ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А.Н. Тихонова

Дородный Дмитрий Алексеевич

Перевод статьти Роберта Сильвермана ”Мультиполиномиальное квадратичное решето”

Курсовая работа
по специальности 10.05.01 «Компьютерная безопасность»
студента образовательной программы специалитета

Студент	Преподаватель доцент
подпись	
Дородный Д.А.	Нестеренко А.Ю.
ФИО	подпись ФИО

Москва, 2021 г.

Оглавление

Аннотация

Обсуждается модификация Петером Монтгомери алгоритма факторизации больших целых чисел с использованием квадратичного решета, а также ее реализации. Использование модификации позволяет факторизацию с на порядок меньшим количеством просеивания, чем в базовом алгоритме. Она позволяет факторизовать числа в диапазоне 60 десятичных знаков за день, используя большой миникомпьютер. Алгоритм обладает свойствами, которые делают его хорошо адаптируемым к распараллеливанию.

0.1 Введение

Базовый алгоритм квадратичного решета имеет корни, которые отсылают к М. Краичику, однако явно заявил и проанализировал алгоритм С. Померанц. Было всего две реализации данного алгоритма, первая Дж. Гервером в Рутгерсе, а вторая Дж. Дэвисом и Д. Холдриджем в Национальных Лабораториях Сандия. Последние использовали компьютер Cray-1 чтобы факторизовать числа в диапазоне 60-70 десятичных знаков из "Проекта Каннингхема". Они также использовали Cray XMP чтобы разложить на множители тогда еще "разыскиваемое" число $(10^{71}-1)/9$ за 9 с половиной часов. И тогда как некоторые аргументы по теме были уже освещены Померанцем [1], мы представляем нашу статью как более ориентированную в сторону реализации алгоритма. Базовый алгоритм строится на составлении решения к следующему уравнению, где N - число, которое требуется факторизовать.

$$A^2 \equiv B^2 \mod N \tag{1}$$

Если $A \not\equiv B \mod N$ и $A \not\equiv -B \mod N$, тогда $(A+B), N$ и $(A-B, N)$ - делители N.

Эта версия квадратичного решета генерирует набор квадратичных вычетов N, используя следующий многочлен:

$$Q(x) = (x + [\sqrt{N}])^2 - N \equiv H^2 \mod N. \tag{2}$$

Сразу становится очевидно, что если простое число $p|Q(x)$, тогда $p|Q(x+kp)$ для всех $k \in$. Таким образом, значения многочлена можно профакторизовать с помощью решета, когда мы решаем $Q(x) \equiv 0 \mod p$. Это выражение может быть решено любым из числа доступных алгоритмов [2, стр 437]. Потенциальные делители p из $Q(x)$ в точности те простые чсла, символ Лежандра которых $(N/p) = 1$ и элемент -1 нужен, чтобы хранить знак. Теперь алгоритм продолжается следующим образом:

i Выбрать факторную базу $FB = p_i|(N/p) = 1, p_i, prime, i = 1, \ldots, F$ для какого-то подходящего значения F, и $p_0 = 1$ для знака.

ii Решить квадратное уравнение $Q(x) \equiv 0 \mod p_i$ для всех $p_i \in FB$. Для каждого из p_i будет найдено два корня: r_1, r_2.

iii Инициализировать массив просеивания нулями на интервале $[-M, M]$ для какого-то подходящего M.

iv Для всех $p_i \in FB$ прибавить значение $[log(p_i)]$ к ячейкам массива в индексах: $r_1, r_1 \pm p_i, r_1 \pm 2p_i, \ldots$ затем $r_2, r_2 \pm p_i, p_2 \pm 2p_i, \ldots$

v Значение $Q(x)$ будет примерно $M\sqrt{N}$ на $[-M, M]$, так что сравниваем каждую ячейку с $[\log N/2 + \log M]$. Полностью факторизованные вычеты будут иметь их соответствтующее значение близким к этому. Для таких значений построим точную факторизацию перебором делителей. Факторизации находятся так редко, что временные затраты на эти переборы делителей незначительны. При подсчете не требуется проверять все простые числа в факторной базе. Если x - ячейка в массиве просеивания, нужно только лишь вычислить $R \equiv x \mod p_i$. Только если R равно одному из двух корней следует продолжить и выполнить многоточное деление:

$$Q(x) = \prod_{i=0}^{F} p_i^{a_i}, p_i \in FB. \tag{3}$$

Пусть v_j - соответствующий вестор экспонент $[\alpha_{j1}, \alpha_{j2}, \alpha_{j3}, \ldots, \alpha_{jF}]$ с $H_j^2 = Q(x)$.

vi Насобирать $F + 1$ факторизацию. Затем требуется найти набор вычетов, произведение которых является квадратом при помощи метода Гаусса над полем $GF(2)$ на матрице, полученной при приведении всех векторов v_j по модулю 2. Это дает нам линейную зависимость по модулю 2 на степенях, и произведение векторов в этой зависимости дает квадрат. Затем, составление соотношения (1) - тривиальная задача.

Основная сложность этого подхода заключается в том, что необходимо насобирать количество вычетов, сопоставимое с количеством простых чисел в базе факторизации. Чтобы было возможно получить достаточно факторизаций, M должно быть очень большое, и вычеты будут расти линейно пропорционально M.

Обходной путь к этой проблеме был изначально предложен Петером Монтгомери. Просто использовать много многочленов, чтобы генерировать вычеты и просеивать каждый многочлен на намного меньшем интервале. Использование нескольких многочленов позволяет сохранить интервал просеивания довольно маленьким, что делает вычеты проще факторизуемыми. Мы применили этот подход и нашли его крайне эффективным. Он позволяет найти достаточно факторизованных остатков используя менее чем одну десятую чать от общей длины решета, используемого в версии с одним многочленом. Мы также показываем, что стоимость смены многочлена мала.

На самом деле, сандийская реализация была сделана в двух этапах. Их первая версия использовала один многочлен. Более поздняя версия использовала несколько многочленов, но в замаскированной а не в явной форме. Эта поздняя версия создавала неявные многочлены из подпоследовательностей основного решета, которое делилось на большое простое число q вне факторной базы. Они назвали этот метода "особые q". Многочлены, предложенные Монтгомери в какой-то мере лучше, однако, потому что они производят вычеты, которые в среднем меньшие и более простые в реализации.

В секции 2 мы обсудим, как выбирать коэффициенты многочленов. Секция 3 покажет, как эффективно вычислять эти коэффициенты. В секции 4 обсудим базовые шаги алгоритма. Секция 5 расскажет о выборе различных входных параметров алгоритма. Секция 6 представит некоторые числовые результаты. В секции 7 поговорим о параллельной реализации и, наконец, в секции 8 сравним наш алгоритм с другими методами.

0.2 Выбор коэффициентов

Выберем $Q(x) = Ax^2 + Bx + C$. Для того, чтобы $Q(x)$ мог генерировать квадратичные вычеты, необходимо чтобы $B^2 - 4AC = N$. Так как последнее выражение сравнимо с 0 или 1 по модулю 4, это означает, что если $N \equiv 3 \mod 4$ необходимо сначала умножить N на небольшую константу k, такую что $kN \equiv 1 \mod 4$. Вообще это хорошая практика, потому что обычно это позволяет построить факторную базу, в которой намного больше маленьких простых чисел. Мы позднее обсудим функцию, которая может быть использована для вычисления этого множителя k. Предложение Померанца [1] позволяет ограничиться лишь следующим выражением: $kN \equiv 1 \mod 4$. Нужно всего лишь взять средний коэффициент $2B$ вместо B. Однако, мы еще не дошли до реализации этого предложения. Для нас будет преимуществом возможность сохранить относительно малое значение $Q(x)$ в какой-то мере над [-M, M]. Есть несколько очевидных

способов сделать это. Например,

$$\text{Минимизируем } \sup |Q(x)| \text{на} [-M, M] \tag{4a}$$

или

$$\text{Минимизируем } \int_{-M}^{M} |Q(x)| dx \tag{4b}$$

или

$$\text{Минимизируем } \int_{-M}^{M} Q(x)^2 dx \tag{4c}$$

А также

$$B^2 - 4AC = kN \text{ и } A, B, C \in Z \tag{4d}$$

Легко заметить, что (4a) и (4b) практически эквивалентны. Длина основания параболы $2M$, и ее площадь будет прямо пропорциональна ее высоте. Уменьшая целочисленные ограничения и решая каждую из изложенных выше задач множителя Лагранжа, можно прийти к выводу, что все одни дают один и тот же результат. Точный ответ на (4a):

$$\begin{aligned} A &= W_1 \sqrt{kN}/M \\ B &= 0 \\ C &= W_2 M \sqrt{kN} \end{aligned} \tag{5}$$

где

$$W_1 = \sqrt{2}/2 \text{ и } W_2 = -1/2\sqrt{2}.$$

Единственное различие между результатами различными задачами минимизации это то, что константы W_1, W_2 слегка изменяются. W_1 изменяется в диапазоне от .70 до .75 и W_2 изменяется от $-.4$ до $-.3$ при $W_1 W_2 = -\frac{1}{4}$.

Максимальное значение $Q(x)$ над $[-M, M]$ это $M\sqrt{kN}/2\sqrt{2}$, улучшение на порядок $\sqrt{8}$ относительно (2) и "особыми q" многочленами в Сандии.

$B = 0$ непосредственно следует из соображений симметрии, но (4a) и (4c) дают похожие результаты для A и C потому что ограничение $B^2 - 4AC = kN$ сильно связывает форму $Q(x)$. Самый простой путь увидеть это - это осознать, что в корнях наклон графика будет $\pm\sqrt{kN}$. Нам бы очень хотелось сплющить параболу, но ограничение на дискриминант означает, что кривая должна иметь некоторую "крутость". Таким образом, мы не можем ничего поделать, кроме как двигать параболу вверх и вниз.

0.3 Вычисление коэффициентов

Простой метод для выбора A, B и C получается из метода для быстрого нахождения квадратных корней. Для удовлетворения (4d) необходимо чтобы

$$B^2 \equiv kN \mod 4A \tag{6}$$

Пусть $A = D^2, (D/kN) = 1, D \equiv 3 \mod 4$ и $A \approx \sqrt{kN/2}/M$. Желательно, чтобы D было простым, потому что если простое число из факторной базы делит A, то $Q(x) \equiv 0 \mod p$ имеет только один корень, и вероятность того, что $p|Q(x)$ на $[-M, M]$ падает с $2/p$ до $1/p$. Для

практических целей достаточно, чтобы D было только лишь возможно простым числом. Альтернативно, можно выбрать D как произведение простых чисел не входящих в факторную базу, но его факторизация должна быть известна, чтобы можно было решить (6). Наша реализация принимает D как возможно простое число. Чтобы найти коэффициенты, надо посчитать

$$h_0 \equiv (kN)^{(D-3)/4} \mod D, \tag{7a}$$

$$h_1 \equiv kNh_0 \equiv (kN)^{(D+1)/4} \mod D. \tag{7b}$$

тогда

$$h_1^2 \equiv kN(kN)^{(D-1)/2} \mod D \equiv kN \mod D, (D/kN) = 1 \tag{8}$$

Пусть

$$h_2 \equiv (2h_1)^{-1}[\frac{kN - h_1^2}{D}] \mod D \tag{9}$$

Тогда имеем

$$B \equiv h_1 + h_2 D \mod A \tag{10}$$

и

$$B^2 \equiv h_1^2 + 2h_1h_2D + h_2^2D^2 \equiv kN \mod A \tag{11}$$

Так как B должно быть нечетным, если оно получилось четное - вычесть его из A. Значение $(2h_1)^{-1} \mod D$ легко получается, так как $h_0 \equiv h_1^{-1} \mod D$ уже было посчитано. Также имеем

$$C = [\frac{B^2 - kN}{4A}] \tag{12}$$

Это не необходимо, на практике, действительно вычислять (12), так как значение C на самом деле не требуется. Однако, вычисление можно провести для проверки других вычислений. Посчитаем и сохраним значение $1/2D \mod kN$ на потом. Это позволит нам быстро посчитать $Q(x)$ огда нам нужно найти факторизацию. Как результат того, как мы выбирали D, мы также имеем

$$Q(x) \equiv H^2 \equiv (\frac{2Ax + B}{2D} \mod kN). \tag{13}$$

Требуется некоторая осторожность: Если x находится между действительными корнями $Q(x)$, тогда $Q(x)$ - отрицательное, и необходимо вычесть его значение из kN.

Стоимость нахождения коэффициентов по большей части зависит от нахождения возможно простого числа и проверки вычетом на D, подсчетом h_1 и $1/2h_1 \mod D$ и $1/2D \mod kN$. Наконец, корни $Q(x) \mod p_i, p_i \in FB$

$$(-B \pm \sqrt{kN})(2A)^{-1} \mod p_i, \tag{14}$$

так как $B^2 - 4AC$ инвариантно.

Большинство стоимости смены многочленов появляется в вычислении (14). Стоимость подсчета (14) доминируется вычислением $1/2A \mod p_i$, что должно быть выполнено для всех простых чисел в базе факторизации. Даже с эффективным алгоритмом для выполнения этой операции, таким как расширенный алгоритм Эвклида, как правило приходится выполнять эти операции тысячи раз при смене многочленов. На SUN-3/75, для числа длиной 60 десятичных знаков и факторной базой из 3000 простых чисел, требуется $.9$ секунд чтобы посчитать все коэффициенты и 2.6 секунд, чтобы посчитать все корни.

0.4 Описание алгоритма

Здесь мы показываем базовые шаги в алгоритме.

i Выберем множитель k, такой что $kN \equiv 1 \mod 4$ и kN богато на маленькие квадратичные вычеты. Мы предпочитаем $kN \equiv 1 \mod 8$, так как $2 \in FB$ только если это условие выполняется.

ii Выберем размер факторной базы F, длину интервала просеивания $2M + 1$, и большую константу T. Предложения по поводу значений этих величин даны ниже.

iii Посчитаем тестовое значение

$$[\log \frac{M\sqrt{kN/2}}{p_{max}^T}]$$

Где p_{max} - это наибольшее простое число из базы факторизации. Если $T \leq 2$, то когда бы значение в ячейке массива просеивания ни превосходило это тестовое значение, это будет означать, что значение соответствующего $Q(x)$ будет полностью факторизуемым. Если $T > 2$, тогда $Q(x)$ может не раскладываться полностью. Однако, эти частичные факторизации могут также быть полезными, как мы увидим далее.

iv Посчитать факторную базу FB и $\sqrt{kN} \mod p_i$ для всех $p_i \in FB$. Посчитать $[\log p_i]$ для всех $p_i \in FB$.

Algorithm 1: MPQS

```
Result: 2 делителя числа N
Выбор множителя k : k * N ≡ 1 mod 8;
Выбор параметров M, F, T;
Подсчет тестового значения;
Генерация факторной базы fBase;
while Недостаточно факторизаций do
    Подсчет коэффициентов многочлена Q(x);
    Решение Q(x) = 0 mod p_i ∀p_i ∈ fBase;
    Просеивание полученных решений;
    for Значения массива, большие тестового do
        Вычислить Q(x);
        Перебрать делители по факторной базе;
        if Полностью факторизуется then
            сохранить степени разложения и √Q(x) mod kn;
        else
            выполнить LPP;
        end
    end
end
```

v Процедура Больших Простых (LPP). Многие из факторизаций найденных в iv не будут полными по факторной базе. Однако, можно заметить, что если $Q(x)$ факторизуется как

$$Q(x) = \prod_i p_i^{a_i} L, L > 1 \tag{15}$$

Тогда если мы находим два или более раз одинаковые значения L, мы можем перемножить соответствующие значения (15) друг с другом. Это даст нам делитель L^2 в правой

части выражения и мы также можем сохранить этот результат для шага с приведением матрицы. Важно, что L не обязательно должно быть простым числом: нам только нужно чтобы 2 или более совпадали. Мы ищем совпадения просто сортируя все значения (15) где ключ - значение L. Мы говорим о (15) как о факторизации большого простого числа. Значение T в ii позволяет нам контролировать размер L. Мы решили сохранять все значения L, которые меньше чем p_{max}^T, где p_{max} - наибольший элемент в факторной базе. Эти изменения могут привести к уменьшению времени работы алгоритма более чем вдвое. Пусть FF - это количество вычетов, полностью факторизованных на нашей факторной базе, а FT - общее количество факторизаций по факторной базе. Тогда

$$R = \frac{FT}{F + FT - FF} \tag{16}$$

Используем это, чтобы определить, когда получено достаточное количество факторизаций. Почти всегда можно обнаружить зависимость, где $R = .96$. На практике вовсе не требуется набирать все $F + 1$ факторизаций. Обычно, хватает около $.9F$ рядов в матрице чтобы найти зависимость, и $R = .96$ означает как раз это.

vi Приведение матрицы. Наконец, собираем все найденные факторизации и приводим матрицу над полем $GF(2)$. Для каждой линейной зависимости S,

$$\begin{aligned} P_1 &\equiv \prod_j H_j \mod kN, j \in S, \\ P_2 &\equiv \prod_i p_i^{\sum_j V_{ji}/2} \mod kN, \forall p_i \in FB, j \in S. \end{aligned} \tag{17}$$

Если $P_1 \not\equiv P_2$ и $P_1 \not\equiv -P_2 \mod kN$, тогда $(P_1 + P_2, kN)$ и $P_1 - P_2, kN$ будут нетривиальными делителями N.

vii Некоторые размышления о программировании. Инициализация массив просеивания при смене многочленов занимает какое-то время. Так как решето - это массив байтов, стоит привести массив просеивания к массиву полных слов на устройстве перед инициализацией. Можно также объединить инициализацию массива с просеиванием самых маленьких простых чисел, если правильно выбрать значение, которым инициализировать массив.

0.5 Выбор вводных параметров

Автор нашел, после многих попыток, что следующие комбинации параметров работают хорошо

Множитель может быть посчитан при помощи следующей, модифицированной версией функции Кнута-Шреппеля.

$$f(k, N) = \sum_i g(p_i, kN) \log p - \frac{1}{2} \log k \forall p_i \in FB \tag{18}$$

$$\begin{aligned} g(p, kN) &= 2/p \text{ если } p \not| k \\ &= 1/p \text{ если } p|k, \end{aligned} \qquad \begin{aligned} g(2, kN) &= 2 \text{ если } N \equiv 1 \mod 8 \\ &= 0 \text{ иначе} \end{aligned}$$

Количество знаков	Размер факторной базы	M	T	Среднее время работы на VAX/780
24	100	5K	1.5	15 сек
30	200	25K	1.5	80 сек
36	400	25K	1.75	400 сек
42	900	50K	2.0	1800 сек
48	1200	100K	2.0	8100 сек
54	2000	250K	2.2	27600 сек
60	3000	350K	2.4	97200 сек
66	4500	500K	2.6	360000 сек

Таблица 1: Таблица реузльтатов факторизаций различных числел

В [3] было замечено, что не очень удобно хранить все факторизации больших простх чисел, который удалось обнаружить. Для этого алгоритма время, затраченное на перебор делителей, весьма значительно для маленьких чисел. Большое T означает, что алгоритм затрачивает много времени составляя факторизации, которые мы можем не использовать в итоге. Это из-за того, что вероятность найти совпадение между двумя большими простыми невелика. Когда числа становятся больше, часть времени, затраченная на факторизацию резко уменьшается. Так как стоимость обработки больших простых чисел маленькая, выгодно хранить как можно больше таких чисел. Для удобства программирования и простоты обработки данных мы ограничили нашу программу числами одинарной точности на нашей машине (32 бита). Значение p_{max}^T как правило больше чем 32 бита, однако, для чисел, которые длиннее 54 зников. Причина этому - требуется какой-то допуск для ошибки округления, так как мы работаем только лишь с целочисленными приближениями к $\log p$. Также, так как просеивание с учетом степеней простых чисел весьма затратно, мы этого не делаем. Вместо этого, несколько экземпляров одного и того же простого числа находятся при создании факторизаций. Выбор большего значения T допускает больше многостепенных делителей.

также, из-за различий в архитектуре машин, желательно разбивать решето на части. Оптимальный размер как правило зависит от размера кэш-памяти, доступной на устройстве. При просеивании, желательно сократить ссылки на глобальную память до минимума и для этого мы должны сделать каждую часть интервала решета достаточно маленькой, чтобы она влезла в кэш-память.

В зависимости от качества множителя k и соответствующего значения модифицированной функции Кнута-Шреппеля, часто можно заметить колебания во времени работы до 2.5 раз. Поэтому, время, указанное выше только примерное и представляет типичные результаты по времени, полученные нами в наших вычислениях.

0.6 Некоторые числовые результаты

Мы представляем несколько разложений из "MOST/MORE wanted" таблиц из Каннингхемского проекта [4], вместе с несколькими другими интересными числами Фибоначчи и Лукаса (U, V соответственно в таблице) и временами факторизаций. Все вычисления производились на SUN-3/75. Как правило, примерно 15-20 процентов общего времени работы было потрачено на вычисление коэффициентов многочленов и поиск их корней, хотя большие числа занимали около 30 процентов. Обработка больших простых чисел из (15), этап приведения матрицы и вычисление соотношения (1) занимало очень малую часть общего времени работы. Эта постобработка зависит только от размера факторной базы и не от числа, которое раскладывается на множители. Для факторной базы из 3000 простых чисел, обработка больших простых чисел заняла около 10 минут, приведение матрицы - 20 минут, вычисление соотношения (1) - 1 минуту.

Для факторной базы из 1000 простых чисел затратили 1 минуту на все эти шаги.

Представление $a, b+$ или $a, b-$ означает $a^b + 1$ и $a^b - 1$ соответственно, тогда как $2446L$ - это часть факторизации числа $2^4 46 + 1$ Ауревелля [4]. Представление Pxx означает простое число длиной xx символов. Факторизации помеченные * были сделаны при помощи параллельной реализации алгоритма, описанной ниже. Время работы для таких факторизаций - это сумма времен на всех машинах.

Number	Size	Factors	TIME (hrs)
V298	45	271293387891105049.P28	0.25
U507	53	173408891952128923997973.P27	2.50
U615	54	1846858344247612322281.P33	2.33
12,67–	54	17577834702049211.P38	3.10
6,91–	56	48215910563832798697.P37	4.10
5,83 +	56	4029666108840585686296627.P31	3.75
6,86 +	56	29147649893760430207 33.P35	6.50
2,224 +	58	167773885276849215533569.P35	11.5
3,131 +	60	114742271896804438572098194909.P30	10.0
2,446L	60	52016435676012089.P43	14.2
2,263–	62	120226360536848498024035943.P36	15.2
11,62 +	63	4311672901046383796549.P41	22.2*
2,239 +	66	32605142983704221670173899.P41	42.1
7,79–	66	913242407367610843676812931.P40	37.5
6,94 +	69	1029538544148223697293. 3058576236553368725 2981.P25	81.0
10,73 +	70	10826684964539959837294043117.P42	87.0*
2,272 +	74	335631827046798245410603730138717057.P38	255.0*
6,128 +	75	2339340566463317436161. 2983028405608735541756929.P29	315.0*
2,269 +	81	4242559157961874288938 11.P57	1265*

0.7 Параллельная реализация

Строение алгоритма позволяет легко распараллелить его между машинами, и мы уже сделали это ранее. Центральный процессор выбирает значение D из Части 3 ,и передает его сопроцессору. Сопроцессов вычисляет корни многочленов, выполняет просеивание и возвращает любые найденные факторизации на центральный процессов. Так как все вычисления на сопроцессорах выполняются независимо друг от друга, и так как сопроцессоры коммуницируют только с центральным процессором а не друг с другом, можно достичь максимальной теоретической эффективности всех процессоров. В нашей реализации мы достигли результата с ускорением в N раз при использовании N процессоров. Еще один плюс нескольких сопроцессоров - это сильно увеличенная реальная память. Количество сваппинга и пейджинга уменьшено фактически до нуля, и у нас намного меньше пропусков кэша.

0.8 Сравнение с другими методами

0.8.1 Алгоритм на непрерывных дробях

Алгоритм CFRAC Моррисона и Бриллхарт был чемпионом среди факторизующих алгоритмов вплоть до реализации квадратичного решета Дэвисом. Считалось, что точка пересечения между этими двумя алгоритмами находится где-то между числами из 50 и 70 десятичных знаков.До настоящего момента никто не реализовывал оба эти алгоритма на одной машине. У нас имеется серьезный опыт работы с VAX версией алгоритма CFRAC, которая использует стратегию раннего выхода Померанца, и мы представляем сравнение этих двух методов:

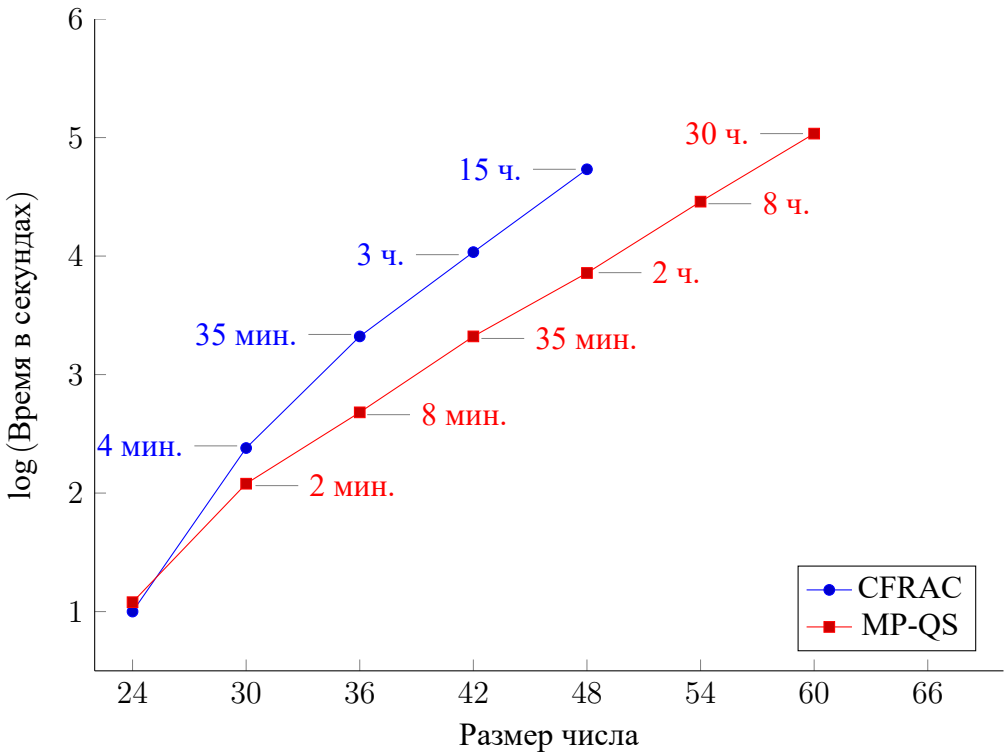

Рис. 1: Сравнение скорости двух алгоритмов

Версия квадратичного решета с несколькими многочленами значительно быстрее, как только длина числа становится хоть сколько то большой.Когда берется число M достаточно большое, чтобы наш алгоритм использовал только один многочлен, время работы сильно возрастает. Точка пересечения между этими алгоритмами находится где-то в районе чисел длиной 40 десятичных знаков.

0.8.2 Квадратичное решето с одним многочленом

Мы представляем здесь данные, взятые из [5], вместе с нашими собственными данными, которые показывают общее количество вычетов, исследованных вариантами QS. Эти данные представляют типичные значения для общего числа просеянных вычетов как нашим методом, так и методом ”особых q” Дэвиса. Данные значения типичны для чисел данного размера, но столбцы не представляют факторизации одинаковых чисел.

Можно заметить резкое улучшение в сравнении с базовым алгоритмом, и в случае ”особых q” и в нашей версии. Наша версия показала себя несколько лучше по следующим причинам:

i Мы использовали множители а Сандия - нет

Размер	MP-QS/VAX	SPECIAL-Q/CRAY	BASIC-QS/CRAY
52	8.0E + 8	1.0E + 9K	9.0E + 9
53	4.0E + 8	-	2.0E + 10
55	5.0E + 8	-	1.4E + 10
58	1.0E + 9	4.3E + 9	2.7E + 10
60	2.1E + 9	2.2E + 10	9.0E + 10
63	1.0E + 9	1.2E + 10	-

Таблица 2: Сравнение времени работы разных алгоритмов и компьютеров

ii Мы использовали версию с большими простыми числами

iii Мы изменяли многочлены так часто как только можно, при использовании большего количества ”особых q” можно было бы улучшить результат Сандии

iv Архитектура трубопровода, реализованная в Cray относительно быстрее чем VAX, даже принимая во внимание разницу в машинной скорости. Так что более выгодно делать больше просеиваний на каждом многочлене

v Наши многчлены производили намного меньшие вычеты

Эти результаты показывают, что желательно изменять многочлены так часто, как можно. Точное время, затраченное на это, в сравнении с просеиванием будет зависеть от машины и скорее всего требует экспериментов. Изменяя многочлены часто мы получаем как минимум 10 знаков преимущества над базовым алгоритмом, и новые многочлены, представленные нами, дают 1 дополнительный знак. Автор благодарен Петеру Монтгомери в Корпорации Разработки Систем за предоставление многих советов и предложения, Дж. Дэвису за разрешение на публикацию результатов из таблицы 3, и центру компьютерных исследований в MITRE за предоставление компьютерного времени, использованного в этом проекте.

Литература

[1] К. Померанц. Личная беседа.

[2] Д. Кнут. *Искусство программирования*, volume 2 of *Получисленные алгоритмы*. Addison Wesley, 2 edition, 1981.

[3] М. Моррисон and Дж. Бриллхарт. Методы факторизации f7. *Math. Comp.*, 29:183–205, 1975.

[4] Дж. Буллхарт, Д. Лемер, Дж. Селфридж, Б. Такерман, and С. Вагстафф мл. Факторизации $b^n \pm 1, b = 2, 3, 5, 6, 7, 10, 11, 12$ включая большие степени. *Contemp. Math.*, 22, 1983.

[5] Дж. Дэвис and Д. Холдридж. Факторизация с использованием алгоритма квадратичного решета. *Sandia Report Sand*, 83-1346, 1983.

[6] Дж. Дэвис and Д. Холдридж. "Отчет о статусе факторизации", Продвижения в криптологии. *Записки с лекций компьютерных наук*, 209:183–215, 1985.

[7] Дж. Гервер. Факторизация больших чисел с помощью квадратичного решета. *Math. Comp.*, 41:287–294, 1983.

[8] П. Монтгомери. Личная беседа.

[9] К. Померанц. "Алгоритм факторизации с квадратичным решетом", Продвижения в криптологии. *Записки с лекций компьютерных наук*, pages 169–182, 1985.

[10] К. Померанц. Анализ и сравнение некоторых целочисленных алгоритмов факторизации. *Math. Centrum Tract*, 154:89–139, 1982.

[11] К. Померанц and С. Вагстафф мл. Реализация алгоритма на непрерывных дробях. *Cong. Numerantium*, 37:99–119, 1983.