Серия «Математика»
2011. Т. 4, № 4. С. 12—26

Онлайн-доступ к журналу:
http://isu.ru/izvestia

ИЗВЕСТИЯ
Иркутского
государственного
университета

УДК 519.673

Приближенный алгоритм вычисления сложности обратимой функции в базисе Тоффоли *

С. Ф. Винокуров, А. С. Францева
Восточно-Сибирская государственная академия образования

Аннотация.
В работе исследуются вопросы нахождения сложности обратимой функции при представлении ее обратимой схемой. Разработаны и реализованы алгоритмы минимизации обратимых схем в базисе Тоффоли [5]; приведены оценки сложности обратимых функций.

Ключевые слова: обратимые функции, сложность, базис Тоффоли, параллельные алгоритмы

Интерес к обратимым вычислениям на сегодняшний день связан с минимизацией потери энергии при преобразовании информации в современных вычислительных устройствах. С другой стороны, обратимые вычисления являются существенной частью квантовых вычислений. Более подробно об этом можно посмотреть в [2].

В работе исследуется вопрос о нахождении сложности обратимой функции в схемном представлении. В статье даны представления обратимой функции в виде таблицы значений, перестановочной матрицы, перестановки и логической схемы. Условимся, что в дальнейшем изложении понятия обратимой функции, таблицы значений, перестановочной матрицы, перестановки и логической схемы, если не указано обратное, воспринимаются как синонимы.

В качестве базиса логических схем для реализации обратимых функций использован известный базис Тоффоли. Можно заметить, что некоторые его элементы реализованы аппаратно [4].

Понятие обратимой функции и ее представления

Булевой (n, k)-функцией $f = (f_1, ..., f_k)$ будем называть отображение из множества $\{0, 1\}^n$ в множество $\{0, 1\}^k$. Синоним для таких функций — многовыходные булевы функции.

* Работа выполнена при финансовой поддержке РФФИ, проект 09-01-00476а.

Под обратимой функцией $f(x_1, ..., x_n)$ будем понимать такую булеву (n, n) функцию $(f_1(x_1, ..., x_n),...,f_n(x_1, ..., x_n))$, что отображение

$$f : \{(\alpha_1, ..., \alpha_n)\} \to \{(f_1(\alpha_1, ..., \alpha_n), ..., f_n(\alpha_1, ..., \alpha_n))\} \quad (*)$$

является однозначным.

Представления обратимой функции.

1. *Таблица значений.* Пусть $\tilde{\alpha}$ обозначает некоторый двоичный набор $(\alpha_1, ..., \alpha_n)$, $\tilde{0} = (0, ..., 0)$ – нулевой набор, $\tilde{1} = (1, ..., 1)$ – единичный набор. Все наборы $\tilde{\alpha}$ лексикографически упорядочиваются и каждому набору $\tilde{\alpha}$ сопоставляется набор $\tilde{\beta}_{\tilde{\alpha}}$ такой, что $\tilde{\beta}_{\tilde{\alpha}} = (f_1(\tilde{\alpha}), ..., f_n(\tilde{\alpha}))$. Тогда получается следующая таблица значений обратимой функции f:

$(\alpha_1, \dots \alpha_n)$	$\tilde{\beta}_{(\alpha_1,...\alpha_n)}$
$(0, \dots, 0)$	$(f_1(\tilde{0}), \dots, f_n(\tilde{0}))$
$\vdots$	$\vdots$
$(\gamma_1, \dots, \gamma_n)$	$(f_1(\tilde{\gamma}), ..., f_n(\tilde{\gamma}))$
$\vdots$	$\vdots$
$(1, \dots, 1)$	$(f_1(\tilde{1}), \dots, f_n(\tilde{1}))$

2. *Перестановка.* Пусть набору $\tilde{\alpha} = (\alpha_1, ..., \alpha_n)$ соответствует число $X = \alpha_1 \cdot 2^{n-1} + \alpha_2 \cdot 2^{n-2} + \dots \alpha_n \cdot 2^0$. Тогда, согласно определению (*) обратимой функции, ее таблицу значений можно ассоциировать с перестановкой:

$$P_f = \begin{pmatrix} 0 & 1 & 2 & \dots & X & \dots & 2^n - 1 \\ i_0 & i_1 & i_2 & \dots & i_X & \dots & i_{2^n-1} \end{pmatrix}$$

3. *Перестановочная матрица.* Индуктивно введем однозначное соответствие между двоичными наборами и бинарными векторами через следующее отображение φ :

$$\varphi(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \ \varphi(1) = \begin{pmatrix} 0 \\ 1 \end{pmatrix};$$
$$\varphi(\alpha_1...\alpha_{n-1}\alpha_n) = \varphi(\alpha_1...\alpha_{n-1}) \otimes \varphi(\alpha_n),$$

где $\otimes$ – кронекерово произведение векторов [3].

Поскольку набор $(\alpha_1...\alpha_{n-1}\alpha_n)$ интерпретируется как бинарная запись числа, то, очевидно, φ отображает этот набор в вектор длины 2^n с единицей на месте с номером $(\alpha_1...\alpha_{n-1}\alpha_n)$ и нулями на остальных местах; нумерация начинается с 0.

Например, $\varphi(01) = \varphi(0) \otimes \varphi(1) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$

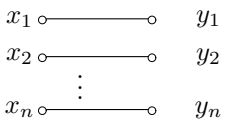
Матрица M_f имет следующий вид. Строка M_f с номером $\tilde{\alpha}$ есть бинарный вектор, соответствующий набору $\tilde{\beta_{\tilde{\alpha}}}$. Действие функции f на набор $\tilde{\alpha}$ соответствует умножению матрицы M_f на вектор $\varphi(\tilde{\alpha})$.

4. *Схемное представление обратимых функций.*

Структурное описание схемы. Логическая схема S, как объект, имеет n входов и n выходов, попарно соединенных между собой.

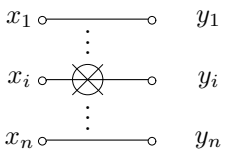
Пусть заданы имена $X = \{x_i\}$ и $Y = \{y_i\}$, $i \in \{1, ..., n\}$ и пусть входам логической схемы S сопоставлены имена x_i, а выходам – y_i.

Тривиальная логическая схема изображается следующим образом:



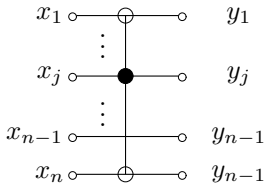
Логическая схема S содержит элементы двух типов.

1. Элементы N_i:



Здесь круг «помещается» на любой вход x_i, где $i \in \{1, 2, ..., n\}$.

2. Элементы B_j:



Причем, круги могут располагаться на любых входах, на один вход не более одного круга, «закрашенный» круг – только один.

Функционирование схемы. Пусть T – класс обратимых функций, построенных следующим образом:

$$\begin{aligned} T_1^n(x_i):\ & (x_1, ..., x_i, ..., x_n) \to (x_1, ..., \bar{x}_i, ..., x_n), \\ & \forall i \in \{1, ..., n\} \\ T_{k+1}^n(x_{i_1}, ..., x_{i_k}, x_j):\ & (x_1, ..., x_j, ..., x_n) \to (x_1, ..., x_j \oplus x_{i_1} \cdot ... \cdot x_{i_k}, ..., x_n), \\ & k < n,\ \{i_1, ..., i_k\} \subset \{1, ..., n\} \\ & \forall j \in \{1, ..., n\},\ j \notin \{i_1, ..., i_k\} \end{aligned}$$

Функции T_1^n класса T представляют собой одноместные отрицания N, функции $T_2^n, \ldots, T_n^n$ называются функциями Тоффоли [1].

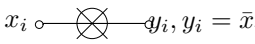
Класс обратимых функций T является базисом для множества обратимых функций [6].

В дальнейшем условимся, там где необходимо, в изображениях логических схем опускать входы «незанятые» кругами.

Функционирование логической схемы:

– тривиальная логическая схема реализует тождественную функцию $I:\ \{\tilde{\alpha}\} \to \{\tilde{\alpha}\}$, по всем наборам $\tilde{\alpha}$;

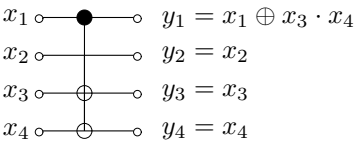
– каждому элементу N_i сопоставим функцию Тоффоли $T_1^n(x_i)$ так, что вход x_i преобразуется в выход $y_i = \bar{x}_i$, а входы x_l $(l \neq i)$, преобразуются в выходы $y_l = x_l$;



– каждому вентилю B_j сопоставим функцию Тоффоли $T_{k+1}^n(x_{i_1}, ..., x_{i_k}, x_j)$ так, что вход x_j преобразуется в выход y_j (закрашенный круг)

$$y_j = x_j \oplus x_{i_1} \cdot ... \cdot x_{i_k},$$

где $x_{i_1}, ..., x_{i_k}$ обозначают те входы, на которые помещены незакрашенные круги; входы x_l $(l \neq j)$, преобразуются в выходы $y_l = x_l$. Ниже приведен пример логической схемы на 4 входах и выходах, содержащей элемент B_1, которому соответствует функция Тоффоли $T_3^4(x_3, x_4, x_1):\ (x_1, x_2, x_3, x_4) \to (x_1 \oplus x_3 \cdot x_4, x_2, x_3, x_4)$.



В дальнейшем изложении, элементы N_i и B_j будем называть элементами Тоффоли T_k^n.

Для удобства записи введем обозначение $\tilde{x} = (x_1, ..., x_n)$. Пусть даны две обратимые функции $f(\tilde{x}) = (f_1(\tilde{x}), ..., f_n(\tilde{x}))$, $g(\tilde{x}) = (g_1(\tilde{x}), ..., g_n(\tilde{x}))$. Тогда функция

$$h(\tilde{x}) = g(f(\tilde{x})) = (g_1(f_1(\tilde{x}), ..., f_n(\tilde{x})), ..., g_n(f_1(\tilde{x}), ..., f_n(\tilde{x})))$$

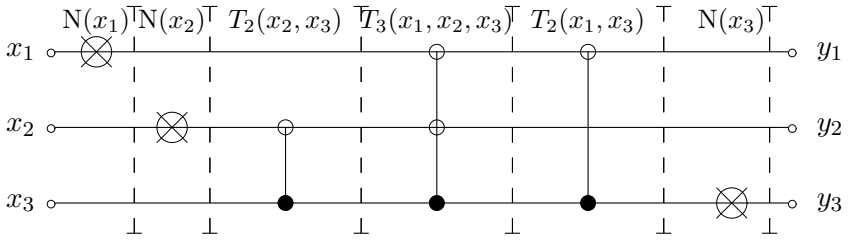
называется суперпозицией функций f и g.

Пусть дана схема S_1 на n входах и выходах, содержащая один элемент Тоффоли T_{k+1}^n, $(k \leq n-1)$, и схема S_2, также, на n входах и выходах, содержащая один элемент Тоффоли T_{m+1}^n, $(m \leq n-1)$. Тогда логическая схема S_3 является результатом склейки схем S_1 и S_2, если выходы $y_1, \ldots, y_n$ схемы S_1 являются входами $x_1, \ldots, x_n$ схемы S_2, причем $x_i = y_i$ по всем $i \in \{1, 2, \ldots, n\}$. Склейке схем соответствует применение операции суперпозиции над функциями T_{k+1}^n и T_{m+1}^n. Схема S_3 реализует обратимую функцию $f_3(\tilde{x}) = T_{m+1}^n(T_{k+1}^n(\tilde{x}))$. Естественно, обратимая функция может быть реализована различным схемами.

Схемы называют обратимыми, если входные значения однозначно восстанавливаются по значениям на выходах. Легко видеть, что схемы в базисе Тоффоли будут обратимыми.

Пример обратимой схемы функции f:

$$f(x_1,\ x_2,\ x_3) = (\bar{x}_1,\ \bar{x}_2,\ \bar{x}_2 \oplus \bar{x}_1\bar{x}_2 \oplus \bar{x}_1 \oplus 1)$$



Отметим, что операции суперпозиции между функциями f и g соответствует, также, произведение перестановок P_f и P_g, произведение перестановочных матриц M_f и M_g.

Сложность обратимой функции

Пусть S – обратимая схема с n входами. Тогда сложностью $L(S)$ схемы S будет называться количество элементов схемы. Обратимая схема $S^{'}$ называется эквивалентной данной схеме S, если обе схемы реализуют одну и ту же обратимую функцию. Обратимая схема называется минимальной, если не существует эквивалентной ей схемы меньшей сложности.

Сложность $L_K(f)$ функции f в классе K обратимых схем определяется так:

$$L_K(f) = min_{S \in K} L(S).$$

Сложность $L_K(n)$ класса F_n всех обратимых функций в классе обратимых схем K определяется следующим образом

$$L_K(n) = max_{f \in F_n} L_K(f).$$

Нижнюю оценку на $L_K(n)$ можно получить мощностным методом. Поскольку любая обратимая функция представляется перестановкой 2^n чисел, легко вычислить общее количество обратимых функций – $|F_n| = 2^n!$. Легко показать, что для T_i^n элементов Тоффоли будет выполняться $|T_i^n| = n2^{n-1}$. Тогда количество обратимых схем сложности k на n входах не больше, чем $(n2^{n-1})^k$. Следовательно, для реализации всех функций из F_n должно выполняться неравенство:

$$(n2^{n-1})^k > 2^n!.$$

Логарфмируя получившееся неравенство и воспользовавшись оценкой Стирлинга $\log_2 k! \approx k \log_2 k$, получим:

$$k \log_2 n + (n-1)k > 2^n n.$$

Отсюда,

$$k > \frac{2^n n}{\log_2 n + n - 1}.$$

В [7] получена верхняя оценка $k < \frac{n2^n}{\log_2 n}$. В итоге,

$$\frac{2^n n}{\log_2 n + n - 1} < k < \frac{n2^n}{\log_2 n}.$$

Точное значение для $L_K(n)$ неизвестно, а полученные неравенства дают слишком большой интервал для сложности $L_K(n)$. Так при $n = 4$ интервал сложностей имеет размеры от 13 до 32.

Всвязи с трудностями решения задачи нахождения $L_K(n)$ возникает интерес к решению задачи нахождения $L_K(f)$. В дальнейшем изложении индекс K будет опускаться, поскольку все реализации функций рассматриваются только в классе обратимых схем.

Алгоритм 1 вычисления сложности обратимой функции

Пусть $f(x_1, ..., x_n)$ – обратимая функция, сложность которой $L(f)$ требуется найти, M_f – перестановочная матрица функции, S_f – минимальная обратимая схема, соответствующая функции. В этом случае выполняется $L(f) = L(S_f)$.

Существуют обратимые функции f_1 и f_2 с минимальными схемами S_{f_1} и S_{f_2}, соответственно, такие что функции $f = f_1(f_2)$ соответствует минимальная схема S_f. В матричном представлении

$$M_f = M_{f_1} \cdot M_{f_2}.$$

Предположим, $L(S_{f_1}) = k$. Тогда для вычисления сложности $L(f)$ требуется перебрать все сформированные предварительно схемы сложности k в качестве S'_{f_1}, соответствующей $(M_{f_1})^{-1}$, в разложении

$$M_{f_2} = (M_{f_1})^{-1} \cdot M_f$$

и выбрать минимальную для f_2.

Таким образом, в данном алгоритме требуется предварительно сформировать библиотеку минимальных схем до сложности k включительно. Вычисляемая в данном алгоритме сложность $L(f)$, находится в диапазоне от $k + 1$ до $2k$, это значение является существенным для небольших n.

Итак, пусть $X = S_f$ – минимальная обратимая схема функции f, сложность которой требуется вычислить. `nt` – библиотеки схем сложности 0, 1, ..., $k-1$, `nt_k` – библиотека схем сложности k.

Запись алгоритма вычисления S_f.

```
integer cost_circuit_2k(nt, nt_k, X) {
    cost, c; // текущая и промежуточная сложности схемы X
    l_nt_k; // число схем сложности k,
    A, y; // схемы;
  cost ← k+1;
  for j ← 0 to l_nt_k do {
    A=product(X, nt_k[j]); // склейка схем;
    y=Search(nt, A); // поиск подсхемы A в библиотеке схем;
    if(y!=nt[last]+1) c ← get_cost(y); // найдена подсхема y,
    получить ее сложность;
    if(c<cost) cost ←  c; }
   return k+cost; }
```

Реализация алгоритма 1 позволила сформировать библиотеку минимальных схем до сложности 8, включительно. Ее объем составил примерно 3,7Gb. Однако переход на большие сложности потребовал сокращения времени работы программы, что привело к параллельной версии алгоритма 1.

Пусть *size* – количество узлов кластера(процессов программы).

1) *Процесс 0* считывает из файла и рассылает, разбитую на порции, библиотеку `nt_k` на процессы i, где $0 < i < size$; процесс 0 считывает и себе порцию.

   Процесс i получает порцию библиотеки `nt_k`.

2) *Все процессы* считывают библиотеку `nt`.

   Процесс 0 получает обратимую функцию X, сформированную пользователем или случайно, проверяет ее на наличие в библиотеке `nt`, если ее там нет, рассылает процессам i.

   Все процессы получают X, либо завершают работу.

   Все процессы применяют подпрограмму `cost_circuit_2k(nt, nt_k, X)` к своей порции библиотеки `nt_k` и отправляют вычисленную сложность процессу 0.

3) *Процесс* 0 получает значения сложности функции X от процессов и выбирает минимальное.

```
integer cost_circuit_2k_P(nt, nt_k, X){
    myrank; // номер процесса,
    l_nt; // размер порции библиотеки nt,
```

```
  l_nt_k; // размер библиотеки nt_k,
  A; // схема,
  Cost; // текущее значение сложности;
  costs: array;
if(myrank==0){ // процесс 0;
  ifstream file(nt_k.txt); // открыть файл с библиотекой nt_k;
  for i ← 1 to size-1 do {   read(file, nt_k, l_nt_k);
    // считать порцию библиотеки схем;
    send(nt_k, l_nt_k, i); } // отправить порцию на процесс i;
  read(file, nt_k, l_nt_k);}
else recv(nt_k, l_nt_k, 0); // все процессы, кроме процесса 0,
получают свою порцию библиотеки nt_k;
ifstream file2(nt.txt); read(file, nt, l_nt); Cost ←  k+1;
if(myrank==0) {
  gen(X); // сгенерировать случайно функцию;
// проверить, нет ли уже схемы X в библиотеке;
  Cost← Check_cost(Cost, nt, X);  }
Bcast(Cost); // разослать сложность Cost на все процессы;
if(Cost<k+1) return Cost; // схема X есть в библиотеке;
else {  Bcast(X); // разослать схему  на все процессы
  Cost ← cost_circuit_2k(nt, nt_k, X); // применить предыду-
щий алгоритм на порции библиотеки nt_k;
  if(myrank!=0) send(Cost, 1, 0);
  else{ costs[0] ← Cost; // процесс 0 собирает сложности;
    for i ← 1 to size-1 do recv(costs+i,1,i);
    return min_element(costs); } } }
```

Параллельный алгоритм был реализован на языке программирования C++ с использованием mpi-библиотеки и запущен в работу на кластере ВСГАО с 16 узлами. Существенное сокращение времени работы параллельного алгоритма связано с тем, что на различных процессах происходит одновременное выполнение двух самых трудоемких операций, это произведение схем (`A=product(X, nt_k[j])`) и поиск в библиотеке (`Search(nt, A)`).

Статистические данные работы алгоритма на классе из 1140 функций, предположительно сложности больше 9.

Сложность	9	10	11	12	13	14	15	16
Количество	0	2	4	42	183	431	409	69

Алгоритм синтеза библиотеки сложностей

Для работы алгоритма 1 требуется сформировать библиотеку минимальных схем до некоторой сложности k. В соответствующем алгоритме синтеза значение k зависит, с одной стороны, от n – чем больше n, тем

быстрее растет количество схем с ростом их сложности. Отсюда, увеличивается потребность в емкости дискового пространства для хранения этих схем. С другой стороны, k зависит от мощности вычислительной системы, на которой будет запускаться данный алгоритм.

Пусть X – схема текущей сложности i, M_X – перестановочная матрица соответствующей обратимой функции.

j-й шаг алгоритма:

1) Вычислить произведение $M_Y = M_X \cdot M_{T_i}$ матриц M_X и M_T, где T – элемент Тоффоли,

2) Проверить схему Y, соответствующую матрице M_Y, на наличие в библиотеке схем `nt`,

3) Если схемы нет, добавить Y в библиотеку.

Повторять j-й шаг для всех элементов класса T и для всех схем сложности i. Причем, при проверки схемы Y в библиотеке достаточно ограничиться схемами сложности $i-2$, $i-1$, i.

Формальная запись алгоритма синтеза

```
void Synthesis(){
     basis, nt: array; // базис Тоффоли и библиотека схем,
     l; // число базисных элементов,
     Cost; // текущая сложность,
     Count_; // число схем текущей сложности,
     Count_all; // текущий размер библиотеки схем,
     X; // схема;
   nt[0] ← =e;// схема сложности 0,
   e – число, соответствующее единичной матрице,
   при n=4 e=0x123456789abcdef;
   gen_basis(basis); // формирование базиса;
   nt← basis; // схемы сложности 1 – базисные элементы;
   Cost←1; Count_ ← l; Count_all ← l+1;
   while(Count_all!=2^n!){
     i1 ← Count_all-Count_; i2 ← Count_; Cost++;Count_  ← 0;
     for i ← i1 to i2 do {// по всем схемам сложности Cost-1;
       for j ← 0 to l-1 do{// по всем базисным элементам;
         X ← product(nt[i], basis[j]); // новая схема;
         x ← Search(Cost, nt, X);
         // поиск среди схем сложности Cost-2, Cost-1, Cost;
         if(x==0){ nt[Count_all]←X; Count_all++; Count_++}
         // пополнение библиотеки;
} } } }
```

Ниже в таблице представлено количество схем по сложностям и время работы программы.

Сложность	Количество	Время (сек)
0	1	
1	20	
2	316	
3	4190	
4	49827	
5	545477	
6	5530949	50
7	51730709	100
8	444320918	852

В ходе работы программы алгоритма синтеза в оперативную память компьютера помещается отсортированная библиотека схем сложности $i-2, i-1$ и сформированные схемы сложности i. Далее осуществляется поиск текущей схемы в указанных массивах. Причем поиск среди схем сложности $i-2, i-1$ – двоичный, его сложность приблизительно равна $\log_2(N_1)$, где N_1 – число схем. Поиск среди схем текущей сложности i – последовательный. Учитывая рост числа схем с ростом сложности i, последовательный поиск существенно замедляет работу алгоритма. Поэтому алгоритм синтеза для конструирования схем следующей сложности $(k+1)$ был распараллелен.

1) Процесс 0:

   а) Считывает из файла по частям библиотеку схем сложности $k-1,\ k$ (массив `nt_`) и отправляет эти части на процессы $1, 2, ..., size-2$, где $size$ – число процессов.

   б) Формирует часть схем сложности $k+1$ (`nt_k1`), сортирует, убирает повторы и отправляет процессу 1.

2) Процесс i : $(i = 1, 2, ..., size-2)$

   а) Получает свою часть библиотеки схем сложности $k-1,\ k$ от процесса 0.

   б) Получает часть схем сложности $k+1$ (`nt_k1`) от процесса $i-1$ и сверяет каждую схему со схемами из библиотеки.

   в) Отправляет схемы, прошедшие проверку, процессу $i+1$.

3) Процесс $size-1$:

а) Получает части схем сложности $k+1$ от процесса $size-2$, накапливая их в одном массиве, `nt_k1`, до определенного предела, который зависит от объема оперативной памяти узла.

б) Сортирует и удаляет повторы в массиве `nt_k1`, затем записывает его в файл.

```
void Synth_parallel(){
     nt_, nt_k, nt_k1: array; // библиотеки
     k − 1, k, сложности k, сложности k + 1,
     l; // размер порции схем;
   if(myrank==0){
     ifstream file(nt_.txt); // открыть файл с библиотекой;
     for(i ← 1 to size-1 do {
       read(file, nt_, l); // считать порцию схем из файла;
       send(nt_, l, i); } // отправить порцию процессу i;
     ifstream file2(nt_k.txt); // открыть файл с библиотекой;
     for(j ← 1 to limit do { read(file2, nt_k, l);
       gener(nt_k1, nt_k, l); // сформировать схемы сложности
       k + 1 в массив nt_k1;
       sort(nt_k1, l); unique(nt_k1); erase(nt_k1);
       // отсотритовать и удалить повторяющиеся схемы в массиве;
       send(nt_k1, l, 1); } // отправить порцию nt_k1 процессу l;
   else if((myrank > 0) and (myrank < size-1)){
     recv(nt_, l, 0); // получить порцию библиотеки схем
     сложности k − 1, k от процесса 0;
     for(j=1 ←  to limit do {
     //limit – количество порций сформированных схем;
       recv(nt_k1, l, myrank-1); // получить порцию
       сформированных схем сложности k + 1 от процесса myrank − 1;
       check(nt_k1, nt_); // убрать схемы порции nt_k1,
       встречающиеся среди схем сложности k − 1, k;
       send(nt_k1, l, myrank+1); } // отправить порцию nt_k1
       следующему процессу myrank + 1;
   else{ // процесс size − 1;
     n←0; ofstream file3(nt_k1.txt);
     for(j ← 1 to limit do {
       recv(nt_k1, l, myrank-1); n++;
       if(n==f_limit) {
       // f_limit – количество накапливаемых порций;
         sort(nt_k1, l); unique(nt_k1); erase(nt_k1);
         write(file3, nt_k1, l); n←0; } } } } // запись в файл;
```

Параллельный алгоритм позволил сформировать библиотеку схем сложности 9.

Замечания о работе алгоритмов

1) *Перевод матриц в векторы.* В выше предложенных алгоритмах наиболее часто используемая операция – это операция суперпозиции обратимых функций, которой соответствует произведение перестановочных матриц, произведение перестановок и склейка обратимых схем, реализующих функции.

   Пусть перестановке $P_F = \begin{pmatrix} 0 & 1 & 2 & \dots & 2^n-1 \\ i_0 & i_1 & i_2 & \dots & i_{2^n-1} \end{pmatrix}$ обратимой функции F соответствует последовательность $(a_0, a_1, ..., a_{2^n-1})$ длины 2^n, где $a_j = i_j$ по всем $0 \le j \le 2^n - 1$.

2) Пусть даны последовательности $(a_0, a_1, ..., a_{2^n-1})$, $(b_0, b_1, ..., b_{2^n-1})$, соответствующие перестановкам P_F и P_G. Тогда произведению перестановок $P_G P_F$ соответствует произведение последовательностей $(c_0, c_1, ..., c_{2^n-1})$, где
$$c_i = b_{a_i}$$

3) Для введения естественного порядка в библиотеку обратимых схем используется *факториальная система счисления* [1].

   По определению число $X = \alpha_N \alpha_{N-1} ... \alpha_1$ записано в факториальной системе счисления, если
$$X = \alpha_1 \cdot 1! + \alpha_2 \cdot 2! + \ ... \ + \alpha_N \cdot N!, \ \alpha_i \le i.$$

   Каждой последовательности $(a_0, a_1, \ ... \ , a_{2^n-1})$ длины 2^n ставится в соответствие число в факториальной системе $A = \alpha_0 \alpha_1 \ ... \ \alpha_{2^n-2}$, которое вычисляется так:

   α_i — число цифр, меньших a_i и стоящих в последовательности справа от него.

   Перевод в десятичную систему счисления. Далее, предварительно вычислив факториалы $1!, \ 2!, \ ..., (2^n - 1)!$, число X, записанное в факториальной системе счисления переводится в десятичную. Пусть это будет число Y.

   В силу того, что в последовательности $(a_0, a_1, \ ... \ , a_{2^n-1})$ нет повторяющихся цифр, введенное соответствие является однозначным, поэтому позволяет задать для каждой последовательности свой индивидуальный номер Y. По этому номеру легко восстановить последовательность и значит перестановку P_F соответствующей функции F. В выше предложенном алгоритме 1 обратимая функция F, сложность которой вычисляется, задается через номер Y_F соответствующей перестановки P_F.

4) *Перевод последовательности в шестнадцатеричную систему счисления.*

Пусть обратимой функции F, зависящей от 4-х переменных, соответствует последовательность $(a_0, a_1, \ \dots\ , a_{15})$. Сопоставим последовательности $(a_0, a_1, \ \dots\ , a_{15})$ число Z, записанное в шестнадцатеричной системе счисления:

$$Z = a_0 a_1 ... a_{15}.$$

Тогда, учитывая диапазон значений элементов a_i последовательности $(a_0, a_1, \ \dots\ , a_{15})$, для хранения числа Z требуется 64 бита, а не 256 бит, как для хранения последовательности.

Еще одно преимущество перехода в шестнадцатеричную систему счисления заключается в том, что теперь произведение последовательностей заменяется на произведение чисел, в котором используются логические операции и операции побитового сдвига (эти операции существенно экономят время работы алгоритмов).

5) В числе Z последняя цифра a_{15} заменим на значение сложности $L(S)$ соответствующей минимальной обратимой схемы S. Теперь

$$Z = a_0 a_1 ... a_{14} L(S)$$

(в алгоритме $L(S)$ не превышает 9). Восстанавливается a_{15} через сложение по модулю 2 остальных цифр: $a_{15} = \sum_{i=0}^{14} a_i$.

Алгоритм 2 вычисления сложности обратимой функции

В задаче вычисления сложности обратимой функции f рассмотрим случай, когда $L(f) > 2 \cdot k$. Здесь возможны два пути изменения алгоритма и оба эти пути приводят к получению приближенного алгоритма. Первый путь аналогичен алгоритму 1, только здесь схема X разбивается не на две, а на три подсхемы и осуществляется перебор 2-х подсхем. Однако, в этом случае сложность $L(f)$ будет ограничена значением $3k$, с одной стороны. С другой стороны, даже при $n = 4$ невозможно осуществить полный перебор подсхем.

Пусть P_f – перестановка обратимой функции f.

1 шаг. P_f разбивается на циклы:

$$P_F = A_1 A_2 ... A_k.$$

Все циклы A_j разбиваются на произведение транспозиций.
Пусть $A_j = (a_1^j, a_2^j, ..., a_l^j)$, тогда легко проверить, что

$$A_j = (a_1^j, a_2^j)(a_1^j, a_3^j)...(a_1^j, a_l^j).$$

2 шаг. Итак, $P_f = T_0T_1...T_s$, где T_i – транспозиции. Вычислить суммарную сложность $Cost$ минимальных схем, соответствующих транспозициям T_i.

3 шаг. Вычислить произведения транспозиций:

$$TP_0 = T_0 \cdot T_1,\ TP_1 = T_1 \cdot T_2,\ ...TP_{s-1} = T_{s-1} \cdot T_s$$

4 шаг. Вычислить сложности c_i минимальных схем, соответствующих произведениям TP_i и выбрать среди них наименьшую сложность c. Пусть сложности c соответствует произведение $TP_i = T_i \cdot T_{i+1}$. Если $c > 2 \cdot k$, возвратить $Cost$ и выйти из алгоритма.

5 шаг. Заменить T_i на TP_i; удалить T_{i+1} и пересчитать TP_{i-1} и TP_i.

6 шаг. Если суммарная сложность $Cost = k$ или, если среди произведений TP_i осталось одно, то выйти и возвратить $Cost$ иначе вернуться к шагу 4.

```
void Tr_cost(T, c_T, nt, nt8, X) {
  get_transpositions(T, c_T, X);// разложить перестановку X в
  транспозиции T и вычислить их сложности c_T;
  j ← 0; while(j<last) { // по всем элементам массива T,
  кроме последнего;
    TP[j] ← product(T[j], T[j+1]);
    c[j] ← Cost_circuit(nt, nt8, TP[j]);  j++; }
  while(TP.size()!=0) {
    m=min_element(c); // возвращает указатель на минимальный
    элемент в массиве c;
    if(*m<17) { // заменить транспозицию на соответствующую
    подсхему;
      T[m-T.begin()] ← TP[m-TP.begin()];
      c_T[m-c_T.begin()] ← c[m-c.begin()];
      delete(T, c_T, m+1); // удалить лишние;
      recount_neighb(TP, c); // пересчитать соседние элементы в
      массивах TP и c;
    } else break; } }
```

В заключении работы алгоритма 2 массив T содержит минимальные подсхемы схемы обратимой функции f. В данном алгоритме при вычислении элементов массивов c и c_T используется последовательный алгоритм 1.

Список литературы

1. Гашков С. Б. Системы счисления и их применение / С. Б. Гашков. – М. : МЦНМО, 2004. – 52 с.: ил. – (Б-ка «Математическое просвещение») ; вып. 29).

2. Квантовый компьютер и квантовые вычисления. Т. 2. – Ижевск : Ред. журн. «Регулярная и хаотическая динамика», 1999. – 287 с.
3. Мальцев А. И. Основы линейной алгебры / А. И. Мальцев. – М. : Наука, 1970. – 400 с.
4. Monz T. [et al.] Realization of the quantum Toffoli gate with trapped ions. arXiv:0804.0082v1 [quant-ph] 1 Apr 2008.
5. Toffoli T. Reversible Computing. Tech.Memo MIT/LCS/TM-151, MIT Lab. for Comp.Sci. – 1980.
6. Toffoli T. Bicontinuous Extensions of Invetible Combinatorial Functions // Mathematical Systems Theory. – 1981. – Vol. 14. – P. 13-23.
7. Vivek V. Shende at al. Synthesis of Reversible Logic Circuits //IEEE Transactions on Computer-Aided design of integrated circuits and systems.– 2003. – Vol. 22, N6.

S. F. Vinokurov, A. S. Frantseva

An approximate algorithm for computing the complexity of reversible functions in the basis of Toffoli

Abstract.

We investigate questions of computing of the complexity of the arbitrarily given reversible function. Previously introduced the concept of reversible computation required, in particular, the concept of a reversible function. We consider various representations of reversible functions (the table of values, permutation matrices, permutations and logic circuits). As the basis of reversible circuits is considered a well-known Toffoli gates[1]. Presented sequential and parallel algorithms for computing of the complexity of the reversible function. The algorithms were run on a cluster containing 16 nodes, each of which has a Intel Pentium Dual CPU, 1,86 Gz, 1,87 Gz, 2Gb, Ethernet network 100Mbit/sek. The cluster is organized in ESSAE. We present estimates of the complexity.

Keywords: reversible functions, complexity, Toffoli gates, sequential and parallel algorithms.

Винокуров Сергей Федорович, доктор физико-математических наук, профессор, Восточно-Сибирская государственная академия образования, 664011, Иркутск, ул. Н. Набережная, 6 тел.: (3952) 240435 (servin38@gmail.com)

Францева Анастасия Сергеевна, аспирант кафедры математической информатики Восточно-Сибирская государственная академия образования, 664011, Иркутск, ул. Н. Набережная, 6 тел.: (3952) 240435 (a.s.frantseva@gmail.com)

Vinokurov Sergey, East Siberian State Academy of Education, 6, N. Naberezhnaya St., Irkutsk, 664011, professor, phone: (3952) 240435 (servin38@gmail.com)

Frantseva Anastasiya East Siberian State Academy of Education, 6, N. Naberezhnaya St., Irkutsk, 664011, postgraduate, phone: (3952) 240435 (a.s.frantseva@gmail.com)