] == 0){
          cur++;
          Baln[cur] = xs;
          Balm[cur] = ys;
       }
```

```
      //ns:
      xs = x;
      ys = y + 1;
      if (ys > Am - 1) ys = 0;
      if (A[xs][ys] == 0){
         cur++;
         Baln[cur] = xs;
         Balm[cur] = ys;
      }
   }
}

output << number + 1 << " "<< count << " ";
cx = long(sumx/count) + 1;
cy = long(sumy/count) + 1;
output << cy <<" "<< cx <<" "<< sqrt(count/3.14) <<" ";
i = 0;
j = 0;

ots2 = 20;
radiusmax = 0;
for(kk = ots2; kk < An-ots2; kk++){
   for(ll = ots2; ll < Am-ots2; ll++){
      neib = 0;
      if (B[kk][ll] == 0){

         //nw:
         xs = kk - 1;
         if (xs < 0) xs = An - 1;
         ys = ll;
         if (B[xs][ys] == 0) neib++;

          //nn:
          xs = kk;
          ys = ll - 1;
          if (ys < 0) ys = Am - 1;
          if (B[xs][ys] == 0) neib++;

          //ne:
          xs = kk + 1;
          if (xs > An - 1) xs = 0;
          ys = ll;
          if (B[xs][ys] == 0) neib++;

          //ns:
          xs = kk;
          ys = ll + 1;
          if (ys > Am - 1) ys = 0;
          if (B[xs][ys] == 0) neib++;
```

```
                    }

                    if ((neib == 2) || (neib == 3) || (neib == 4)){
                        radius = sqrt (pow ( (abs(cx - kk) + 0.5), 2.0)
                           + pow ( (abs(cy - ll) + 0.5), 2.0));

                    if (radius == 0) radius = 0.7;
                    if (radius > radiusmax) radiusmax = radius;
                  }
                }
            }
            if (radiusmax == 0) radiusmax = 0.7; //1-cell size bubble
            output << radiusmax << "\n";
            number++;
          }
          j++;
        }
        i++;
    }
}
```

Приложение I.2

(Move.cpp)

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <math.h>
const An = 1200;
const Am = 500;
const xx = 1000;
const inbox = 986; // max in box
const ttt = 19; //specified number of cells in bubble

int A[An][Am];
int Baln[xx];
int Balm[xx];

void main(void){
    int i, j, cur, x, y, number, number2, xs, ys;
    long count;

    ifstream f_input("54x12x5.dat");
    ifstream f_input2("bal54");
    ofstream Asmall("19");

    // for move
    int dx, dy; // coord in BOX
    const dxc = 0; // center x in BOX
```

```
const dyc = 10; // center y in BOX
const nl = 14, nk = 19; // size of BOX nl, nk
int ra, rb;
int curx, cury; // current pix for NEW file
int firstx, firsty;
const periodx = 3;
const Cn = periodx*nk;
const Cm = 10*nl;
int C[Cm][Cn];
int count_in_box[inbox];
int temp;

for(i = 0; i < Am; i++){
   for(j = 0; j < An; j++){
      f_input >> A[j][i];
   }
}

for(i = 0; i < Cm; i++){
   for(j = 0; j < Cn; j++){
      C[i][j] = 0;
   }
}

for (i = 0; i < inbox;i++){
  f_input2 >> count_in_box[i]; //read ''count of cells'' for each bubble
  f_input2 >> temp; f_input2 >> temp; f_input2 >> temp; f_input2 >> temp;
}

i = 0;
number = 0;
number2 = 0;
while (i < Am){
   j = 0;
   while (j < An){
      if ((A[j][i] == 0)&&(number < 2000)){

         cur = 1;
         count = 0;
         Baln[cur] = j;
         Balm[cur] = i;
         firstx = j;
         firsty = i;
         while ( cur > 0){
            x = Baln[cur];
            y = Balm[cur];
            cur = cur - 1;
            if (A[x][y] == 0){
               count++;
```

```
A[x][y] = 7;
dx = x - firstx;
dy = y - firsty;
ra = int(number2/periodx);
rb = number2 - ra*periodx;

curx = dy + ra*nl + dxc + 1;
cury = dx + rb*nk + dyc;

if (count_in_box[number] == ttt) C[curx][cury] = number2 + 1;
//nw:
xs = x - 1;
if (xs < 0) xs = An - 1;
ys = y;

if (A[xs][ys] == 0){
   cur++;
   Baln[cur] = xs;
   Balm[cur] = ys;
}

//nn:
xs = x;
ys = y - 1;
if (ys < 0) ys = Am - 1;
if (A[xs][ys] == 0){
   cur++;
   Baln[cur] = xs;
   Balm[cur] = ys;
}

//ne:
xs = x + 1;
if (xs > An - 1) xs = 0;
ys = y;
if (A[xs][ys] == 0){
   cur++;
   Baln[cur] = xs;
   Balm[cur] = ys;
}

//ns:
xs = x;
ys = y + 1;
if (ys > Am - 1) ys = 0;
if (A[xs][ys] == 0){
   cur++;
   Baln[cur] = xs;
   Balm[cur] = ys;
```

```
            }
          }
        }
        if (count_in_box[number] == ttt) number2++;
        number++;
        i = 0;
        j = 0;
      }
      j++;
    }
    i++;
}
////// count edgelength in C file for each box
int color, edgelength, n1, n2, n3, n4, n, m, neib;
for(color = 0; color < number2; color++){
   edgelength = 0;
   for(n = 0; n < Cm; n++){
      for(m = 0; m < Cn; m++){
         if (C[n][m] == (color+1)) {

            n1 = 0; n2 = 0; n3 = 0; n4 = 0;

            // n1:
            x = n - 1; y = m;
            if (C[x][y] == 0) n1 = 1;

            // n2:
            x = n; y = m - 1;
            if (C[x][y] == 0) n2 = 1;

            // n3:
            x = n + 1; y = m;
            if (C[x][y] == 0) n3 = 1;

            // n4:
            x = n; y = m + 1;
            if (C[x][y] == 0) n4 = 1;

            neib = n1+n2+n3+n4;
            if( (neib == 1)||(neib == 2)||(neib == 3) )
               edgelength = edgelength + neib;
         };
      }
   }
   Asmall << color+1 << " " << edgelength << "\n";
}

////// print C file - output file
for(i = 0; i < Cm; i++){
```

```
        for(j = 0; j < Cn; j++){
           if (C[i][j] != 0) C[i][j] = 7;
           Asmall << C[i][j] << "";
        }
        Asmall << "\n";
     }
}
```

Приложение I.3

(Independent.cpp)

```
#include <math.h>
#include <fstream.h>

const l = 800, k = 800;
const num = 40;
const lnum = l/num;
unsigned short A[l][k];
int black, edge, edgemin;
unsigned short n1, n2, n3, n4, neb; //neighbors

void main(void)
{
   int n, m, box_number_x, box_number_y, i, j, x, y, difference;

   ifstream  f_input("pic65.dat");
   ofstream f_output("indep");

   for(n = 0; n < l; n++){
      for(m = 0; m < k; m++){
         f_input >> A[n][m];
      }
   }

   for(box_number_x = 0;box_number_x < lnum;box_number_x++){
      for(box_number_y = 0;box_number_y < lnum;box_number_y++){
         black = 0;
         edge = 0;
         edgemin = 0;
         for(i=0;i<num;i++){
            for(j=0;j<num;j++){

               // n1:
               x = i+box_number_x*lnum - 1;
               if (x < 0) x = l - 1;
               y = j+box_number_y*lnum;
               n1 = A[x][y];
```

```
                // n2:
                x = i+box_number_x*lnum;
                y = j+box_number_y*lnum - 1;
                if (y < 0) y = k - 1;
                n2 = A[x][y];

                // n3:
                x = i+box_number_x*lnum + 1;
                if (x == l) x = 0;
                y = j+box_number_y*lnum;
                n3 = A[x][y];

                // n4:
                x = i+box_number_x*lnum;
                y = j+box_number_y*lnum + 1;
                if (y == k) y = 0;
                n4 = A[x][y];

                neb = n1 + n2 + n3 + n4;
                if (A[i+box_number_x*lnum][j+box_number_y*lnum] == 0) {
                   if ((neb == 1)||(neb == 2)||(neb == 3)) edge++;
                }
                if (A[i+box_number_x*lnum][j+box_number_y*lnum] == 0) black++;
            }
        }
        f_output << box_number_y + box_number_x*lnum << " "
           << black << " "<< edge << " ";
        edgemin = sqrt(4*3.1415937*black);
        difference = edge - edgemin;
        f_output << difference << "\n";
      }
   }
}
```

Приложение II.1

(AntiEtching.cpp)

```
#include <math.h>
#include <fstream.h>

const l = 801, k = 801;
unsigned short pic[l][k];

void main(void)
{

   int n, m;
   int x, y;
```

```
    unsigned short n1, n2, n3, n4, neighbors;
    unsigned short cur;

    ifstream f_in("16e.dat");
    ofstream f_out("antietching");

    for(n = 0; n < l; n++){
       for(m = 0; m < k; m++){
          f_in >> A[n][m];
       }
    }

    for(n=0;n<l;n++){
       for(m=0;m<k;m++){
          neb = 0;

          x = n - 1;
          if (x < 0) x = l - 1;
          y = m;
          n1 = A[x][y];

          // n2:
          x = n;
          y = m - 1;
          if (y < 0) y = k - 1;
          n2 = A[x][y];

          // n3:
          x = n + 1;
          if (x == l) x = 0;
          y = m;
          n3 = A[x][y];

          // n4:
          x = n;
          y = m + 1;
          if (y == k) y = 0;
          n4 = A[x][y];

          neighbors = n1 + n2 + n3 + n4;
          cur = A[n][m];

          if ((neighbors == 4)||(neighbors == 3)) cur = 1;
          f_out << cur << " ";
       }
       f_out << "\n";
    }
}
```

Приложение II.2

(Etching.cpp)

```
#include <math.h>
#include <fstream.h>

#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
#include "fstream.h"
#include <math.h>

float ran1(long *idum)
{
   int j;
   long k;
   static long iy=0;
   static long iv[NTAB];
   float temp;

   if (*idum <= 0 || !iy) {
      if (-(*idum) < 1) *idum=1;
      else *idum = -(*idum);
      for (j=NTAB+7;j>=0;j--) {
         k=(*idum)/IQ;
         *idum=IA*(*idum-k*IQ)-IR*k;
         if (*idum < 0) *idum += IM;
         if (j < NTAB) iv[j] = *idum;
       }
       iy=iv[0];
   }
   k=(*idum)/IQ;
   *idum=IA*(*idum-k*IQ)-IR*k;
   if (*idum < 0) *idum += IM;
   j=iy/NDIV;
   iy=iv[j];
   iv[j] = *idum;
   if ((temp=AM*iy) > RNMX) return RNMX;
   else return temp;
}
#undef IA
```

```
#undef IM
#undef AM
#undef IQ
#undef IR
#undef NTAB
#undef NDIV
#undef EPS
#undef RNMX

const l = 801, k = 801;
unsigned short A[l][k];

void main(void)
{
   int n, m, x, y;
   unsigned short n1, n2, n3, n4, neighbors;
   unsigned short cur;
   long idu;
   idu = -1;

   ifstream  f_in("rim.dat");
   ofstream f_out("etching");

   for(n = 0; n < l; n++){
      for(m = 0; m < k; m++){
         f_in >> A[n][m];
      }
   }

   for(n=0;n<l;n++){
      for(m=0;m<k;m++){
         neighbors = 0;
         cur = A[n][m];
         if (cur == 1){

            //n1;
            x = n - 1;
            if (x < 0) x = l - 1;
            y = m;
            n1 = A[x][y];

            // n2:
            x = n;
            y = m - 1;
            if (y < 0) y = k - 1;
            n2 = A[x][y];

            // n3:
            x = n + 1;
```

```
            if (x == l) x = 0;
            y = m;
            n3 = A[x][y];

            // n4:
            x = n;
            y = m + 1;
            if (y == k) y = 0;
            n4 = A[x][y];

            neighbors = n1 + n2 + n3 + n4;

            if (neighbors == 0) cur = 0;
            if (neighbors == 1) cur = 0;
            if (neighbors == 2){
               if (ran1(&idu) < 0.250) cur = 0;
            }
            if (neighbors == 3){
               if (ran1(&idu) < 0.1) cur = 0;
            }
         }
         f_out << cur << " ";
      }
      f_out << "\n";
   }
}
```

Приложение II.3

(Live.pl)

```
#!/usr/bin/perl
$q = 0; $rnd = 0; $NWH = 0; @matrix = ();

&init ();

open (RND, "<$file_rand") or die "rand died $!";
&input ();
for (1..$more) {&go ();
print $_, "\n";
}
close RND;
&output ();

sub NEWP {
my ($x, $y) = (shift, shift);
$matrix [$x][$y] = 1;

$rnd = <RND>; chomp;
if ($NWH == 1) {$matrix [$NW_x [0]][$NW_y [0]] = 0}
```

```
elsif ($NWH == 2) {
   if ($rnd < 0.5) {$matrix [$NW_x [0]][$NW_y [0]] = 0}
   else {$matrix [$NW_x [1]][$NW_y [1]] = 0}
}
elsif ($NWH == 3) {
   if ($rnd < 0.33333) {$matrix [$NW_x [0]][$NW_y [0]] = 0}
   elsif ($rnd < 0.66666) {$matrix [$NW_x [1]][$NW_y [1]] = 0}
   else {$matrix [$NW_x [2]][$NW_y [2]] = 0}
}
elsif ($NWH == 4) {
   if ($rnd < 0.5) {
      if ($rnd < 0.25) {$matrix [$NW_x [0]][$NW_y [0]] = 0}
      else {$matrix [$NW_x [1]][$NW_y [1]] = 0}
      else {
         if ($rnd < 0.75) {$matrix [$NW_x [2]][$NW_y [2]] = 0}
         else {$matrix [$NW_x [3]][$NW_y [3]] = 0;}
      }
   }
}
sub calc_NWH {
my ($x, $y) = (shift, shift);
my $count = 0;
my ($dx, $dy) = (undef, undef);
@NW_x = ();
@NW_y = ();

#1
$dx = $x - 1;
$dy = $y;
$dx = $n - 1 if $dx < 0;
if ($matrix [$dx][$dy]) {
   $count++;
   push @NW_x, $dx;
   push @NW_y, $dy
}

#2
$dx = $x;
$dy = $y + 1;
$dy = 0 if $dy > $m - 1;
if ($matrix [$dx][$dy]) {
   $count++;
   push @NW_x, $dx;
   push @NW_y, $dy
}

#3
$dx = $x + 1;
$dy = $y;
```

```
$dx = 0 if $dx > $n - 1;
if ($matrix [$dx][$dy]) {
   $count++;
   push @NW_x, $dx;
   push @NW_y, $dy
}

#4
$dx = $x;
$dy = $y - 1;
$dy = $m - 1 if $dy < 0;
if ($matrix [$dx][$dy]) {
   $count++;
   push @NW_x, $dx;
   push @NW_y, $dy
}

return $count
}

sub go {

for $vector (0..$n_block * $m_block - 1) {
for $y (0..$m / $m_block - 1) {
   for $x (0..$n / $n_block - 1) {
      my $x_cell = $x * $n_block + $x_vector [$vector];
      my $y_cell = $y * $m_block + $y_vector [$vector];

      next if $matrix [$x_cell][$y_cell];

      #THIS cell == 0!
      $NWH = &calc_NWH ($x_cell, $y_cell);
      next if $NWH == 0;
      &NEWP ($x_cell, $y_cell), next if $NWH == 4;

      $PVAP = $q ** (4 - $NWH);
      $rnd = <RND>; chomp $rnd;
      &NEWP ($x_cell, $y_cell) if $rnd < $PVAP;
   }
}

}
}
sub init {
   $more = 100;

   $file_input = 'input';
   $file_output= 'output';
   $file_rand = 'rand';
```

```
    $n = 399;                # x
    $m = 399;                # y

    $n_block = 3;
    $m_block = 3;
    $q = 0.5;
    @x_vector = (1, 2, 0, 1, 0, 1, 2, 0, 2);
    @y_vector = (1, 0, 2, 0, 1, 2, 1, 0, 2);
}

sub input {
    open (IN, "<$file_input") or die "no input file... (read) $!";
    for $y (0..$m - 1) {
        my $line = <IN>;
        my @tmp = split (' ', $line);
        for $x (0.. $n - 1) {$matrix [$x][$y] = $tmp [$x]}
    }
}
sub output {
    open (OUT, ">$file_output") or die "no output file!!! (write) $!";
    for $y (0..$m - 1) {
        for $x (0..$n - 1) {print OUT " ", $matrix [$x][$y], " "}
    print OUT "\n";
    }
}
```

Литература

[1] Hj.Matzke, A.Turos, G.Linker: *Polygonization of single crystals of the fluorite-type oxide UO_2 due to high dose ion implantation*, Nucl. Instr. and Meth., 1994, vol. B91, p. 294-300.

[2] Hj. Matzke, J. Spino: *Formation of the rim structure in high burn-up fuel*, Jour. of Nucl. Materials, 1997, vol. 248, p. 170-179.

[3] Hj. Matzke, M. Kinoshita: *Polygonization and high burn-up structure in nuclear fuels*, Jour. of Nucl. Materials, 1997, vol. 247, p. 108-115.

[4] M. Kinoshita, T. Sonoda, S. Kitajima, A. Sasahara, E. Kolstad, Hj. Matzke, V.V. Rondinella, A.D. Stalios, C.T. Walker, I.L.F. Ray, M. Steindlin, D. Halton, C. Ronchi: *High burn-up rim project. Irradiation and examination to investigate rim-structured fuel*, Proc. Int. Conf. on LWR Fuel Perfomance, Amer. Nucl. Soc., Park City, April 9-14, 2000, p.590

[5] Chan Bock Lee, Youn Ho Jung: *An attempt to explain the high burn-up structure formation mechanism in UO_2 fuel*, Jour. of Nucl. Materials, 2000, vol. 279, p. 207.

[6] E.P.Akishina, I.Antoniou, V.V.Ivanov, B.F.Kostenko and A.D.Stalios: *Cellular Automata Study of High Burn-Up Structures*, "Chaos, Solitons & Fractals", 18 (2003) 1111-1128.

[7] S.Wolfram (ed.): *Theory and Aplications of Cellular Automata*, World Scientific, 1986.

[8] T.Toffoli, N.Margolus, *Cellular-Automaton Machines: A New Environment for Modeling*, MIT Press, Cambridge, Massachussetts, 1987.

[9] Site: *http://www.mirwoj.opus.chelm.pl.*

[10] E.P. Akishina. Program CONVERT (не опубликована).

[11] R. Brun, O. Couet, C. Vandoni and P. Zanarini: *PAW - Physics Analysis Workstation*, CERN Program Library Q121, 1989.

[12] W.T. Eadie, D. Dryard, F.E. James, M. Roos and B. Sadoulet: *Statistical Methods in Experimental Physics*, North-Holland Pub.Comp., Amsterdam-London, 1971.

[13] I.Antoniou, V.V.Ivanov, B.F.Kostenko, J.Spino and A.D.Stalios: *Fractal Analysis of high burn-up structures in UO_2*, "Chaos, Solitons & Fractals" 19 (2004) 731-737.

[14] E.P.Akishina, I.Antoniou, V.V.Ivanov, B.F.Kostenko and A.D.Stalios: *Cellular Automata Modeling of High Burn-Up Structures in UO_2*. In: V Int. Congress on Mathematical Modeling, September 30-October 6, 2002, Book of abstracts, Vol. **I**, p. 137, Dubna, Moscow region, Russia, 2002;
"Computational Methods in Sciences and Engineering" (in press).

[15] R.P.Feynman, Statistical Mechanics. A Set of Lectures, Benjamin, Inc., Massachusetts, 1972.

[16] I.Antoniou, V.V.Ivanov, B.F.Kostenko, J.Spino and A.D.Stalios: *Fractal Analysis of High Burn-Up Structures in UO_2*. In: *"Studies of the RIM Effect in UO_2"*, Eds. I.Antoniou, V.Ivanov, SOLVAY preprint 01-3, Brussels, 2001.

[17] Т. Лиггетт: *Марковские процессы с локальными взаимодействиями*, Москва, Мир, 1989, с. 268-305.

Получено 30 сентября 2003 г.

Акишина Е. П., Иванов В. В., Костенко Б. Ф.
Р11-2003-184

Изучение структур сильного выгорания двуокиси урана
с помощью клеточных автоматов:
алгоритмы и программы

Предложен новый метод изучения пространственных структур, образующихся в результате выгорания двуокиси урана в ядерных реакторах современных атомных электростанций. В основе метода лежит представление изображений указанных структур в виде рабочего поля клеточного автомата (КА). Это позволило, во-первых, осуществить извлечение важных количественных характеристик структур непосредственно с микрофотографий поверхности уранового топлива. Во-вторых, выяснилось, что КА позволяют легко сформулировать динамику процесса эволюции изучаемых структур в терминах элементов микрофотографий, таких как пятна, границы пятен, трещины и др. Установлена связь этой динамики с некоторыми точно решаемыми моделями теории клеточных автоматов, в частности — моделью Изинга и моделью голосования. Дано детальное описание некоторых алгоритмов КА, позволяющих как производить обработку изображений поверхности топлива, так и моделировать процессы ее эволюции в результате выгорания или химического травления.

Работа выполнена в Лаборатории информационных технологий ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2003

Перевод авторов

Akishina E. P., Ivanov V. V., Kostenko B. F.
P11-2003-184

Investigation of High Burnup Structures
in Uranium Dioxide Applying Cellular Automata:
Algorithms and Codes

A new method of research in spatial structures that result from uranium dioxide burning in nuclear reactors of modern atomic plants is suggested. The method is based on presentation of images of the mentioned structures in the form of the working field of a cellular automaton (CA). First, it has allowed one to extract some important quantitative characteristics of the structures directly from the micrographs of the uranium fuel surface. Secondly, the CA has been found out to allow one to formulate easily the dynamics of the evolution of the studied structures in terms of such micrograph elements as spots, spots' boundaries, cracks, etc. Relation has been found between the dynamics and some exactly solvable models of the theory of cellular automata, in particular, the Ising model and the vote model. This investigation gives a detailed description of some CA algorithms which allow one to perform the fuel surface image processing and to model its evolution caused by burnup or chemical etching.

The investigation has been performed at the Laboratory of Information Technologies, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna, 2003

Редактор *М. И. Зарубина*
Макет *Н. А. Киселевой*

Подписано в печать 24.10.2003.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 2,5. Уч.-изд. л. 2,9. Тираж 310 экз. Заказ № 54148.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.
E-mail: publish@pds.jinr.ru
www.jinr.ru/publish/