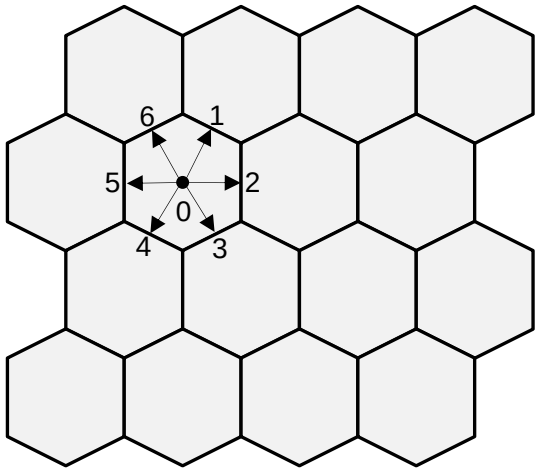
Реализация клеточно-автоматной модели FHP на графическом ускорителе при помощи фреймворка TensorFlow

Выполнила: Матолыгина Наталия Андреевна, аспирант ТГУ

Структура модели

FHP-модель – двумерная модель потока с 6 соседями

- Фаза сдвига: каждая частица перемещается на 1 клетку в направлении вектора скорости
- Фаза столкновения: изменение состояния клеток согласно некоторым правилам столкновения

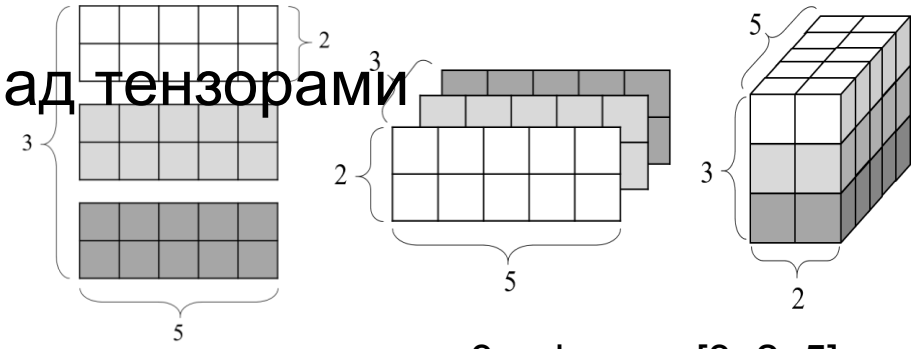


TensorFlow

- Клеточный автомат – тензор
- Сдвиг и столкновение – операции над тензорами

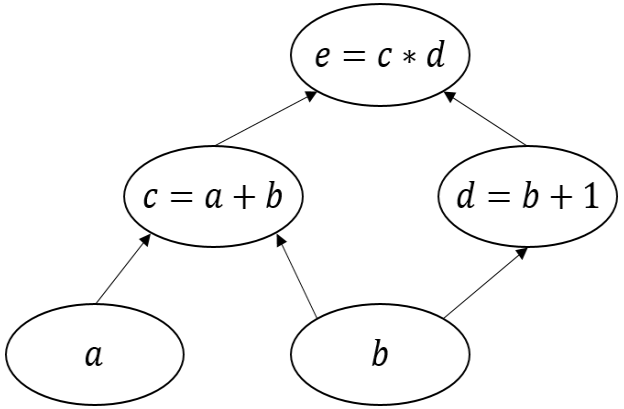
Тензор (многомерная матрица) – основная структура данных фреймворка TensorFlow

Параметры тензора: ранг, форма, размер, тип



тензор ранга 3 и формы [3, 2, 5]

Граф вычислений – это граф потоков данных, в котором математические операции представлены в виде узлов, а данные – в виде ребер между этими узлами

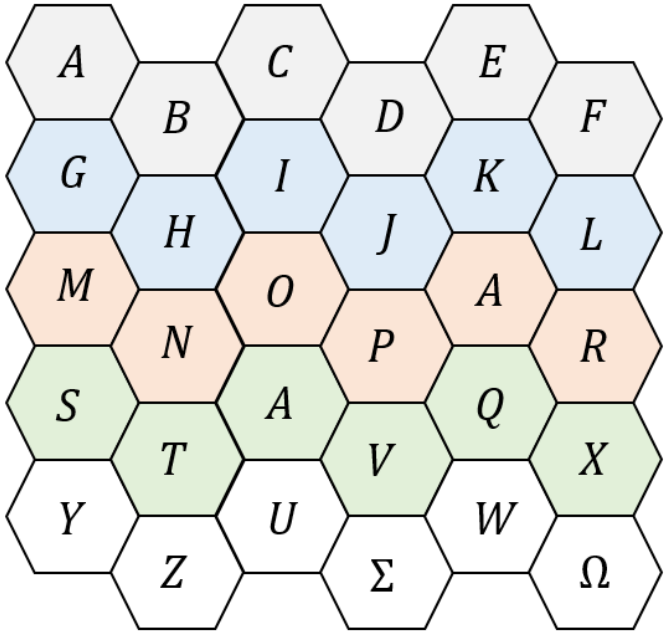


Свойства:

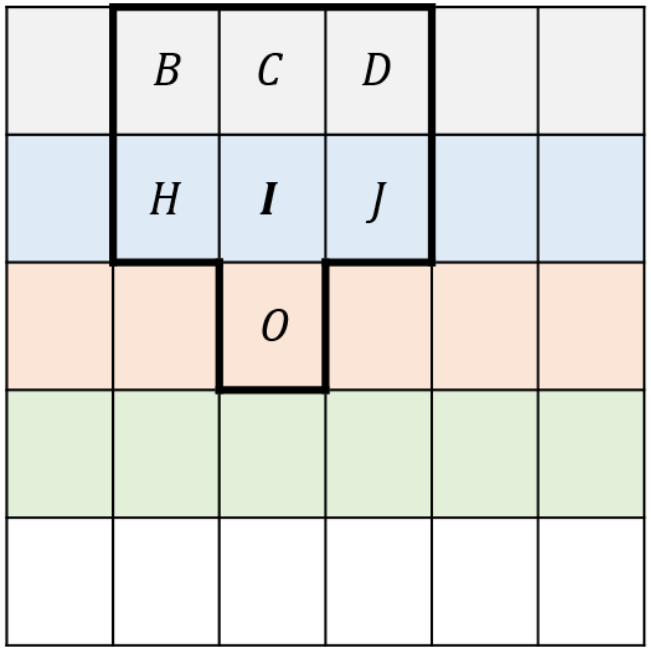
- Листовые вершины – тензоры
- Тензор не может быть неоконченным узлом
- Иерархический порядок операций
- Обход графа в обратном направлении формирует подвыражения, которые объединяются в выражение
- При обходе графа в прямом направлении, встречаемая вершина становится зависимостью для следующей
- Работа в узлах одного уровня не зависит друг от друга

Реализация

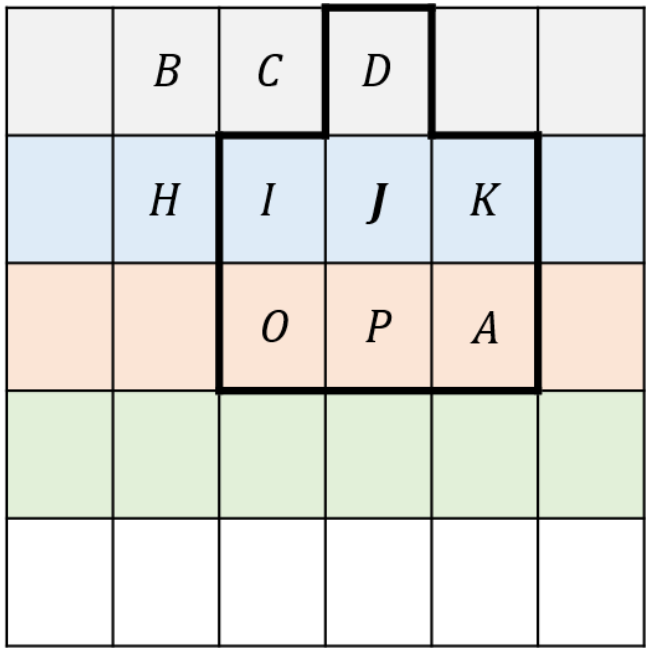
1. Отображение шестиугольной сетки на прямоугольную



Тип соседства для чётных столбцов



Тип соседства для нечётных столбцов



Реализация сдвига

- Фаза сдвига – пользовательская операция на уровне манипуляций с основными структурами данных (массивами)
- Язык - CUDA

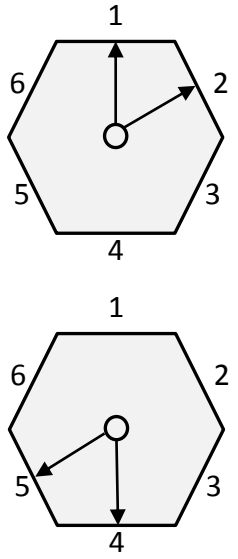
```
Init Tensor
<tf.Variable 'Variable:0' shape=(4, 4) dtype=int32, numpy=
array([[25, 17,  0,  0],
       [17,  4,  6, 65],
       [ 0,  2,  8, 98],
       [ 0,  0,  0,  0]])>
Result Tensor
tf.Tensor(
[[ 1  9  2  4]
 [ 1 18 68  3]
 [16  0 64  8]
 [ 0  0 32  0]], shape=(4, 4), dtype=int32)
```

```
// частица от соседа 0
out[i] |= (in[i] & mask_0);

// частица от соседа 1 прилетает в направлении 4
out[i] |= ((i > ColNum) ? (in[i - ColNum] & mask_4) : 0);

// частица от соседа 2 прилетает в направлении 5
if ((j != ColNum - 1) && (j_mod_2 || i >= ColNum)) {
  out[i] |= (in[i + 1 - (j_mod_2 ? 0 : ColNum)] & mask_5);
}

// частица от соседа 3 прилетает в направлении 6
if ((j != ColNum - 1) && (!j_mod_2 || (i <= numElements - ColNum))) {
  out[i] |=( in[i + 1 + (j_mod_2 ? ColNum : 0)] & mask_6);
}

// частица от соседа 4 прилетает в направлении 1
out[i] |= ((i < numElements - ColNum) ? (in[i + ColNum] & mask_1) : 0);

// частица от соседа 5 прилетает в направлении 2
if ((j != 0) && (!j_mod_2 || (i <= numElements - ColNum))) {
  out[i] |= (in[i - 1 + (j_mod_2 ? ColNum : 0)] & mask_2);
}

// частица от соседа 6 прилетает в направлении 3
if ((j != 0) && (j_mod_2 || i >= ColNum)) {
  out[i] |= (in[i - 1 - (j_mod_2 ? 0 : ColNum)] & mask_3);
}
```

Реализация столкновения (1)

1. Обработка клеток-стенок

```
//// Проверка на стенку. Если 7-ой бит единичный, то меняем направление на противоположное
if (out[i] & 128) {
    // Если есть частица в 1 направлении и нет в 4, то сбрасываем бит в 1 и устанавливаем в 4
  if ((out[i] & mask_1 != 0) && (out[i] & mask_4 == 0)) {
    out[i] ^= 2;
    out[i] |= 16;
  }
    // Если есть частица в 4 направлении и нет в 1, то сбрасываем бит в 4 и устанавливаем в 1
    else if ((out[i] & mask_4 != 0) && (out[i] & mask_1 == 0)) {
        out[i] ^= 16;
        out[i] |= 2;
    }

    // Если есть частица в 2 направлении и нет в 5, то сбрасываем бит в 2 и устанавливаем в 5
    else if ((out[i] & mask_2 != 0) && (out[i] & mask_5 == 0)) {
        out[i] ^= 4;
        out[i] |= 32;
    }
    // Если есть частица в 5 направлении и нет в 2, то сбрасываем в 5 и устанавливаем в 2
    else if ((out[i] & mask_5 != 0) && (out[i] & mask_2 == 0)) {
        out[i] ^= 32;
        out[i] |= 4;
    }
```

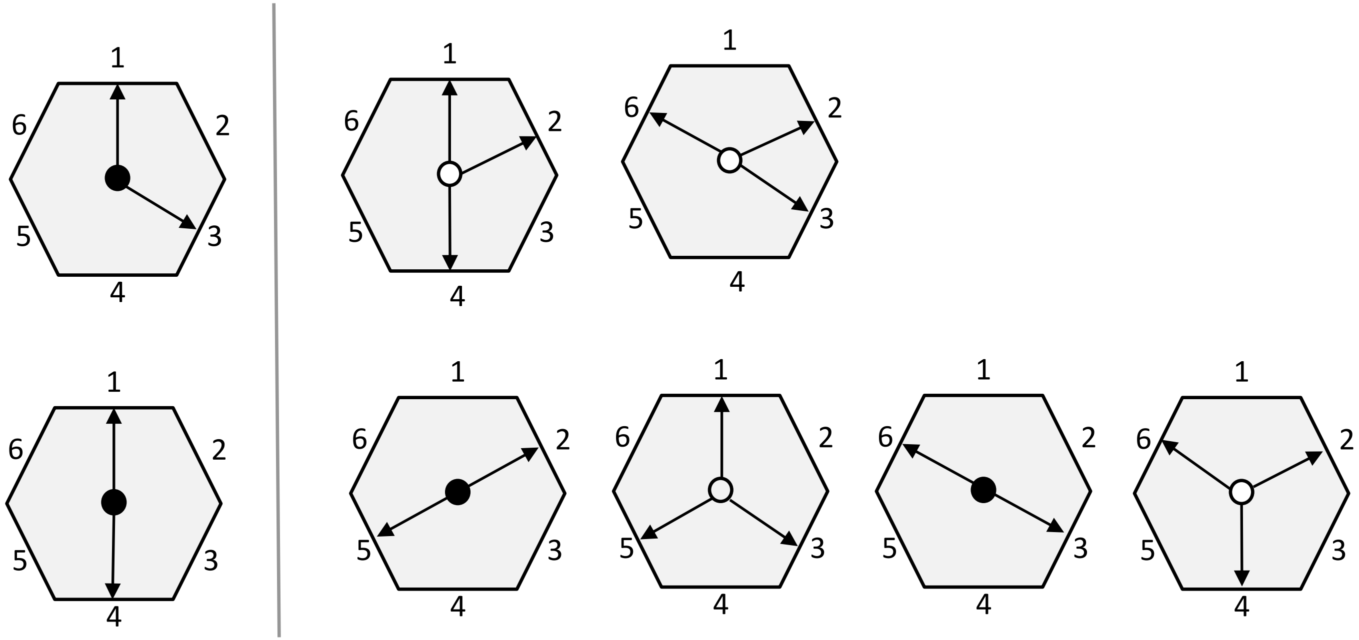


Реализация столкновения (2)

2. Обработка рабочих клеток

```
struct transition {
    short tNumber;  // Кол-во переходов
    short trans[MAX_TRANSITION];  // Следующие состояния
};
```

```
struct transition transitionTable[128] = {
  // из состояния 0 0000000
  {1, {0}},
  // из состояния 1 0000001
  {1, {1}},
  // из состояния 11 0001011
  {2, {22, 76}},
  // из состояния 19  0010011
  {4, {37, 42, 73, 84}},
```



Реализация столкновения (3)

Одно состояние

```
t = transitionTable[out[i]]; // Берём из структуры out[i]-ый элемент
//Если возможный переход один
if (t.tNumber == 1) {
  out[i] = t.trans[0];
}
```

Два состояния

```
else {
  // Если возможных вариантов 2
  if (t.tNumber == 2) {
    if (rand[i] < 0.5) {
      out[i] = t.trans[0];
    } else {
      out[i] = t.trans[1];
    }
  }
```

Четыре состояния

```
// Иначе если возможных вариантов 4
else if (t.tNumber == 4) {
  if (rand[i] < 0.25) {
    out[i] = t.trans[0];
  } else if (rand[i] < 0.5) {
    out[i] = t.trans[1];
  } else if (rand[i] < 0.75) {
    out[i] = t.trans[2];
  } else if (rand[i] < 1.0) {
    out[i] = t.trans[3];
  }
```

Сборка библиотеки

Регистрация пользовательской операции

```
REGISTER_OP("FHP")
    .Attr("T: numbertype")
    .Input("input: T")
    .Input("input_rand: float32")
    .Output("output: T")
    .SetShapeFn([](::tensorflow::shape_inference::InferenceContext* c) {
      c->set_output(0, c->input(0));
      return Status::OK();
    });
```

Необходимые файлы:

FHP_model.cc
FHP_model.cu.cc
FHP_model.h
BUILD

BUILD-файл

```
tf_custom_op_library(
    name = "FHP_model.so",
    srcs = ["FHP_model.cc"],
    gpu_srcs = ["FHP_model.cu.cc"],
)
```

Сборка библиотеки

Инструмент сборки для TensorFlow

bazel build --config=opt --config=cuda //tensorflow/core/user_ops:FHP.so

```
import tensorflow as tf
from tensorflow.python.ops import bitwise_ops
import numpy as np

flow = tf.load_op_library('FHP_model.so')
```

Реализация на TensorFlow (1)

Исходный тензор – ранг 2

```
Исходный тензор
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
```

Тензор-стенка – ранг 2

```
Тензор-стенка
10000000 10000000 10000000 10000000 10000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 10000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
10000000 10000000 10000000 10000000 10000000
```

bitwise_ops.bitwise_or(Исходный тензор, Тензор-стенка)

```
Исходный тензор со стенками
10000000 10000000 10000000 10000000 10000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 10000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
10000000 10000000 10000000 10000000 10000000
```

Реализация на TensorFlow (2)

Тензор поглощения – ранг 2

```
Тензор-поглощения
11111111 11111111 11111111 11111111 10000000
11111111 11111111 11111111 11111111 00000000
11111111 11111111 11111111 11111111 00000000
11111111 11111111 11111111 11111111 00000000
11111111 11111111 11111111 11111111 10000000
```

Тензор источник – ранг 2

```
Тензор-источник
00111110 00000000 00000000 00000000 00000000
00011010 00000000 00000000 00000000 00000000
01111000 00000000 00000000 00000000 00000000
01001001 00000000 00000000 00000000 00000000
00011111 00000000 00000000 00000000 00000000
```

bitwise_ops.bitwise_and(Исходный тензор со стенками, Тензор-поглощения)

bitwise_ops.bitwise_or(Исходный тензор со стенками и поглощением, Тензор-источник)

```
Исходный тензор со стенками, поглощением и источником
11001010 10000000 10000000 10000000 10000000
01011010 00000000 00000000 00000000 00000000
00101011 00000000 10000000 00000000 00000000
01010001 00000000 00000000 00000000 00000000
11111011 10000000 10000000 10000000 10000000
```

Реализация на TensorFlow (3)

На каждой итерации:

flowTensor = flow.FHP(input=Исходный тензор со стенками и источником, input_rand=Тензор случайных чисел, name='FHP')

resultTensor = bitwise_ops.bitwise_and(flowTensor, Тензор-поглощения)

Эксперимент:

- Размер клеточного автомата – 1000x1000 клеток
- Количество итераций - 100

Время выполнения 100 итераций – 0.67 сек

Спасибо за внимание!