

ЛИТЕРАТУРА

1. *Зубков А. М.* Рекуррентные формулы для распределений функционалов от дискретных случайных величин // *Обозр. прикл. промышл. математики.* 1996. Т. 3. Вып. 4. С. 567–573.
2. *Зубков А. М.* Методы расчета распределений сумм случайных величин // *Труды по дискретной математике.* М.: Физматлит, 2002. Т. 5. С. 51–60.
3. *Зубков А. М.* Вычисление распределений чисел компонент и циклических точек случайного отображения // *Математические вопросы криптографии.* 2011. Т. 1. № 2. С. 5–18.
4. *Filina M. V. and Zubkov A. M.* Exact computation of Pearson statistics distribution and some experimental results // *Austrian J. Statistics.* 2008. V. 37. No. 1. P. 129–135.
5. *Filina M. V. and Zubkov A. M.* Tail properties of Pearson statistics distributions. // *Austrian J. Statistics.* 2011. V. 40. No. 1&2. P. 47–54.

УДК 004.432.2

ВЫЧИСЛЕНИЕ СОВЕРШЕННОЙ НЕЛИНЕЙНОСТИ БУЛЕВОЙ ФУНКЦИИ НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ

А. В. Медведев

Для криптографических приложений представляют интерес функции с высокими показателями «нелинейности», потому что они труднее поддаются анализу. Существует несколько характеристик нелинейности функции, одна из них — совершенная нелинейность [1], которая показывает, насколько функция удалена от класса функций с линейной структурой. Разработан алгоритм вычисления совершенной нелинейности произвольной булевой функции для параллельной реализации на видеокартах NVIDIA, поддерживающих технологию CUDA [2].

Обозначим через $P_2(n)$ множество всех булевых функций от $n \geq 1$ переменных и X — область их определения, $X = \{0, 1\}^n$.

Определение 1. Для функции $f \in P_2(n)$ и набора $a \in X$ функция $f'_a(x) = f(x) \oplus f(x \oplus a)$ называется *производной функции f по направлению a* .

Определение 2. Говорят, что булева функция f имеет линейную структуру, если существует вектор $a \in X \setminus \{0^n\}$, что $f'_a = \text{const}$, т. е. $f = f_a$ либо $\neg f = f_a$. Множество всех функций в $P_2(n)$, имеющих линейную структуру, обозначается $LS(n)$.

Определение 3. Число $CN_f = d(f, LS(n)) = \min_{g \in LS(n)} d(f, g)$ называется *совершенной нелинейностью* функции f . Здесь $d(f, g)$ — расстояние между функциями f и g , равное количеству наборов, на которых они различаются.

Существует ряд способов вычисления CN_f ; наиболее подходящим для параллельной реализации оказался следующий:

$$CN_f = \min_{a \in X \setminus \{0^n\}} \min(w(f'_a), 2^n - w(f'_a))/2,$$

где $w(\cdot)$ — вес булевой функции. Основную трудность здесь представляет вычисление функции $f(x \oplus a)$. Проблема состоит в том, что CUDA предполагает запуск большого числа нитей параллельно [3], но их совокупная память ограничена (3 Гб для Tesla C2050), поэтому невозможно хранить значения $f(x \oplus a)$ для каждой нити, и вес производной вычисляется «по частям» (алгоритм 1).

Вектор значений булевой функции от n переменных представляется в памяти массивом 2^N -битных ячеек — элементов базового типа языка C/C++ (в реализации используется 32-битный тип `long int`, $N = 5$). Количество ячеек определяется по формуле $m = 2^{n-N}$. Далее будем обозначать i -ю ячейку функции f через f_i .

Алгоритм 1. Алгоритм вычисления $w(f'_a)$

Вход: f, a

Выход: $w(f'_a)$

$w \leftarrow 0$

Для $i = 0, \dots, m - 1$

$w \leftarrow w + \text{weight}(f_i(x) \oplus f_i(x \oplus a))$

Вернуть $w = w(f'_a)$

Данный алгоритм не требует вычисления $f(x \oplus a)$ в явном виде, но надо уметь получать i -ю ячейку функции $f(x \oplus a)$. Заметим, что вектор значений $f(x \oplus a)$ получается из вектора значений функции $f(x)$ перестановкой бит по определённому закону. Например, если $a_1 = 1$, то меняются местами половины вектора значений исходной функции, иначе ничего не меняется; если $a_2 = 1$, то меняются между собой половины внутри половин всего вектора, иначе они не меняются, и т. д. рекурсивно.

Алгоритм 2 принимает на вход вектор a и номер i , а на выходе выдает номер j ячейки $f(x)$, который соответствует ячейке с номером i в $f(x \oplus a)$.

Алгоритм 2. Алгоритм вычисления номера ячейки для функции $f(x \oplus a) \in P2(n)$

Вход: $i, a = a_1 \dots a_n, m$ — количество ячеек

Выход: j , такой, что $f_i(x) = f_j(x \oplus a)$ (без учета перестановки бит внутри ячейки)

$l \leftarrow 0$ // инициализация левой границы интервала

$j \leftarrow i$

$iw \leftarrow m \gg 1$ // половина длины текущего интервала

Для $k = 1, \dots, n$

Если $i < l + iw$, **то**

Если $a_k = 1$, **то**

$j \leftarrow j + iw$

иначе

$l \leftarrow l + iw$

Если $a_k = 1$, **то**

$j \leftarrow j - iw$

$iw \leftarrow iw \gg 1$

Конец цикла.

Вернуть j

Перестановка бит внутри ячеек происходит по такому же закону, но в реализации используется более быстрый способ на основе битовых операций.

Ниже приведены результаты экспериментов по вычислению CN_f при разной степени распараллеливания. Испытания проводились на компьютере Intel(R) Core(TM)2 Quad Q6700 2,66 ГГц с установленной видеокартой NVIDIA Tesla C2050. На CPU алгоритм реализован в двух вариантах — последовательном и с использованием библиотеки

OpenMP, которая позволяет задействовать все ядра центрального процессора (в нашем случае 4 ядра). В таблице показано время работы (в секундах) соответствующих реализаций.

n	CPU		GPU
	Посл. алг.	OpenMP	CUDA
15	1,02	0,25	0,01
16	4,13	1,03	0,04
17	16,9	4,22	0,15
18	69,9	17,4	0,65

Таким образом, реализация для видеокарты с использованием технологии CUDA оказалась быстрее, чем последовательный алгоритм, примерно в 106 раз, а в сравнении с параллельной реализацией на центральном процессоре — примерно в 26,5 раз.

ЛИТЕРАТУРА

1. Агibalов Г. П. Избранные теоремы начального курса криптографии. Томск: Изд-во НТЛ, 2005.
2. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
3. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_BestPractices.pdf

УДК 519.7

О СХОДИМОСТИ ГИБРИДНОГО SAT+ROBDD-ЛОГИЧЕСКОГО ВЫВОДА¹

А. А. Семенов, А. С. Игнатьев

Проблема обращения дискретных функций возникает во многих теоретических и прикладных областях современной кибернетики. Пусть $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ — вычисляемая детерминированным образом за полиномиальное от длины входа время дискретная функция. Если $A(f)$ — некоторый полиномиальный алгоритм, вычисляющий f , то $A(f)$ задает семейство функций вида $f_n : \{0, 1\}^n \rightarrow \{0, 1\}^*$, $n \in N$. Задача обращения произвольной функции f_n из данного семейства состоит в следующем: известно $y \in \text{Range } f_n$, требуется, зная текст программы $A(f)$, найти произвольный $x \in \{0, 1\}^n$, такой, что $f_n(x) = y$. Описанная задача является вычислительно трудной в общей постановке — она не может быть решена за полиномиальное время в предположении, что $P \neq NP$. Поскольку различные практические задачи могут рассматриваться как частные случаи сформулированной, разработка вычислительных алгоритмов для её решения является актуальной областью.

Описанную проблему можно сводить к SAT-задачам [1], используя эффективные алгоритмы трансляции программ в булевы уравнения (см., например, [2]). Для решения получаемых SAT-задач можно использовать различные методы, лучшие из которых базируются на алгоритме DPLL. В КНФ, получаемой в процессе трансляции алгоритма $A(f_n)$, можно выделить множество X_n , состоящее из n так называемых «переменных входа» рассматриваемой функции. Мощность данного множества равна n . В [3] говорится о том, что X_n является «сильной формой» множества-лазейки для алгоритма DPLL, который применяется к КНФ $C(f_n)$, кодирующей задачу обращения f_n

¹Работа выполнена при поддержке гранта РФФИ № 11-07-00377-а.