



APPLICATION OF THE TENSOR APPROACH TO THE SOFTWARE IMPLEMENTATION OF THE CELLULAR AUTOMATON FLOW MODEL

N. A. Matolygina, M. L. Gromov, A. K. Matolygin

National Research Tomsk State University,
634050, Tomsk, Russia

DOI: 10.24412/2073-0667-2023-2-74-85

EDN: JIPOXI

Cellular-automaton models are actively used to model physical processes. The cellular automata in these models are large and require a large number of iterations to observe effects of interest. Researchers use parallel technologies to organize calculations in order to get the result quickly. The choice of technology usually depends on the level of the researcher's skills in the field of parallel programming. Parallel programming is a complex skill, and to get it you need to learn a lot of theory and even more practice. We have proposed a special tensor approach to the software implementation of cellular automata to free the researcher from these difficulties and help to create a parallel software product.

A special framework that automatically parallelizes computations at NVIDIA GPU cores is central to the approach. The chosen framework is TensorFlow. The main data structure of TensorFlow is a tensor (multidimensional matrix). Thus, in order to implement a cellular automaton using the tensor approach, it is necessary to represent the cellular automaton as a tensor, and the logic of the automaton transition from one state to another as operations on tensors.

There are two options for applying tensor operations. The first option is to use ready-made operations. The framework implements simple operations that work with ordinary matrices, and more complex operations, such as convolution. In this case, the researcher needs to analyze how the cellular automaton works and select the necessary tensor operations that implement the automaton. The second option is to create your own tensor operation. A custom operation is created using CUDA technology. In this case, the structural elements are ordinary two-dimensional arrays that represent tensors. The user needs to independently allocate the number of threads and blocks required for calculations. The TensorFlow developer libraries allow you to implement operations on data both as programs for the central processor and as programs for graphics adapters in the CUDA C programming language.

In this paper, to demonstrate the performance and capabilities of the tensor approach, the well-known cellular automaton FHP flow model is implemented. There are two phases in this model: collision and propagation. It was decided to implement our own operation, which describes the logic of the automaton, after analyzing the existing TensorFlow operations. Both phases of the FHP model are implemented by us with one custom operation, which is implemented in TensorFlow. The operation takes as input a tensor corresponding to a cellular automaton and a tensor of random numbers to determine the propagation direction. At the output, the operation generates a tensor, to the elements of which the propagation phase and the collision phase are applied. Experiments with the model implemented using the tensor approach were carried out. The gas flow in an open longitudinal pipe is simulated. The particle source is located on one side of the pipe. In the first part of the experiment, an obstacle in the form of a small oblong object is located in the pipe, in the second part — a small

round object. The experimental results are visualized and the picture of the process agrees with the results obtained in other literature sources.

Additionally, a comparative experiment was carried out to evaluate the effectiveness of the tensor approach. In this case, we compared the number of automaton cells processed per second in our implementation on TensorFlow and in an implementation written using only CUDA technology. The results of the comparison showed that when implementing the FHP flow model using the tensor approach, fewer (by 10 or 100 times, depending on the size of the cellular automaton) cells are processed in the same time than in an implementation built on only one CUDA technology. Despite the fact that the implementation of the flow model in TensorFlow is inferior to the implementation written in CUDA in terms of time, the process of building a parallel software implementation is simpler and does not require the researcher to have a deep understanding of the features of parallel programming.

Key words: cellular automaton, tensor approach, gas flow.

References

1. Kalgin K. V. Kletchno-avtomatnoe modelirovanie fiziko-himicheskikh processov na vychisliteljakh s parallel'noj arhitekturoj: dis. . . kand. tehn. nauk. Novosibirsk: 2012. 82 s.
2. Subbotina A. Ju., Hohlov N. I. Realizacija kletchnykh avtomatov «Igra “Zhizn”» i kletchnogo avtomata Kohomoto–Oono s primeneniem tehnologii MPI // Komp'yuternye issledovaniya i modelirovanie. 2010. T. 2. № 3. S. 319–322.
3. Sharifulina A. E. Parallel'naja realizacija kataliticheskoy reakcii ($\text{CO}+\text{O}_2 \rightarrow \text{CO}_2$) // Vestnik JuUrGU. 2012. № 47(306). S. 112–126.
4. Szkoda S., Koza Z., Tykierko M. Accelerating cellular automata simulations using AVX and CUDA // arXiv preprint . 2012. arXiv:1208.2428v1.
5. Kalgin K. V. Realizacija algoritmov s melkozernistym parallelizmom na graficheskikh uskoriteljakh // Sib. zhurn. vychisl. matem. 2011. T. 14. № 1. S. 46–55.
6. TensorFlow [Electron. res.]: <https://www.tensorflow.org>.
7. Shalyapina N. A., Gromov M. L. «Life» in Tensor: Implementing Cellular Automata on Graphics Adapters // Proceedings of the Institute for System Programming of the RAS. 2019. T. 31. N 3. S. 217–228. DOI: [https://doi.org/10.15514/ISPRAS-2019-31\(3\)-17](https://doi.org/10.15514/ISPRAS-2019-31(3)-17).
8. Frisch U., Hasslacher B., Pomeau Y. Lattice-Gas automata for Navier-Stokes equations // Phys. Rev. Lett. 1986. N 56. P. 1505.
9. Tumakov D. N. Tehnologija programmirovaniya CUDA: uchebnoe posobie / Kazanskij gosudarstvennyj universitet. Kazan', 2017. 112 s.
10. Szkoda S., Koza Z., Tykierko M. Multi-GPGPU Cellular Automata Simulations using OpenACC // Zenodo. 2014. P. 1–6. DOI: 10.5281/zenodo.822901.



ПРИМЕНЕНИЕ ТЕНЗОРНОГО ПОДХОДА К ПРОГРАММНОЙ РЕАЛИЗАЦИИ КЛЕТОЧНО-АВТОМАТНОЙ МОДЕЛИ ПОТОКА

Н. А. Матолыгина, М. Л. Громов, А. К. Матолыгин

Национальный исследовательский Томский государственный университет,
634050, Томск, Россия

УДК 004.4‘2

DOI: 10.24412/2073-0667-2023-2-74-85

EDN: LIPOXI

В работе описывается опыт применения тензорного подхода к программной реализации клеточно-автоматной модели потока FHP. Данный подход ориентирован на применение многоядерных видеокарт и специальной рабочей среды, которая автоматически распределяет вычисления по ядрам видеокарты без вмешательства исследователя. Продемонстрирована возможность внедрения пользовательских операций в рабочую среду и проведены компьютерные эксперименты.

Ключевые слова: клеточный автомат, тензорный подход, газовый поток.

Введение. При применении клеточно-автоматных моделей для исследования физических процессов перед исследователями возникает задача программной реализации модели. Для того чтобы пронаблюдать картину протекания «сложного» физического процесса, смоделированного при помощи клеточных автоматов, необходимы клеточные пространства больших размеров и большое количество итераций работы автомата. Так, например, если клеточно-автоматная модель работает в асинхронном режиме, необходимо оперировать пространством в $10^{10} – 10^{12}$ частиц в течение $10^3 – 10^5$ итераций [1]. Поэтому для того чтобы результат моделирования был получен за адекватное время, необходимо организовать вычисления параллельно.

В работах исследователей клеточных автоматов описан опыт распараллеливания вычислений при помощи различных технологий. Выбор конкретной технологии напрямую зависит от уровня навыков в параллельном программировании. Так, например, авторы работы [2] в своих экспериментах с клеточными автоматами используют стандартизованный коммуникационный интерфейс для создания параллельных программ в модели передачи сообщений — MPI (Message Passing Interface). В работе [3] программная реализация блочно-синхронного клеточного автомата построена при помощи связки MPI и стандарта OpenMP (Open Multi-Processing), в [4–5] для реализации клеточно-автоматных моделей применяется технология CUDA (Compute Unified Device Architecture).

Трудоемкость программирования при помощи данных технологий довольно велика и подразумевает самостоятельное распределение вычислительной работы по процессам и организацию обмена данными. Именно поэтому исследователю предстоит больший объем работы и большая ответственность за качество ее выполнения. Для того чтобы освободить

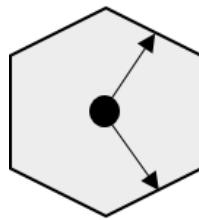


Рис. 1. Форма клетки в модели FHP

исследователя от необходимости осваивать особенности параллельного программирования, можно воспользоваться специальными рабочими средами, которые самостоятельно распараллеливают вычисления и распределяют задания между доступными процессорами вычислительного устройства. В частности, к таким рабочим средам можно отнести фреймворк TensorFlow [6]. Главным преимуществом фреймворка является высокий уровень абстракции, который облегчает разработку программной реализации. Это означает, что исследователю не нужно думать о том, как организовать параллельные участки программы и распределить данные между ними. Его задачей является описание логики программы и ее математической части. На основе использования TensorFlow ранее нами был предложен подход к программной реализации синхронных клеточных автоматов, который называется тензорным [7]. Для того чтобы оценить применимость данного подхода в задаче моделирования физических процессов, мы обратились к известной клеточно-автоматной модели потока FHP [8] и провели компьютерный и сравнительный эксперименты.

1. Клеточный автомат и модель FHP. Клеточный автомат — это множество автоматов Мура $A = \{a_{i,j} | (i,j) \in \mathbb{Z} \times \mathbb{Z}\}$, заданное на некоторой функции окрестности $N(i,j)$. Каждый элемент $a_{i,j} = \langle S, \hat{s}, I, O, \varphi, \psi \rangle$ множества A есть клетка, где S — конечное непустое множество состояний автомата с выделенным начальным состоянием $\hat{s} \in S$, $I = S^n$ — конечное множество входных воздействий, $O = S$ — конечное множество выходных реакций, $\varphi: S \times I \rightarrow S$ — полностью определенная функция переходов, $\psi: S \rightarrow O$ — полностью определенная функция выходов.

В модели FHP состояние клетки ассоциируется с набором содержащихся в ней частиц. Клетки классифицируются по типу: 1) рабочая клетка; 2) клетка-стенка. Частицы классифицируются по скорости: 1) движущаяся частица; 2) частица покоя. Особенностью рассматриваемой модели потока является то, что в клетке не может быть больше одной частицы покоя и больше одной частицы, движущейся в заданном направлении. Каждая клетка имеет форму шестиугольника, и ее состояние описывается булевым вектором длины 8: $s(a_{i,j}) = (s_1, s_2, \dots, s_8)$. Компонента s_1 идентифицирует тип клетки: если $s_1 = 0$, то клетка рабочая, если $s_1 = 1$, то клетка — стенка. Компонента s_8 определяет наличие в клетке частицы покоя: если $s_8 = 0$, то частицы покоя в клетке нет, если $s_8 = 1$, то в клетке присутствует частица покоя. Компоненты s_2, \dots, s_7 отвечают за направления движущихся частиц, например, для состояния (00001011) направления в клетке будут иметь следующий вид (рис. 1).

Клеточный автомат работает в двухтактном синхронном режиме. Первый такт — фаза сдвига, частицы перемещаются в соседние клетки по направлению вектора скорости. Второй такт — фаза столкновения, клетки переходят в новое состояние согласно правилам столкновения, которые носят вероятностный характер и определяются только внутренним состоянием клетки на предыдущем такте.

2. Тензорный подход к реализации клеточных автоматов. Предлагаемый подход к реализации включает аппаратную и программную составляющие. Аппаратная составляющая представляет графический процессор от фирмы NVIDIA, поддерживающий программно-аппаратную архитектуру параллельных вычислений CUDA. Программная часть CUDA включает элементы, необходимые для разработки параллельных программ:

- CUDA C — язык программирования (потоковое расширение языка C++);
- компилятор NVCC;
- API;
- набор библиотек.

Программная составляющая подвода включает специальную рабочую среду — фреймворк TensorFlow. Основной структурой данных в TensorFlow является многомерная матрица, которая в терминах этого фреймворка называется *тензором*. Тензоры характеризуются четырьмя параметрами:

- Ранг — количество измерений тензора.
- Форма — количество компонентов по каждому измерению тензора.
- Размер — общее количество компонентов тензора.
- Тип — тип данных, назначенный компонентам тензора (целочисленный восьмибитный, шестнадцатибитный с плавающей запятой и т. п.).

Работа с TensorFlow построена вокруг создания и выполнения графа вычислений. Граф вычислений — это граф потоков данных, в котором математические операции представлены в виде узлов, а данные — в виде ребер между этими узлами. Граф никогда не может быть циклическим, и каждая операция приводит к формированию нового тензора.

Таким образом, для представления клеточного автомата в рамках тензорного подхода необходимо сам клеточный автомат представить в виде тензора, а логику перехода автомата из одного состояния в другое — при помощи операций над тензорами. Для тензоров справедливо большинство арифметических операций, работающих с матрицами, а именно: сложение, вычитание, поэлементное умножение и деление, матричное умножение и т. д. В TensorFlow реализованы основные математические операции, функции и алгоритмы. Кроме этого, функционал фреймворка позволяет встраивать в него пользовательские операции. Библиотеки разработчиков позволяют реализовывать операции над данными как программы для центрального процессора, так и как программы для графических процессоров.

3. Представление модели FHP в рамках тензорного подхода.

3.1. *Архитектура модели.* Реализация клеточно-автоматной модели потока FHP может быть осуществлена в два этапа. На первом этапе необходимо отобразить шестиугольную форму клеток автомата на прямоугольную без нарушения соседства. На втором этапе необходимо выразить фазы сдвига и столкновения при помощи тензорных операций.

На рис. 2 продемонстрирован способ отображения, в котором тип соседства чередуется в зависимости от четности столбца. Для этого поле клеток будем изображать немного «не классически» (рис. 2, а): слева классический вид поля (клетки «углом вверх»), справа — вид, применяемый в нашей работе («стороной вверх»).

3.2. *Реализация тензорных операций на CUDA.* В случае если вычисления, требуемые для решения задачи, невозможно выполнить стандартными операциями TensorFlow по причине их отсутствия или выбор тензорных операций неочевиден, в фреймворке имеется возможность внедрять собственные операции при помощи технологии CUDA. Технология CUDA позволяет определять специальные функции — ядра, которые выполняются па-

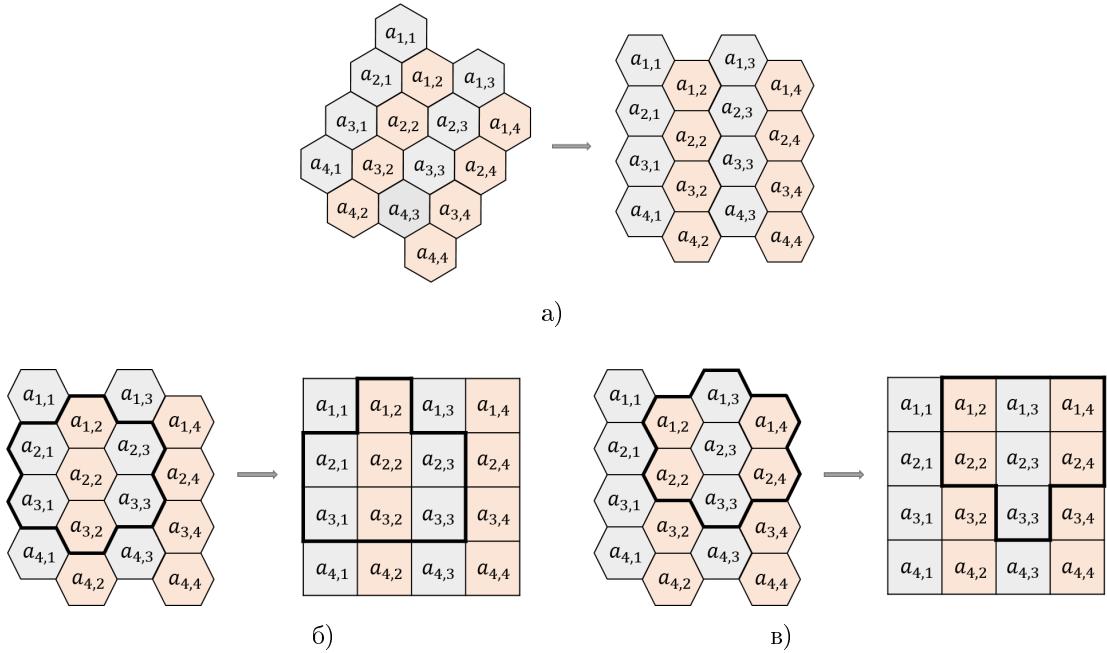


Рис. 2. Способ отображения шестиугольной формы клеток на прямоугольную: а) вид поля клеточного автомата (слева «классический», справа применяемый в нашей работе); б) тип соседства для нечетных столбцов; в) тип соседства для четных столбцов

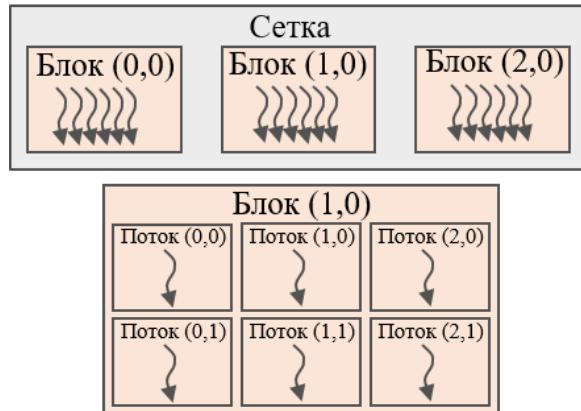


Рис. 3. Структура организации вычислений в CUDA

ралльно на GPU в виде множества потоков. Отдельные потоки группируются в блоки потоков одинакового размера. Блоки потоков объединяются в сетки блоков потоков (рис. 3) [9].

Потоки, блоки и сетки имеют встроенные параметры: *threadIdx* — индекс потока внутри блока, *blockIdx* — индекс блока внутри сетки, *blockDim* — размер блока в потоках, *gridDim* — размер сетки в блоках. Каждый параметр является структурой с полями *.x*, *.y*, *.z*.

Для создания пользовательской тензорной операции необходимо: 1) определить интерфейс операции; 2) реализовать ядро для операции.

Интерфейс операции определяется в файле «.cc», содержащем исходный код программы на языке C++, в теле макроса REGISTER_OP, осуществляющем регистрацию опе-

```

1 REGISTER_OP("FHP")
2     .Attr("T: numbertype")
3     .Input("input: T")
4     .Input("input_rand: float32")
5     .Output("output: T")
6     .SetShapeFn([](::tensorflow::shape_inference::InferenceContext* c) {
7         c->set_output(0, c->input(0));
8         return Status::OK();
9     });

```

Рис. 4. Фрагмент программы, определяющий интерфейс операции

```

load("//tensorflow:tensorflow.bzl", "tf_custom_op_library")

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

```

Рис. 5. Пример BUILD-файла

рации. Указываются имя операции, типы и имена входных и выходных файлов, а также строки документации и атрибуты, если таковые требуются для операции. Интерфейс операции для модели потока проиллюстрирован на рис. 4.

На рис. 4 в строке 1 задается имя операции — *FHP*, в строке 2 через атрибут *.Attr* определяется размер входного тензора *T* типа *numbertype*, в строках 3–5 определяются имена и типы для входов и выходов¹, в строке 6 через функцию формы *.SetShapeFn()* объявляется, что форма выходного тензора *output* совпадает с формой первого входного тензора *input*.

Компиляция операции происходит с помощью утилиты Bazel, исполняющей BUILD-файл, в котором перечисляются файлы с реализацией для CPU ".cc", для GPU ".cu.cc" на языке CUDA C и имя файла ".so" для скомпилированной операции. Пример BUILD-файла приведен на рис. 5.

Недостатком вычислений на CUDA является отсутствие возможности синхронизации потоков разных блоков. Соответственно, чтобы гарантировать законченность вычислений в некоторой области памяти, необходима дополнительная синхронизация. В фреймворке TensorFlow этот недостаток устраняется следующим образом. Каждая программная функция, вычисляющая операцию над тензорами, получает на вход кроме тензоров операндов операции (они называются *входными тензорами*) еще и тензор, в который необходимо поместить результат (он называется *выходным тензором*). Входные тензоры всегда должны оставаться неизменными, из них можно только читать значения, а результат локальных вычислений необходимо помещать в соответствующую компоненту выходного тензора. Такой подход требует дополнительной памяти даже в тех случаях, когда можно было бы обойтись и без нее (например, при смене знаков у всех компонентов тензора), однако он гарантирует, что разные вычислительные потоки, запущенные в разное время, работают с

¹Для хранения состояния одной клетки достаточно одного байта. Однако, согласно документации фреймворка TensorFlow, при регистрации операции могут быть использованы только 32-битные типы данных (int32, float32). Возможно, в будущих версиях TensorFlow данное ограничение будет снято.

```

1 // Определение функции-ядра
2 __global__ void FlowKernel(const int size,
3                             int* in,
4                             float* rand,
5                             int* out)
6 {
7     int start = blockDim.x * blockIdx.x + threadIdx.x;
8     int stride = blockDim.x * gridDim.x;
9     for (int i = start; i < size; i += stride) {
10         ...// Фаза сдвига
11         ...// Фаза столкновения
12     }
13 }
```

Рис. 6. Фрагмент программы, определяющий функцию-ядро операции

одними и теми же входными данными, поэтому синхронизации между ними не требуется. Если, конечно, эти потоки не конкурируют за доступ (запись) к одним и тем же элементам выходного тензора. Но подобную ситуацию стоит отнести к ошибкам, и ее нужно исправлять, а не пытаться синхронизовать потоки.

3.3. Особенности реализации модели FHP. В модели FHP фазы сдвига и столкновения реализованы одной пользовательской операцией и имеют одну функцию-ядро *FlowKernel* (рис. 6). В качестве параметров в функцию передаются: размер матрицы *size*, представляющей клеточный автомат (предполагается, что матрица имеет *N* строк и *M* столбцов и хранится в виде одномерного массива по строкам), сама матрица *in*, матрица случайных чисел из диапазона [0, 1) *rand* того же размера, что и *in*, и матрица для хранения результата *out*. Каждый поток блока потоков вычисляет один элемент матрицы *out*. Индекс текущего потока вычисляется в переменной *start* (строка 7 рис. 6), шаг *stride* (строка 8 рис. 6) равен размеру одной строки.

Каждый вычислительный поток *i* определяет частицы, которые должны попасть в текущую клетку из соседних клеток, и кладет результат в ячейку *out[i]* (тензоры, передаваемые на вход функций, разворачиваются TensorFlow в одномерные массивы). Тем самым осуществляется фаза сдвига. Далее выполняется фаза столкновения. При выполнении этой фазы программа выбирает одну из двух веток вычислений в зависимости от типа клетки. В случае, если клетка — стенка, то для нее реализуется отражение, т. е. частицы, находящиеся в клетке, меняют направление своего движения на противоположное (отражение без трения). В случае, если клетка является рабочей, выбор нового состояния осуществляется с помощью таблицы переходов, которая представлена двумерным массивом. Поскольку один бит 8-битного булева вектора, кодирующего состояние клетки, идентифицирует стенку, то количество различных состояний рабочей клетки, а значит и элементов массива переходов, равно $2^7 = 128$. Сами булевые векторы интерпретируются как целые двоичные числа, которые рассматриваются как индексы ячеек массива. Зная индекс, выбирается соответствующий элемент массива. Далее, в зависимости от того, сколько для текущего состояния определено переходов, случайным образом выбирается один. Для этого диапазон вещественных чисел [0; 1) разбивается на равные поддиапазоны по количеству возможных переходов, и проверяется, в какой поддиапазон попадает значение элемента тензора *rand*, соответствующего рассматриваемой клетке. Номер под-

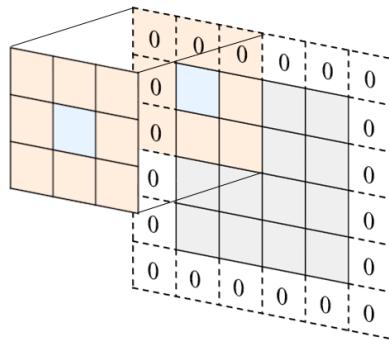


Рис. 7. Иллюстрация тензорной операции свертки с типом отступа SAME

диапазона задает номер перехода. Например, если возможных переходов 3, то если $p < 0,33$, выбирается первый переход, если $p \geq 0,33$ и $p < 0,66$, то второй, иначе — третий.

4. Результаты экспериментов. Для проведения компьютерных экспериментов был рассмотрен двумерный случай, когда поток движется вдоль ограничивающих его стенок (т. е. имитация движения по трубе). С одной стороны перпендикулярно стенкам располагается источник частиц, с другой стороны частицы поглощаются. После выполнения заданного количества итераций работы клеточного автомата производилось осреднение полученных значений по окрестности некоторого диаметра. Осреднение реализовано с использованием возможностей TensorFlow, а именно стандартной тензорной операции свертки — `tf.nn.conv2d`, аргументами которой являются два тензора ранга 2, соответствующие клеточному автомatu и окрестности осреднения, шаг окрестности осреднения и тип отступа окрестности осреднения. На рис. 8 показан используемый в работе тип отступа SAME. Здесь окрестность осреднения (обозначено желтым цветом) выходит за пределы клеточного автомата (обозначено серым цветом), но при этом сам автомат дополняется недостающими строками и столбцами, элементы которых заполняются нулями (обозначено белым цветом).

В первой части эксперимента был выбран случай, когда внутри трубы расположено препятствие в виде небольшого предмета продолговатой формы. Размер клеточного автомата — 100×1000 клеток. Результат выполнения 10000 итераций представлен на рис. 8. Рисунок представляет собой фрагмент области моделирования с препятствием. Серым цветом проиллюстрированы клетки-стенки, синим — стрелки, представляющие осредненные векторы скорости частиц. Начало стрелки располагается в центре окрестности осреднения, диаметр которой составляет 9 клеток. Длина стрелки соответствует модулю усредненного вектора скорости.

Во второй части эксперимента был выбран случай, когда внутри трубы расположено препятствие в виде небольшого предмета округлой формы. Результат выполнения 10000 итераций представлен на рис. 9. Рисунок представляет собой фрагмент области моделирования с препятствием.

Выполнение 10000 итераций заняло в нашей системе (процессор Intel Core i7-8700K, 3.7 ГГц, объем ОЗУ 24 ГБ, ОС Windows 10-x64, видеокарта GeForce RTX 2080 Ti (ОЗУ 11 ГБ, эффективная частота памяти 14000 МГц, базовая частота ядра 1350 МГц, 4352 ядер CUDA)) 21 секунду.

Третья часть эксперимента проведена с целью оценки вычислительных возможностей применяемого тензорного подхода. Для этого размеры клеточного автомата в модели FHP

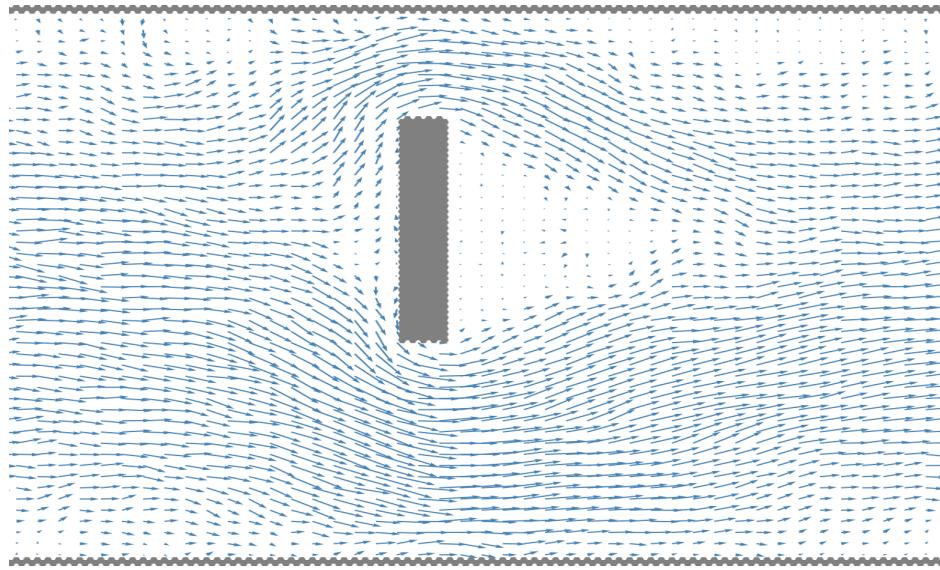


Рис. 8. Результат моделирования потока с препятствием продолговатой формы при помощи модели FHP

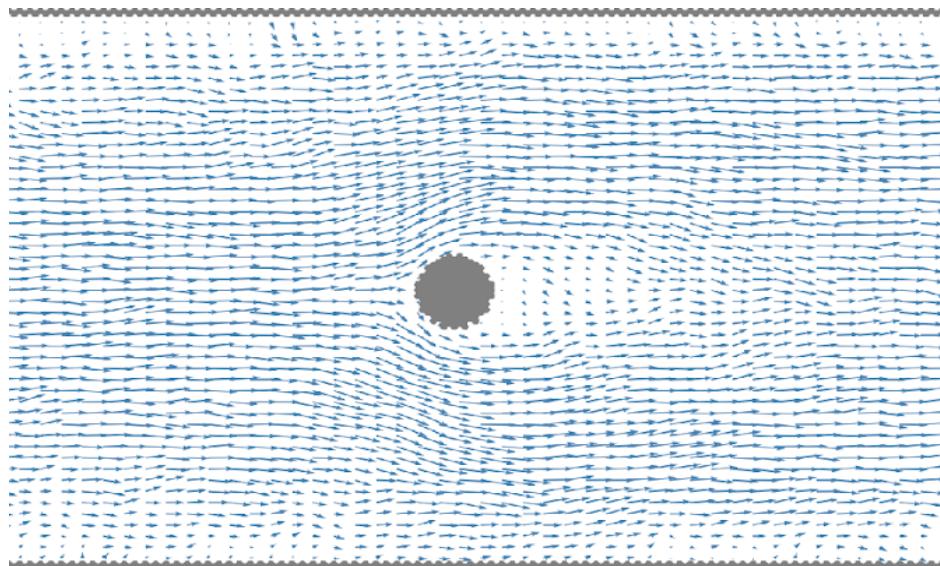


Рис. 9. Результат моделирования потока с препятствием окружной формы при помощи модели FHP

были увеличены в 100 раз, т. е. 1000×10000 клеток. Расчет такого же количества итераций работы автомата занял 600 секунд.

Известна работа [10], в которой авторы проводили вычислительные эксперименты с FHP моделью потока при помощи только технологии CUDA. Вычисления проводились на видеокартах NVIDIA нескольких поколений, и в качестве параметра эффективности выбрано количество клеток, обрабатываемых за секунду. Так, например, на видеокарте Tesla K20m (ОЗУ 5 ГБ, эффективная частота памяти 5200 МГц, базовая частота ядра 706 МГц, 2496 ядер CUDA) за 1 секунду было обработано $57 \cdot 10^8$ клеток. В нашем же эксперименте для автомата размером 100×1000 клеток за 1 секунду было обработано $47 \cdot 10^6$ клеток, а для автомата размером 1000×10000 клеток — $17 \cdot 10^7$ клеток при 10000 итераций работы.

К сожалению, авторы работы [10] не указывают, какого размера автомат и какое количество итераций было выбрано для проведения экспериментов. Но даже без этого знания можно сделать вывод о том, что при реализации модели потока FHP при помощи тензорного подхода за одно и тоже время обрабатывается меньше (в 10 или 100 раз, в зависимости от размера клеточного автомата) клеток, чем в реализации, построенной только на одной технологии CUDA. Мы полагаем, что такое отличие в производительности кроется в особенностях организации фазы столкновения, которая, согласно авторам модели FHP, в некоторых случаях требует выбирать переход случайным образом. В нашей реализации для этого на каждой итерации генерируется тензор со случайными компонентами. Этот тензор затем подается на вход функции, вычисляющей следующее глобальное состояние автомата. То есть в программе итеративно выполняются следующие шаги: видеокарта генерирует тензор со случайными компонентами, тензор глобального состояния и «случайный» тензор загружаются в память видеокарты, видеокарта вычисляет следующее глобальное состояние, тензор глобального состояния выгружается из памяти видеокарты в общую память компьютера, и процесс повторяется. Авторы работы [10] вместо случайного выбора перехода применяют его детерминированный аналог: каждая клетка хранит направление (по или против часовой стрелки), в котором система сталкивающихся частиц была развернута в предыдущий раз, и в случае выполнения столкновения разворачивают систему частиц в противоположном направлении. В результате после большого количества итераций суммарный момент поворотов оказывается равным 0. Подобный подход бесспорно позволяет сократить как время вычислений, так и накладные расходы (нет необходимости каждый раз генерировать новые случайные числа и «гонять» тензоры из и в общую память компьютера), но авторы работы не представили доказательств, что получившаяся модель сохранит свои свойства по сравнению с исходной FHP моделью. И все же, мы также планируем применить подобный подход в будущем и сравнить время моделирования обновленной реализации с реализацией из работы [10].

Заключение. В работе продемонстрирована возможность применения тензорного подхода к программной реализации клеточно-автоматной модели потока FHP. Результаты компьютерных экспериментов показали, что реализация модели потока на TensorFlow уступает реализации, написанной на «чистой» CUDA во временном смысле. Мы связываем это с накладными расходами на генерацию случайных чисел и на передачу данных из общей памяти компьютера в память видеокарты и возвращении результата из памяти видеокарты в память компьютера. Однако в будущем мы планируем применить то же ухищрение, что и авторы работы [10]. При этом сохранится главное неоспоримое преимущество реализации клеточно-автоматной модели при помощи тензорного подхода: простота процесса построения параллельной программной реализации. Нет необходимости отдельно организовывать параллельные участки программ и самостоятельно распределять данные между ними, всю ответственность за это берет на себя фреймворк TensorFlow, который эффективно эксплуатирует многоядерность видеокарты.

Список литературы

1. Калгин К. В. Клеточно-автоматное моделирование физико-химических процессов на вычислителях с параллельной архитектурой: дис. . . канд. техн. наук. Новосибирск: 2012. 82 с.

2. Субботина А. Ю., Хохлов Н. И. Реализация клеточных автоматов «Игра “Жизнь”» и клеточного автомата Кохомото–Оно с применением технологии MPI // Компьютерные исследования и моделирование. 2010 Т. 2. № 3 С. 319–322.
3. Шарибулина А. Е. Параллельная реализация каталитической реакции ($\text{CO} + \text{O}_2 \rightarrow \text{CO}_2$) // Вестник ЮУрГУ. 2012. № 47(306). С. 112–126.
4. Szkoda S., Koza Z., Tykierko M. Accelerating cellular automata simulations using AVX and CUDA // arXiv preprint. 2012. arXiv:1208.2428v1.
5. Калгин К. В. Реализация алгоритмов с мелковзернистым параллелизмом на графических ускорителях // Сиб. журн. вычисл. матем. 2011. Т. 14. № 1. С. 46–55.
6. TensorFlow. [Электрон. рес.]: <https://www.tensorflow.org>.
7. Shalyapina N. A., Gromov M. L. «Life» in Tensor: Implementing Cellular Automata on Graphics Adapters // Proceedings of the Institute for System Programming of the RAS. 2019. Т. 31. № 3. С. 217–228. DOI: [https://doi.org/10.15514/ISPRAS-2019-31\(3\)-17](https://doi.org/10.15514/ISPRAS-2019-31(3)-17).
8. Frisch U., Hasslacher B., Pomeau Y. Lattice-Gas automata for Navier-Stokes equations // Phys. Rev. Lett. 1986. N 56. P. 1505.
9. Тумаков Д. Н. Технология программирования CUDA: учебное пособие / Казанский государственный университет. Казань, 2017. 112 с.
10. Szkoda S., Koza Z., Tykierko M. Multi-GPGPU Cellular Automata Simulations using OpenACC // Zenodo. 2014. Р. 1–6. DOI: 10.5281/zenodo.822901.



Матолыгина Наталия Андреевна — лаборант кафедры информационных технологий в исследовании дискретных структур ТГУ. E-mail: nat.shalyapina@gmail.com. Область научных интересов: клеточные автоматы, программирование, математическое моделирование.

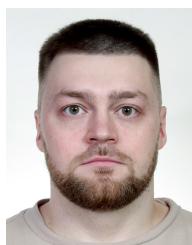
Matolygina Natalia Andreevna — laboratory assistant of the Department of Information Technologies in the Study of Discrete Structures. Research interests: cellular automata, programming, mathematical modeling.



Громов Максим Леонидович — канд. физ. мат. наук, доцент кафедры информационных технологий в исследовании дискретных структур ТГУ. E-mail: Maxim.leo.gromov@gmail.com. Область научных интересов: теория автоматов, тестирова-

ние на основе формальных моделей, программирование.

Gromov Maxim Leonidovich — Candidate of Physico-Mathematical Sciences, Associate Professor at the Department of Information Technologies in the Study of Discrete Structures. Research interests: finite state machines, testing based on formal models, programming.



Матолыгин Арсений Константинович — аспирант кафедры информационных технологий в исследовании дискретных структур ТГУ. E-mail: amatolygin@mail.ru. Область научных интересов: клеточные автоматы, программирование.

Matolygin Arseniy Konstantinovich — Postgraduate at the Department of Information Technologies in the Study of Discrete Structures. Research interests: cellular automata, programming.

Дата поступления — 09.03.2023