Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Сравнение алгоритмов обращения элементов поля $GF(2^N)$

Курсовая работа студента 344 группы

Щербакова Александра Владимировича

Научный руководитель **Платонов С.М.**

(Руководитель исследовательской лаборатории RAIDIX)

Санкт-Петербург

2013

Оглавление

Введение

Во многих системах хранения данных, использующих технологию RAID, расчёт контрольных сумм производится кодами Рида-Соломона с помощью арифметики конечных полей Галуа $GF(2^N)$. Использование этих полей удобно, так как позволяет работать при кодировании со словами-векторами из 0 и 1.

Расчёта контрольных сумм производится для того, чтобы можно было восстановить данные при отказе одного или нескольких дисков. В такой операции есть необходимость производить деление в рабочем поле Галуа. Обычно, используется лишь 2 контрольные суммы, и количество необходимых обратных элементов достаточно мало, чтобы быть рассчитанным заранее и использоваться в последствие при расчётах.

Однако, с ростом размера поля Галуа, количества дисков в RAID-массиве и дисков с контрольными суммами хранение таблиц обратных элементов становится невозможным. Дело в том, что в общем случае пришлось бы хранить таблицу обратных для всех элементов поля, а это (для $GF(2^N)$, в котором каждый элемент можно однозначно представить -битным двоичным числом) составляет:

$$N * 2^N \text{ бит.}$$

При $N = 32$ получаем 16 Гб, что тяжело хранить в памяти. Значит, нужен эффективный способ, позволяющий вычислять обратный элемент в $GF(2\hat{}N)$.

Постановка задачи

Дано: $GF(2^N)$ (точнее, многочлен f, по модулю которого $GF(2^N)$ построено), а также $a \in GF(2^N)\backslash\{0\}$. Нужно найти $a^{-1} \in GF(2^N)\backslash\{0\}$ – такой элемент, что

$$a * a^{-1} = 1 \in GF(2^N).$$

В рамках данной курсовой работы мне было необходимо сравнить и реализовать ряд алгоритмов обращения элементов $GF(2\hat{}N)$.

Определения

$\boldsymbol{GF(2^N)}$ – это поле, построенное из кольца многочленов $Z_2[x]$ по модулю неприводимого в $Z_2[x]$ многочлена f вида:

$$f(x) = x^N + p(x), \deg p < N.$$

Тогда, для $\forall a \in GF(2^N)\backslash\{0\}$ существует ***обратный элемент*** $a\hat{}(-1) \in GF(2^N)\backslash\{0\}$, такой что:

$$a *_{GF(2^N)} a^{-1} = 1.$$

Так как элементы $GF(2^N)$ – это полиномы степени не выше $N-1$ с коэффициентами из $\{0,1\}$, их удобно представлять в виде двоичных слов длины N. При этом ***сложение*** в $GF(2^N)$ – это операция $\oplus$ (XOR), а ***умножение*** $(*_{GF(2^N)})$ двух элементов можно свести к умножению на x и сложению следующим образом:

$$\forall a, b \in GF(2^N),$$

$$a = a(x) = a_{N-1} * x^{N-1} + \cdots + a_1 * x + a_0,$$

$$b = b(x) = b_{N-1} * x^{N-1} + \cdots + b_1 * x + b_0,$$

$$a *_{GF(2^N)} b = \big(a(x) * b(x)\big) mod\ f(x) = \big((a_{N-1} * x^{N-1} + \cdots + a_1 * x + a_0) * b\big) mod f =$$

$$=$$

$$\Big(\ldots\Big(a_{N-1} * x *_{GF(2^N)} b + a_{N-2} * b\Big) *_{GF(2^N)} x + a_{N-3} * b\Big) *_{GF(2^N)} x + \cdots) *_{GF(2^N)} x + a_0 * b.$$

А $a *_{GF(2^N)} x$ выглядит следующим образом:

$$a *_{GF(2^N)} x = \begin{cases} a * x, & a_{N-1} = 0, \text{что эквивалентно } a \ll 1 \\ (a * x) \oplus f, & a_{N-1} = 1, \text{что эквивалентно } (a \ll 1) \oplus f \end{cases}$$

Континуантой **индекса** $\boldsymbol{n}$ называется многочлен $K_n(x_1, \ldots, x_n)$, определяемый рекуррентным соотношением:

$$K_{-1} = 0, K_0 = 1,$$

$$K_{m+1}(x_1, \ldots, x_m, x_{m+1}) = x_{m+1} * K_m(x_1, \ldots, x_m) + K_{m-1}(x_1, \ldots, x_{m-1}).$$

Кольцо K называется ***евклидовым***, если в нём можно делить с остатком.

Результант двух многочленов P и Q над некоторым полем K, старшие коэффициенты которых равны 1, называется выражение:

$$res(P, Q) = \prod_{(x,y):P(x)=0,Q(y)=0}(x - y).$$

Матрица $A = \{a_{ij}\}_{i,j=1}^{n}$ называется ***ганкелевой***, если на любой диагонали, перпендикулярной главной, у неё стоят равные элементы, то есть:

$$\forall 1 \le i_1, i_2, j_1, j_2 \le n : i_1 + j_1 = i_2 + j_2 \Rightarrow a_{i1j1} = a_{i2j2}.$$

Алгоритм 1. Возведение в степень.

По умножению, $GF(2^N)\backslash\{0\}$ – это циклическая группа порядка 2^{N-1}, поэтому справедливо соотношение:

$$a^{-1} = a^{2^N - 2} \quad \forall a \in GF(2^N).$$

Алгоритмическая сложность этого метода составляет $O(N^2)$, так как возведение в такую степень можно проделать за $O(N)$[1], и сложность умножения также составляет $O(N)$.

Алгоритм 2. Подсчёт континуанты.

Ещё один алгоритм обращения элементов в $GF(2^N)$ вытекает из следующего соображения: f – неприводим в $Z_2[x]$, а само $GF(2^N)$ – евклидово кольцо, поэтому:

$$\forall a \in GF(2^N)\backslash\{0\} \Rightarrow \text{НОД}(a, f) = 1.$$

Если мы напишем линейное представление $\text{НОД}(a, f)$, получим:

$$1 = f * q + a * w \ (mod\ f).$$

$$f * q \equiv 0(mod\ f) \Rightarrow a * w \equiv 1(mod\ f) \Rightarrow w = \ a^{-1}.$$

Если в процессе нахождения $\text{НОД}(a, f)$ получаем последовательность $q_1, \dots, q_s$ – частные при делении с остатком – то w можно представить в виде:

$$w = \ K_s\ (q_1, \dots, q_s), \text{где } K_s\ (q_1, \dots, q_s\ ) \text{ – это континуанта [2].}$$

Сложность этого алгоритма напрямую зависит от числа шагов нахождения НОД. Поэтому оценкой его скорости в самом плохом случае является O(N^2).

Алгоритм 3. Обращение элементов $GF(2^N)$ с использованием ганкелевых матриц.

Ещё один метод, основанный на результате Кронекера по представлению результанта полиномов посредством ганкелевой матрицы, предложен профессором А.Ю.Утешевым [4].

Сам алгоритм сводится к выполнению следующих действий:

1) Из f получается набор $\{d_i\}$ – коэффициентов разложения $\frac{1}{f}$ в ряд Лорана;
2) Из $\{d_i\}$ и обращаемого элемента a получается последовательность $\{c_i\}_{i=1}^{2*(N-1)}$;
3) Эта последовательность подается на вход алгоритму Берлекэмпа-Месси, на выходе получаем набор коэффициентов обратного элемента.

Сложность алгоритма в общем случае составляет $O(N^2)$. Однако, при хорошем f, сложность этого метода становится относительно невысокой, так как количество единиц в наборе $\{d_i\}$ оказывается мало (что облегчает работу остальных этапов).

Сравнение алгоритмов.

Стоит отметить, что все 3 алгоритма имеют одинаковую алгоритмическую сложность ($O(N^2)$), однако, из-за реализации базовых операций в $GF(2^N)$ и своих зависимостей от них имеют довольно сильно отличающиеся константы.

Алгоритм 1 не поддается эффективным оптимизациям, но одним из его достоинств является то, что количество операций является константой, а значит, используя, например, набор инструкций SSE или AVX можно существенно ускорить работу по обращению набора элементов, выполняя действия для обращения нескольких элементов одновременно. Зависит от реализации умножения в $GF(2^N)$.

Алгоритм 2 достаточно эффективен и обычно именно он используется при обращении элементов $GF(2^N)$. Однако число шагов в нём зависит от количества итераций поиска НОД(a, f), поэтому выполнять обработку нескольких элементов параллельно с помощью SSE или AVX нельзя. Зависит от реализации умножения и деления с остатком в $GF(2^N)$.

Алгоритм 3 не использует базовые операции в $GF(2^N)$, и работает, по своей сути, с векторами из 0 и 1. Это позволяет избавиться от негативного влияния усложнения операций в $GF(2^N)$ с ростом N. Его также нельзя распараллелить из-за явной зависимости поведения алгоритма Берлекэмпа-Месси от набора $\{c_i\}_{i=1}^{2*(N-1)}$, хотя длина набора всегда одна и та же. С другой стороны, одной из хороших сторон этого алгоритма является возможность совершить предрасчет шага 1), облегчающая работу этого алгоритма для большого объёма данных, в то время как у двух других алгоритмов такие оптимизации невозможны.

Реализация.

С целью максимально ускорить вычисления и использовать оптимизирующие возможности компилятора, было принято решение реализовывать алгоритмы на языке C.

Для всех алгоритмов был специально реализован расчётный модуль, предоставляющий вычисления базовых операций в $GF(2^N)$, чтобы уменьшить погрешность сравнения. В нём присутствуют операции сложения, умножения, деления с остатком.

Реализован был расчётный модуль для $N = 8, 16, 32, 64, 128, 256$. Дело в том, что существуют типы данных, имеющие ровно такую длину в битах, что облегчало работу с ними. Кроме того, в рамках данной задачи использовалось утверждение, что за 1 раз обрабатывается набор данных по длине равный 4 килобайтам. Поэтому эти значения оказываются хороши ещё и тем, что длина в 4 килобайта кратна им.

Для построения $GF(2^N)$использовались следующие f:

N	f
8	0x11D
16	0x1100B
32	0x100400007
64	0x10000400000000013
128	0x10..087
256	0x10...01000B

Таблица 1. Полиномы, использованные для построения $GF(2^N)$.

Это – полиномы с наименьшим числом единиц в двоичной записи [5]. Они хороши тем, что облегчают параллельное умножение в $GF(2^N)$, так как его можно реализовать за количество операций, линейно зависящее от количества единиц в f.

Замеры.

Замеры производились на компьютере со следующими характеристиками:

- Процессор - Intel® Xeon(R) CPU E5-2620 0 @ 2.00GHz × 18;
- ОЗУ - 39,4 ГиБ;
- ОС – Debian 7.0 x64.

$N\ (GF(2\hat{}N))$	Возведение в степень	Подсчёт континуанты	Использование ганкелевых матриц
8	741	480	1181
16	2395	1481	2991
32	8929	5675	8191
64	35774	19169	22671
128	878762	158103	51344
256	5216242	1176736	182407

Таблица 1: Скорость работы алгоритмов обращения (в ts)

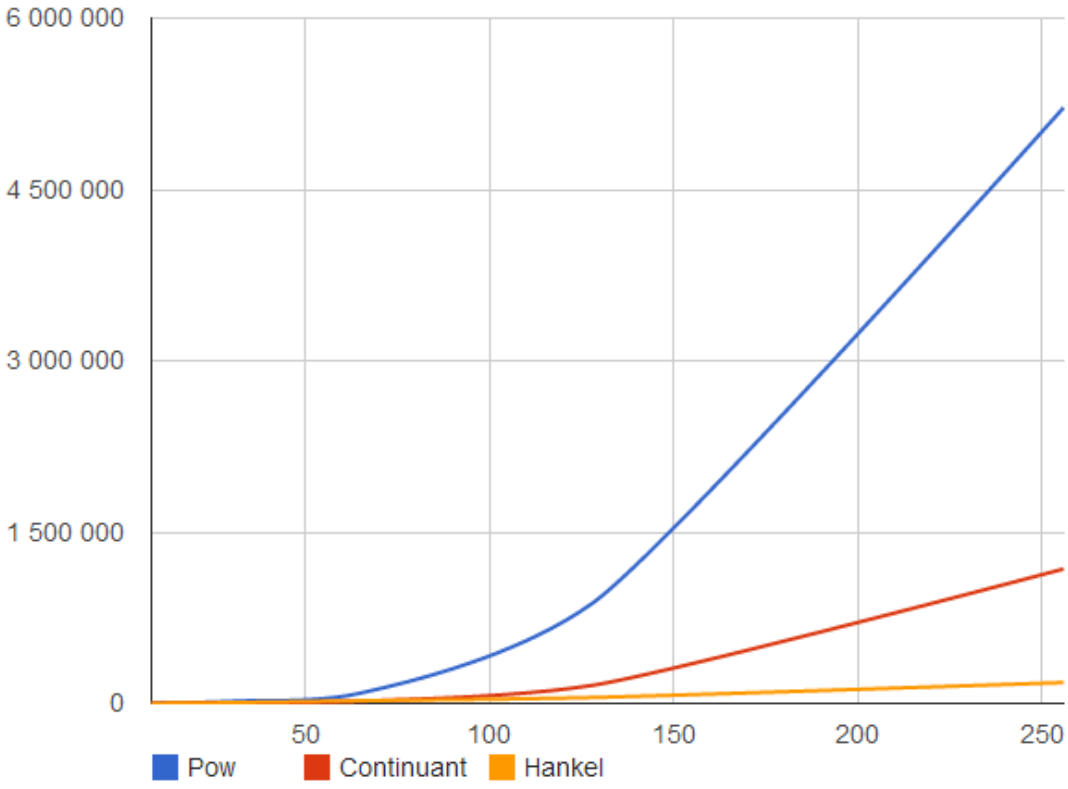

График 1: Сравнение производительности алгоритмов обращения

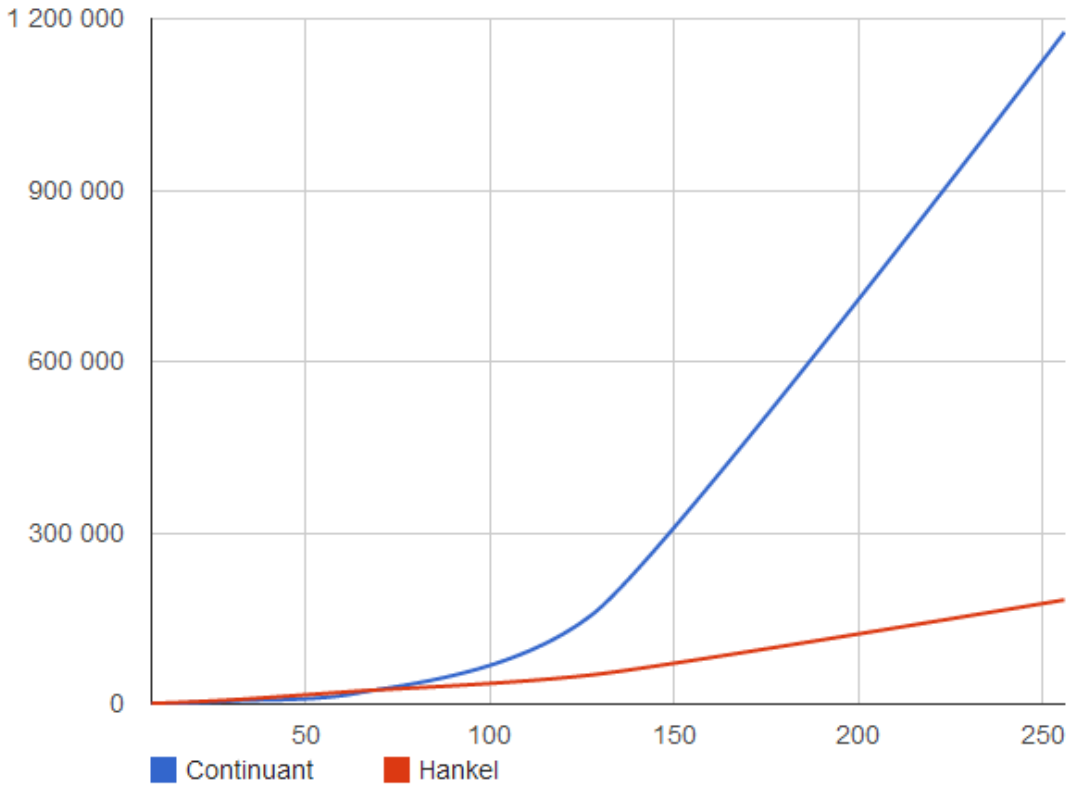

График 1: Сравнение производительности алгоритмов 2 и 3

Из замеров очевидно, что обращение континуантой выгодно использовать для полей с $N \leq 64$, а дальше – пользоваться последним алгоритмом. Дело в том, что, хоть при расчётах и можно использовать регистры длин 128 (SSE) или 256 (AVX), операция логического сдвига всего регистра на k разрядов пока что отсутствует, поэтому наблюдается резкое падение производительности на $N > 64$.

Результаты.

Мной были реализованы три алгоритма обращения элементов $GF(2^N)$ и проведено их сравнение. На основании этого можно сделать вывод, что для относительно небольших полей самым эффективным является метод подсчёта континуанты, а для больших – метод обращения элементов с использованием ганкелевых матриц. Результаты данной работы будут использованы компанией RAIDIX.

Список литературы.

1. Кнут Д. Э. *Искусство программирования. Т. 2: Полученные алгоритмы* / Кнут Д. Э. // Издательство Вильямс — 2004. — С. 832
2. Лидл Р., Нидеррайтер Г. *Конечные поля. Том 1.* / Лидл Р., Нидеррайтер Г. // М.Мир. — 1988.
3. Блейхут Р. *Быстрые алгоритмы цифровой обработки сигналов.* / Блейхут Р. // М. Мир. — 1989
4. Uteshev A.Yu., Cherkasov T.M. *The search for the maximum of a polynomial*. / Uteshev A.Yu., Cherkasov T.M. // J. Symbolic Computation. — 1998. — Vol. 25, № 5. — P. 587-618.
5. Gadiel Seroussi, *Table of Low-Weight Binary Irreducible Polynomials. /* Gadiel Seroussi // HPL-98-135, 1998.