

# Об алгоритме проверки наличия подквазигруппы в квазигруппе

Собянин П.И.

Рассматривается алгоритм проверки наличия подквазигруппы в квазигруппе. Сравнивается скорость выполнения его последовательной и параллельной версий на вычислительной архитектуре CUDA.

**Ключевые слова:** квазигруппа, подквазигруппа, GPU, CUDA.

## 1. Введение

В некоторых задачах криптографии оказывается полезным применение квазигрупп. Примером таких задач могут служить:

- построение кодов аутентификации (А-кодов) [1];
- шифрование [1];
- построение однонаправленных функций [1].

В работе сравнивается производительность последовательной [2] и параллельной версий алгоритма, устанавливающего наличие или отсутствие подквазигруппы в заданной квазигруппе. Вычисления проводились на GPU с архитектурой CUDA[3].

Задача данной статьи – исследовать разницу в производительности двух версий алгоритма. Насколько можно судить по имеющимся публикациям, подобное сравнение еще не проводилось.

Дальнейшая часть статьи построена следующим образом: в разделе 2 рассматривается сам алгоритм поиска подквазигрупп в заданной квазигруппе; раздел 3 посвящен описанию сравнительного эксперимента, его результатам и выводам; в разделе 4 подведены итоги текущей работы и освещены дальнейшие планы в предметной области.

## 2. Описание алгоритма

Пусть  $Q$  – конечная квазигруппа с элементами  $q_1, q_2, \dots, q_n, n \in N$ , и операцией  $f$ . Перед запуском алгоритма нормализуем вид квазигруппы, заменяя ее элемент  $q_i$  на его индекс  $i$ .

Рассмотрим следующую процедуру:

- 1) Рассмотрим очередной элемент квазигруппы  $i_k, 1 \leq k \leq n$ .
- 2) Инициализируем множество  $SQ = \{i_k\}$ .
- 3) Инициализируем множество  $SU = \emptyset$ .
- 4) Если  $|SQ| = |SU|$  или  $|SQ| = |Q|$ , то перейти к шагу 6, иначе перейти к шагу 5.
- 5) Рассмотрим произвольный индекс  $j \in SQ \setminus SU$ . Рассмотрим всевозможные пары  $(k, l)$ , такие, что по крайней мере одно из значений  $k$  и  $l$  равно  $j$ , а оставшийся элемент либо равен  $j$ , либо принадлежит  $SU$ . Для каждой пары вычислим  $t = f(k, l)$ . Если  $t \notin SQ$ , то добавляем  $t$  к  $SQ$ . После рассмотрения всех пар добавляем  $j$  к  $SU$  и переходим к шагу 4.
- 6) Если  $k$  из шага 1 равен  $n$ , то перейти к шагу 7, иначе – к шагу 1, увеличив  $k$  на единицу.
- 7) Конец.

Другими словами, здесь  $SQ$  – это множество-кандидат в подквазигруппы,  $SU$  – множество просмотренных элементов. Последовательная версия алгоритма поочередно рассматривает элементы квазигруппы в качестве начальных элементов множества-кандидата. Параллельная же версия рассматривает всех кандидатов "одновременно", запуская алгоритм на множествах с разными начальными элементами на соответствующих им разных вычислительных потоках.

Оригинальное описание алгоритма приведено в [3].

## 3. Эксперимент

**Описание.** В качестве входных данных эксперимента генерировались наборы из 10 случайных квазигрупп заданного порядка. Порядок квазигрупп варьировался от 4 до 64. Алгоритм запускался на каждой квазигруппе из набора, замерялось время его выполнения в микросекундах.

Таким образом, для каждого набора входных данных на выходе мы получили информацию о времени работы алгоритма на каждом элементе набора. По каждому набору выходных данных брались минимальное, максимальное и среднее время исполнения, и по этим величинам строились сравнительные графики.

Мы тестировали версии алгоритма на видеокарте Tesla C2070, на которой установлен графический процессор Tesla T20 с 448 вычислительными ядрами, работающими на частоте 1.15 ГГц.

**Результаты.** По оси  $X$  на всех рисунках отложен порядок входной квазигруппы. По оси  $Y$  на всех рисунках отложено время выполнения операции в микросекундах.

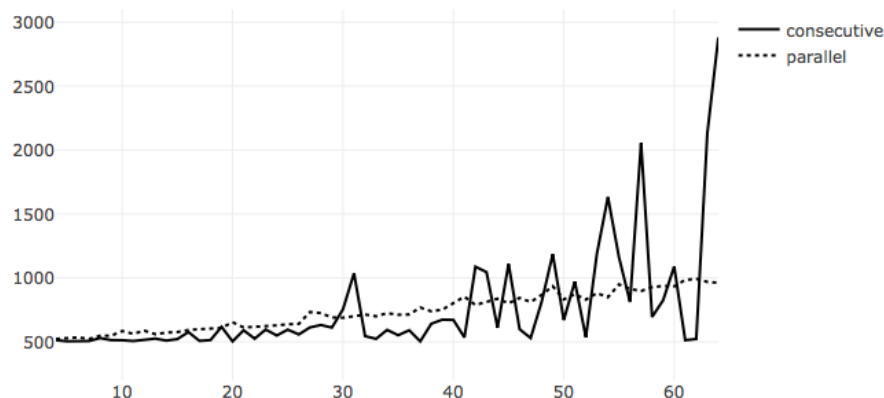


Рис. 1. Быстрейшее время работы двух версий алгоритма в зависимости от порядка квазигруппы.

Результаты тестов показывают, что порядок роста времени совпадает с теоретическим: в параллельном случае рост линейный, а в последовательном – порядка  $O(n^2)$ , как и ожидалось.

На рис. 1 видно, что зачастую последовательный алгоритм обрабатывает быстрее параллельного. Это объясняется тем, что последователь-

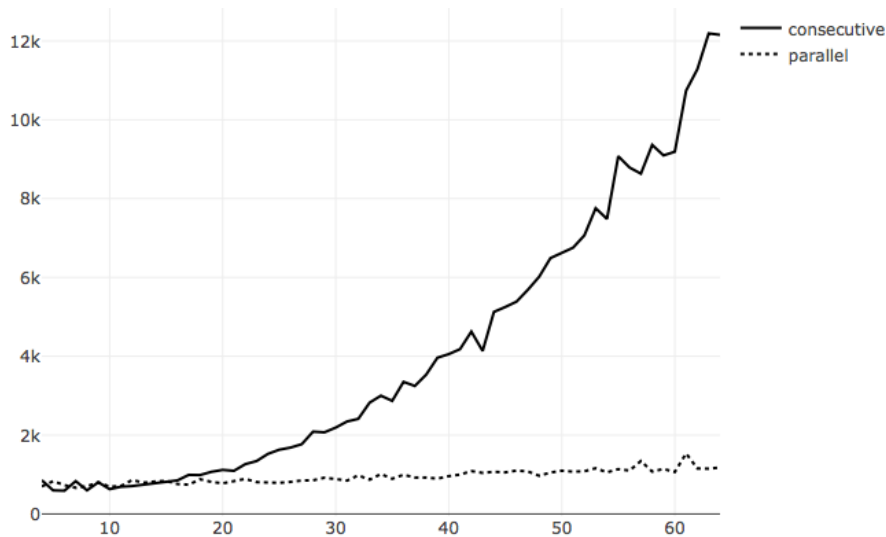


Рис. 2. Наименьшая скорость работы двух версий алгоритма в зависимости от порядка квазигруппы.

ный алгоритм завершает свою работу сразу, как только подквазигруппа будет найдена, т.е. цикл из шага 1 обрывается до его завершения; параллельная же версия дожидается окончания работы процедуры на каждом используемом в данный момент вычислительном потоке и только тогда завершает свою работу. Время работы алгоритма на каждом потоке, вообще говоря, различно – это зависит от входного элемента. Так, например, какой-то поток может найти подквазигруппу за одну итерацию и завершить работу; на другом же потоке алгоритм может установить отсутствие подквазигруппы с входным элементом, и для этого ему потребуется максимум итераций, что влечет за собой увеличение времени работы.

#### 4. Заключение

Параллельная версия алгоритма дает значительное ускорение по сравнению с последовательной версией. В дальнейшем планируется обобщить

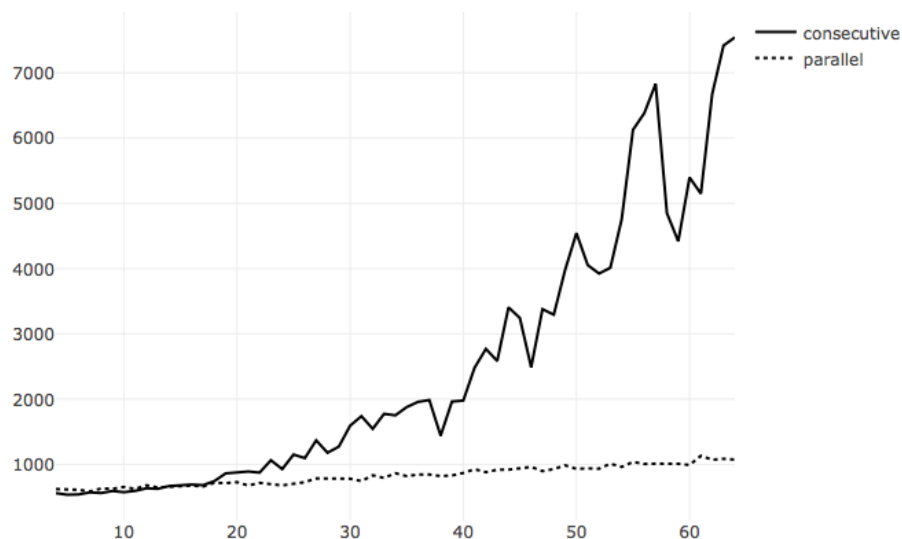


Рис. 3. Среднее время работы двух версий алгоритма в зависимости от порядка квазигруппы.

этот алгоритм на случай  $n$ -квазигрупп, проверить его корректность и оценить его сложность.

Автор выражает искреннюю признательность А. В. Галатенко за постановку задачи и обсуждение результатов работы.

## Список литературы

- [1] М. М. Глухов, “О применениях квазигрупп в криптографии”, *ПДМ*, **2**, 2008, 28–32.
- [2] “Исходный код алгоритма <https://github.com/nikita-toropov/rf/blob/master/main.cpp>”.
- [3] “NVIDIA CUDA <http://developer.nvidia.com/object/cuda.html>”.
- [4] Н.А. Торопов, *Алгоритм проверки наличия подквазигрупп в квазигруппе*, Филиал Московского Государственного Университета в г. Ташкенте, 2018.

**About algorithm which checks if subquasigroup exists in a given quasigroup**  
**Sobyenin P.I.**

Algorithm which checks if subquasigroup exists in a given quasigroup. Performance of its consequent and CUDA parallel versions are compared.

**Keywords:** quasigroup, subquasigroup, GPU, CUDA.