Математические структуры и моделирование
2012, вып. 26, с. 94–109

УДК 519.688:004.75

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ И СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ ФАКТОРИЗАЦИИ В СИСТЕМАХ С РАСПРЕДЕЛЁННОЙ ПАМЯТЬЮ

А.В. Макаренко, А.В. Пыхтеев, С.С. Ефимов

В статье рассматриваются наиболее популярные на сегодняшний день алгоритмы факторизации целых чисел, предложена их параллельная реализация в системах с распределённой памятью. Получены сравнительные оценки для каждого метода на различном числе параллельных процессов. Сделаны общие выводы об эффективности использования алгоритмов.

Введение

Некоторые часто используемые алгоритмы асимметричного шифрования, такие как RSA, используют вычислительную сложность факторизации в качестве основы своей криптостойкости. Если для гипотетического квантового компьютера алгоритм, осуществляющий разложение числа на простые множители за полиномиальное время, уже разработан, то вопрос существования такового для классического компьютера остаётся открытым. А значит, ускорение существующих методов факторизации является одной из наиболее актуальных задач. Кроме всевозможных доработок и улучшений самих алгоритмов наиболее эффективным средством ускорения их работы является параллельная реализация и выполнение на системах с распределённой или общей памятью.

Ранее K.A. Kanabar проводил достаточно объёмные исследования некоторых распространённых алгоритмов факторизации, в том числе анализировались такие, как метод Ферма, ρ-метод Полларда, метод эллиптических кривых, которые также были исследованы в настоящей работе. Kanabar предложил такую методику измерений, при которой используются числа с двумя близкими сомножителями, тремя близкими сомножителями или тремя произвольными сомножителями. При этом последовательно увеличивается разрядность исследуемых чисел. В работе также приведены общие результаты по сравнению быстродействия алгоритмов факторизации, среди которых, кроме названных выше, исследовались метод последовательного деления и (P-1)-метод Полларда.

Омский государственный университет им. Ф.М. Достоевского
E-mail: alv.makarenko@gmail.com, alex.pykhteyev@gmail.com, s_efimov@mail.ru

R.P. Brent в своей работе [1] обобщил способы модификации и улучшения многих алгоритмов факторизации, а также привёл несколько рекомендаций по частичной или полной параллельной реализации алгоритмов, многие из которых были использованы нами в данном исследовании. Также были приведены оценки ускорения работы параллельных алгоритмов по сравнению со своими последовательными аналогами и их строгое математическое обоснование.

Цель проведённого исследования — реализовать параллельные алгоритмы факторизации, оценить их эффективность, сравнив и систематизировав результаты, полученные в ходе проведения экспериментов.

В данной работе использовались алгоритмы и результаты описанные в [1–7].

1. Алгоритмы факторизации

В работе исследованы следующие методы:

Экспоненциальные:

- метод Ферма,
- ρ-метод Полларда,
- метод Лемана,
- метод Шенкса.

Субэкспоненциальные:

- метод эллиптических кривых,
- метод квадратичного решета.

Классификация методов произведена в соответствии с их вычислительной сложностью.

Далее следует псевдокод вышеназванных методов на алгоритмическом языке, составленный нами на основе реального программного кода, а также описаны особенности параллельной реализации алгоритмов.

1.1. Метод Ферма

На входе: нечётное число n

1: $x \leftarrow [\sqrt{n}]$
2: $y \leftarrow x^2 - n$
3: **while** $\mathrm{isSquare}\,(y) \neq$ **true do**
4: $\quad x \leftarrow x + 1$
5: $\quad y \leftarrow x^2 - n$

6: **end while**
7: $s \leftarrow x - \sqrt{y}$
8: $t \leftarrow x + \sqrt{y}$
9: **if** $s \neq 1$ **and** $s \neq n$ **then**
10: **return** s, t
11: **end if**

Функция isSquare $(\dots)$ возвращает $true$, если аргумент является полным квадратом, и $false$ иначе.

Параллельная реализация основана на том, что каждый процесс, участвующий в факторизации, обрабатывает свои значения в качестве кандидатов на число x, проверяет все условия, описанные в алгоритме, определяя, найден ли нетривиальный множитель. Через некоторые промежутки времени главный процесс-координатор «опрашивает» вычислительные процессы на предмет выявления множителя. Если этот множитель найден одним из процессов, то поступает сигнал о завершении всех процессов, иначе продолжается поиск множителей.

1.2. Метод Шермана Лемана

На входе: нечётное число n.

1: **for** $a \leftarrow 2$ **to** $\left[n^{1/3}\right]$ **do**
2: **if** $a|n$ **then**
3: **return** a
4: **end if**
5: **end for**
6: **for** $k \leftarrow 1$ **to** $\left[\mathrm{n}^{1/3}\right]$ **do**
7: **for** $d \leftarrow 0$ **to** $\left[\frac{n^{1/6}}{4\sqrt{k}}\right] + 1$ **do**
8: **if** isSquare $\left(\left(\left[\sqrt{4kn}\right] + d\right)^2 - 4kn\right)$=**true then**
9: $A \leftarrow \left[\sqrt{4kn}\right] + d$
10: $B \leftarrow \sqrt{A^2 - 4kn}$
11: **if** $1 < \gcd(A + B, n) < n$ **then**
12: **return** $A + B$
13: **else if** $1 < \gcd(A - B, n) < n$ **then**
14: **return** $(A - B)$
15: **end if**
16: **end if**
17: **end for**
18: **end for**

Функция $\gcd(\dots)$ возвращает наибольший общий делитель аргументов.

Параллельная реализация аналогична таковой в методе Ферма. На первом шаге каждый процесс обрабатывает свои значения числа a, на втором шаге

каждый процесс проверяет свой диапазон чисел k. При этом результаты через некоторое количество итераций отправляются главному процессу-координатору, вследствие чего им принимается решение о завершении всех процессов, если нетривиальный множитель найден, или о продолжении работы алгоритма.

1.3. Метод Шенкса

На входе: нечётное число n, маленький множитель k

1: $P_0 \leftarrow \left\lceil \sqrt{kn} \right\rceil$
2: $Q_0 \leftarrow 1$
3: $Q_1 \leftarrow kn - P_0^2$
4: **repeat**
5: $\quad b_i \leftarrow \left\lfloor \dfrac{\left\lceil \sqrt{kn} \right\rceil + P_{i-1}}{Q_i} \right\rfloor$
6: $\quad P_i \leftarrow b_i Q_i - P_{i-1}$
7: $\quad Q_{i+1} \leftarrow Q_{i-1} + b_i \left(P_{i-1} - P_i\right)$
8: **until** isSquare$(Q_i) =$ **false**
9: $b_0 \leftarrow \left\lfloor \dfrac{\left\lceil \sqrt{kn} \right\rceil + P_{i-1}}{Q_i} \right\rfloor$
10: $P_0 \leftarrow b_0 \sqrt{Q_i} + P_{i-1}$
11: $Q_0 \leftarrow \sqrt{Q_i}$, $Q_1 \leftarrow \dfrac{kn - P_0^2}{Q_0}$
12: **repeat**
13: $\quad b_i \leftarrow \left\lfloor \dfrac{\left\lceil \sqrt{kn} \right\rceil + P_{i-1}}{Q_i} \right\rfloor$
14: $\quad P_i \leftarrow b_i Q_i - P_{i-1}$
15: $\quad Q_{i+1} \leftarrow Q_{i-1} + b_i \left(P_{i-1} - P_i\right)$
16: **until** $P_{i-1} \neq P_i$
17: **if** $1 < \gcd(P_i, n) < n$ **then**
18: $\quad$ **return** $\gcd(P_i, n)$
19: **end if**
20: **return** FAIL

Если нетривиальный делитель не найден, то следует использовать другое значение k.

Параллельная реализация данного алгоритма заключается в использовании различных маленьких множителей k. Главный процесс выступает в роли координатора. Остальные процессы выполняют описанный выше алгоритм со своим k. Как только какой-либо из процессов находит нетривиальный делитель, то происходит отправка его координатору, который посылает сигнал завершения работы алгоритма каждому процессу.

1.4. Метод ρ-Полларда

На входе: нечётное число n.

1: $x_i \leftarrow \text{Random}\,(n)$
2: $x_{2i} \leftarrow f\,(x_i) \mod n$
3: **while** $\gcd\,(x_{2i} - x_i, n) = 1$ **or** $\gcd\,(x_{2i} - x_i, n) = n$ **do**
4: $\quad x_i \leftarrow f\,(x_i) \mod n$
5: $\quad x_{2i} \leftarrow f(f\,(x_i)) \mod n$
6: **end while**
7: **return** $\gcd\,(x_{2i} - x_i, n)$

Функция $Random\,(n)$ возвращает случайное число от 1 до n. Функция $f\,(x_i)$ выбирается с коэффициентами из Z_n. Обычно, $f\,(x_i) = x^2 + 1$.

Параллельная реализация данного алгоритма заключается в генерации различных значений x_i для каждого процесса. Главный процесс выступает в роли координатора. Таким образом, удаётся просмотреть различные рекуррентные последовательности для поиска нетривиального делителя. Как только какой-либо из процессов находит нетривиальный делитель, то происходит отправка его координатору, который посылает сигнал завершения работы алгоритма каждому процессу.

1.5. Метод эллиптических кривых

На входе: число n, границы $v\,(n)\,,\ w\,(n) \in\ \mathbb{N}$. Выбираем случайную ЭК $E_{a,b} \mod n$ и точку $P_0 = (x_0, y_0)$ на ней.

1: $k \leftarrow 1$
2: $m \leftarrow 0$
3: **for** r – простое, $r \leftarrow 2$ **to** w **do**
4: $\quad$ **while** $r^m \leq v + 2\sqrt{v} + 1$ **do**
5: $\quad\quad m \leftarrow m + 1$
6: $\quad$ **end while**
7: $\quad k \leftarrow k \times r^m$
8: $\quad m \leftarrow 0$
9: **end for**
10: $d \leftarrow \text{compute}(kP_0)$
11: **if** $1 < d < n$ **then**
12: $\quad$ **return** d
13: **end if**
14: **return** FAIL

Функция compute(kP) возвращает делитель числа n в ходе вычисления кратного k точки P. Возвращаемое значение FAIL в алгоритме означает, что метод эллиптических кривых является вероятностным, и попытка нахождения нетривиального делителя по вышеописанному алгоритму не всегда даёт положительный результат.

Параллельная реализация основана на том, что каждый процесс выбирает разные эллиптические кривые и далее выполняет алгоритм, описанный выше, без изменений.

1.6. Метод квадратичного решета

На входе: число n, количество чисел k факторной базы, радиус интервала просеивания L.

1: $FB \leftarrow$ buildFactorBase(k)
2: **for** $i \leftarrow 1$ **to** k **do**
3: $x \leftarrow$ solveEquation($FB[i]$)
4: solutions.add($x, FB[i]$)
5: **end for**
6: **for** $i \leftarrow -L$ **to** L **do**
7: $W[i] \leftarrow (m+i)^2 - n$
8: **end for**
9: **for** $i \leftarrow 1$ **to** solutions.size() **do**
10: sieving($solutions[i]$)
11: **end for**
12: $M \leftarrow$ createMatrix()
13: **if** $1 <$ solveMatrix(M) $< n$ **then**
14: **return** solveMatrix(M)
15: **end if**
16: **return** FAIL

Функция buildFactorBase(k) возвращает факторную базу размерностью k. Функция solveEquation(FB[i]) ищет решение сравнения:

$$(m+x)^2 - n \equiv 0 \pmod{FB[i]},$$

где $m = [\sqrt{n}]$. Массив solutions хранит пары чисел (x, p), где x является решением сравнения $(m+x)^2 - n \equiv 0 \pmod{p}$. Функция solutions.size() возвращает размер массива solutions. Функция createMatrix() строит систему линейных уравнений. Функция solveMatrix(M) решает систему линейных уравнений и возвращает делитель (возможно тривиальный) числа n.

Если нетривиальный делитель не был получен, то следует увеличить радиус просеивания или размер факторной базы.

Параллельная реализация данного алгоритма заключается в разбиении интервала просеивания на несколько частей. Каждый процесс ищет гладкие числа в своём интервале и результат отправляет главному процессу, который собирает все данные и переходит к составлению системы линейных уравнений на основе полученной информации.

2. Результаты экспериментов

Все алгоритмы выполнялись на языке программирования Java с использованием Java SE 6 update 31. Для параллельной реализации алгоритмов исполь-

зовалась адаптация для Java интерфейса передачи сообщений MPI (Message Passing Interface) — MPJ Express v0.38.

Исследование проводилось на компьютерах следующей конфигурации:

- ЦП — Dual Core E5200 2.5 ГГц,
- ОЗУ — 3.24 Гб,
- ОС — Mandriva Linux 2008,
- ЛВС — 100 Мбит/с.

Алгоритмы факторизации тестировались на числах вида $n = pq$, где p, q — простые, так как разложение именно таких чисел на множители является наиболее трудной задачей.

В ходе исследования производилось два вида экспериментов:

- Факторизация чисел n фиксированной разрядности (80 бит) с последовательным увеличением расстояния между множителями p и q от двух до четырёх миллиардов.
- Факторизация чисел n с последовательным увеличением разрядности числа (от 20 до 110 бит), где $p \in \left(0,\ \ n^{1/3}\right]$ — небольшой простой множитель.

В ходе выполнения каждого алгоритма факторизации были последовательно задействованы 1, 2, 4 и 8 процессов, каждый из которых работал на отдельном компьютере, то есть тесты проводились на вычислительной системе с распределённой памятью.

2.1. Метод Ферма

Время нахождения нетривиального делителя (см. рис. 1) числа линейно возрастает с увеличением расстояния между двумя делителями. При этом если делители достаточно близки друг к другу, то разложение происходит относительно быстро. Именно поэтому в асимметричной системе шифрования RSA одним из основных требований является выбор множителей модуля, расстояние между которыми достаточно велико, иначе модуль возможно факторизовать за малое время с помощью метода Ферма, и шифр окажется ненадёжным.

Кроме того, было достигнуто линейное ускорение при увеличении количества процессов, задействованных в нахождении множителя числа.

Для данного метода измерения второго типа с последовательным увеличением разрядности факторизуемого числа не проводились, так как метод Ферма крайне медленно работает с числами, которые содержат малый множитель.

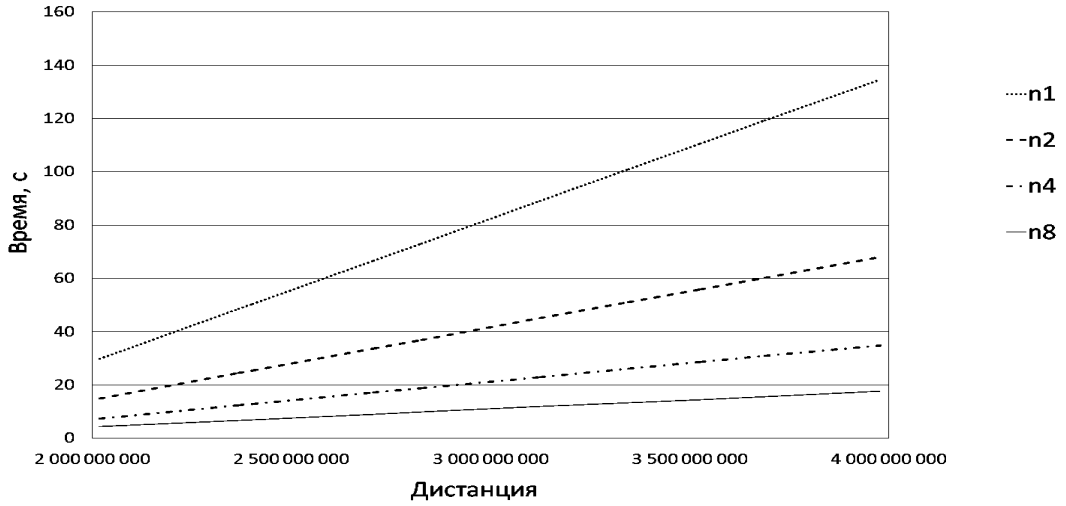

Рис. 1. Результаты измерений для метода Ферма

2.2. Метод Шермана Лемана

Как усовершенствование метода Ферма метод Шермана Лемана также существенно зависит от расстояния между множителями числа (см. рис. 2), время его выполнения линейно зависит от дистанции. Но для таких типов чисел данный алгоритм значительно проигрывает алгоритму Ферма.

Для чисел с небольшим простым делителем ситуация обратная — метод Лемана благодаря последовательным делениям на шаге один достаточно быстро выделит простой множитель (см. рис. 3).

Теоретические результаты совпадают с практическими — получена экспоненциальная зависимость времени выполнения метода Лемана от разрядности факторизуемого числа. Так как простой множитель для чисел разрядности больше 90 бит становится достаточно большим, время его выделения существенно возрастает по сравнению с числами малой разрядности.

Также в ходе исследования было получено линейное ускорение при увеличении количества процессов как для первого вида измерений, так и для второго.

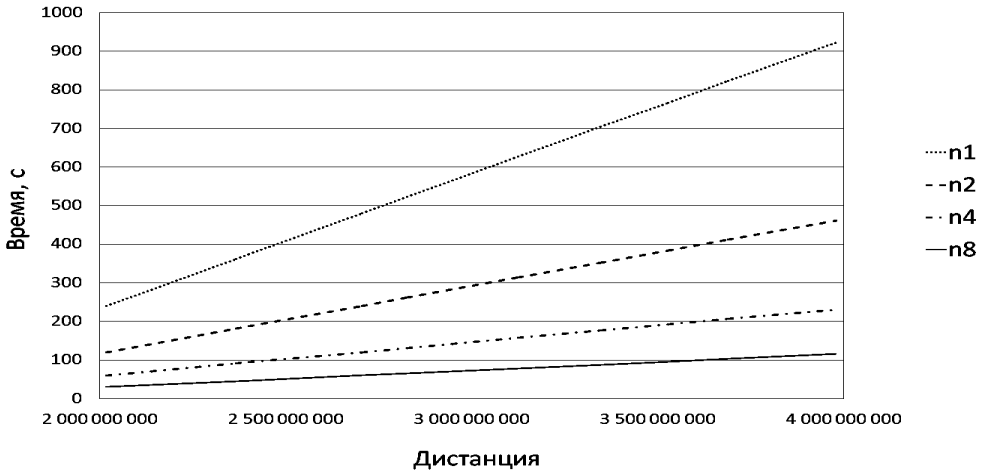

Рис. 2. Результаты измерений для метода Лемана

2.3. Метод Шенкса

Метод Шенкса не зависит от расстояния между сомножителями (см. рис. 4). Время работы данного метода для чисел с фиксированной разрядностью различно.

Для чисел второго типа была получена экспоненциальная зависимость времени работы алгоритма от разрядности (см. рис. 5). С увеличением разрядности сомножителей увеличивается время работы алгоритма, поэтому данный алгоритм подходит лишь для выделения малых простых делителей числа.

2.4. Метод ρ-Полларда

Метод Полларда не зависит от расстояния между сомножителями (см. рис. 6) и для чисел с фиксированной разрядностью выдает примерно одинаковый результат с небольшой погрешностью.

Была получена экспоненциальная зависимость времени работы алгоритма от разрядности числа (см. рис. 6). Для чисел с относительно небольшим множителем метод Полларда показывает наилучшие результаты среди алгоритмов с экспоненциальной сложностью. Линейного ускорения достичь не удалось.

2.5. Метод эллиптических кривых

С увеличением дистанции между множителями величина наименьшего простого множителя уменьшается. Так как время выполнения алгоритма факторизации с эллиптическими кривыми зависит лишь от величины этого множителя, оно постепенно уменьшается с увеличением расстояния между множителями (см. рис. 8). Фактически данные кривые – это небольшие участки экспонент (см. рис. 9).

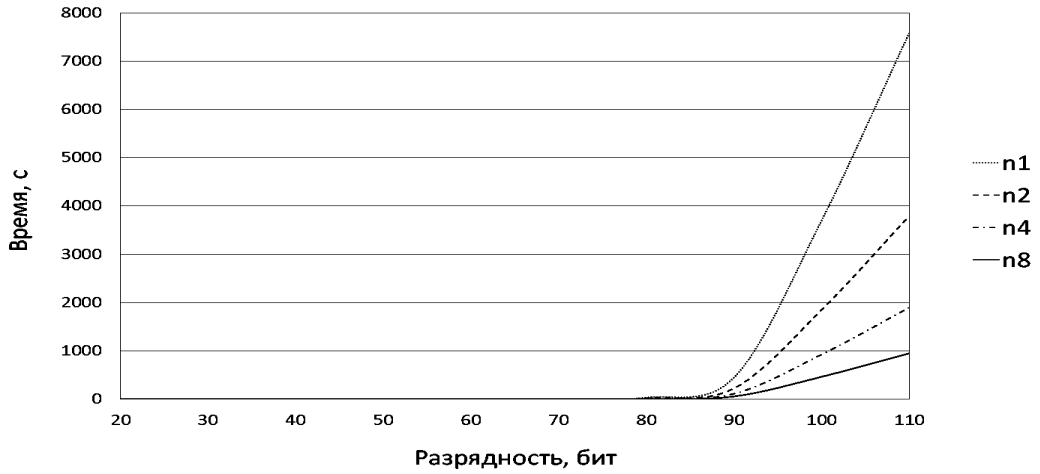

Рис. 3. Результаты измерений для метода Лемана

Ввиду того, что алгоритм вероятностный, и вычисления, связанные с нахождением нетривиального простого делителя числа, возможно, не будут распределены равномерно между процессами, в точности линейного ускорения достичь не удалось, но результаты близки к таковому (см. рис. 9). Также следует отметить, что время выполнения алгоритма значительно меньше по сравнению с любым экспоненциальным алгоритмом, рассмотренным выше.

2.6. Метод квадратичного решета

Для чисел с фиксированной разрядностью время работы алгоритма не меняется в зависимости от разрядности сомножителя (см. рис. 10).

Линейного ускорения достичь не удалось (см. рис. 11), ввиду того что метод квадратичного решета для чисел с разрядностью до 110 бит работал достаточно быстро и на одном процессе, а при распараллеливании алгоритма происходит отправка данных каждым процессом, которая влияет на время работы алгоритма.

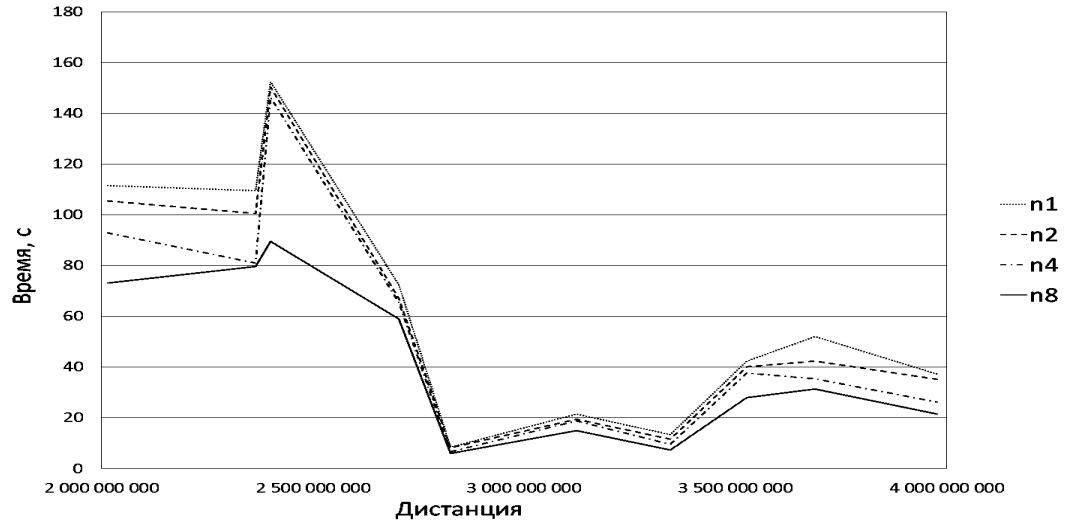

Рис. 4. Результаты измерений для метода Шенкса

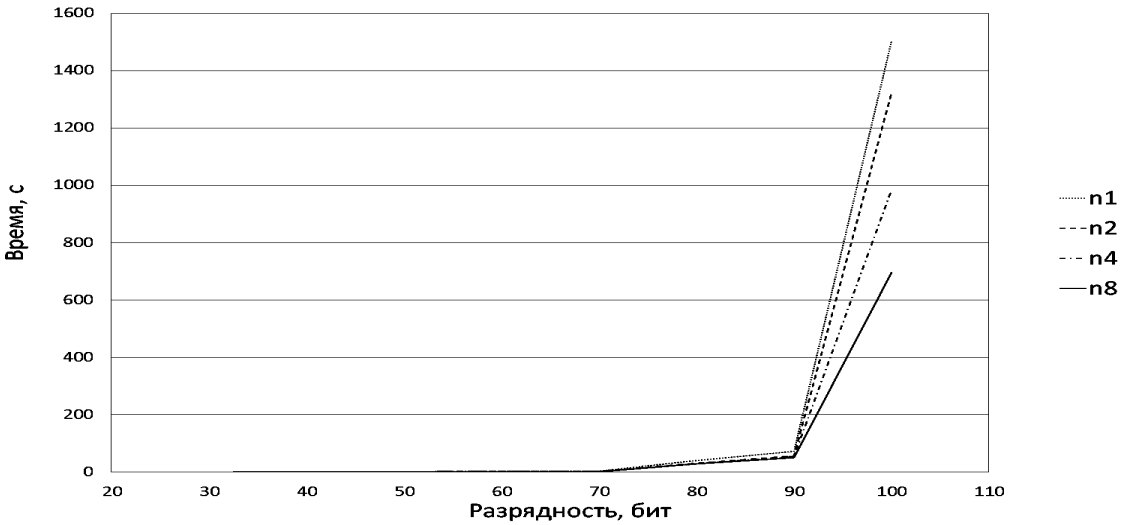

Рис. 5. Результаты измерений для метода Шенкса

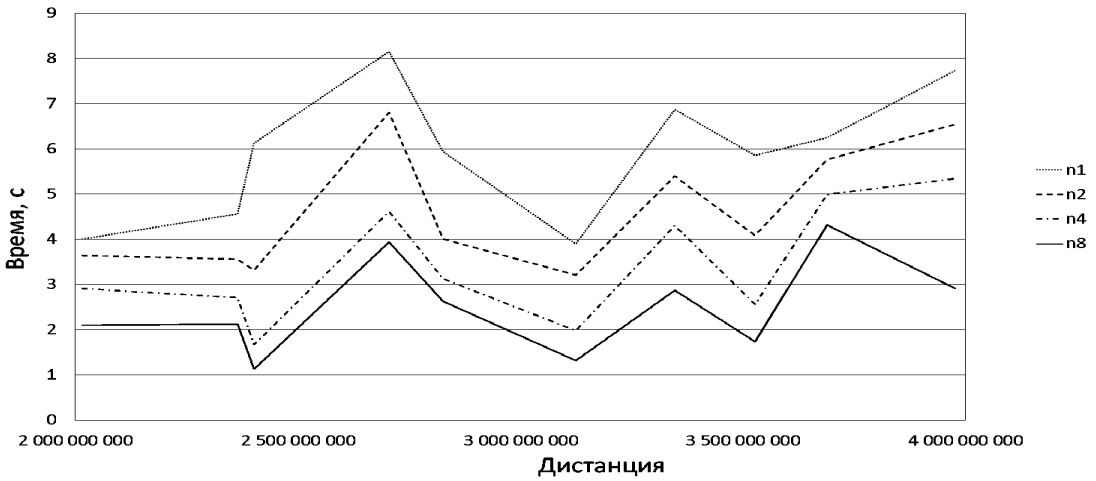

Рис. 6. Результаты измерений для метода Полларда

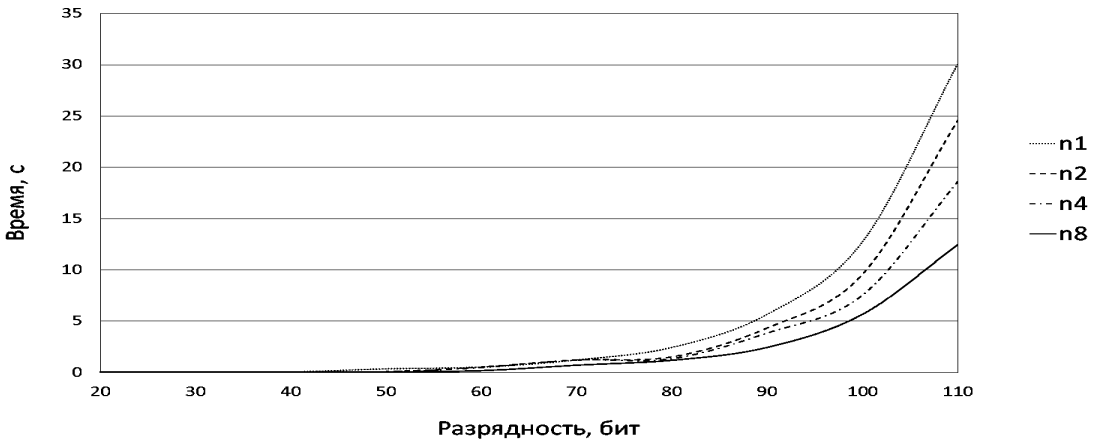

Рис. 7. Результаты измерений для метода Полларда

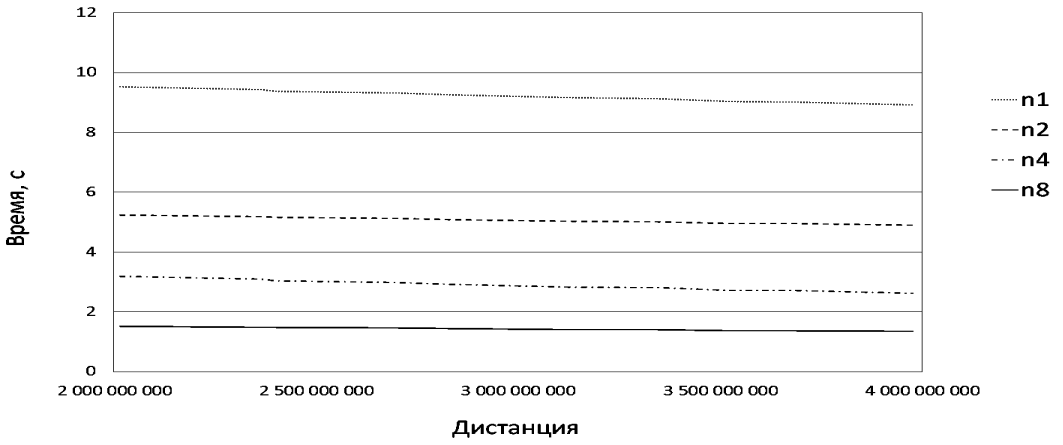

Рис. 8. Результаты измерений для метода эллиптических кривых

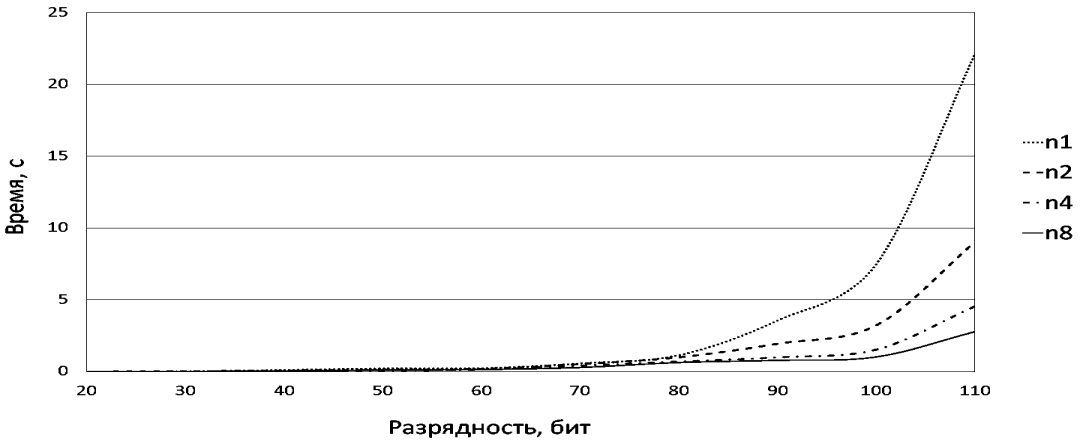

Рис. 9. Результаты измерений для метода эллиптических кривых

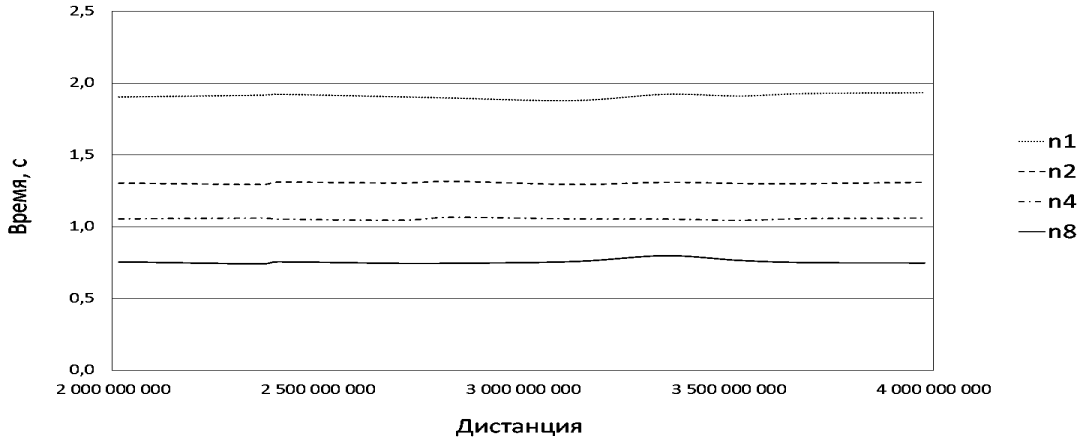

Рис. 10. Результаты измерений для метода квадратичного решета

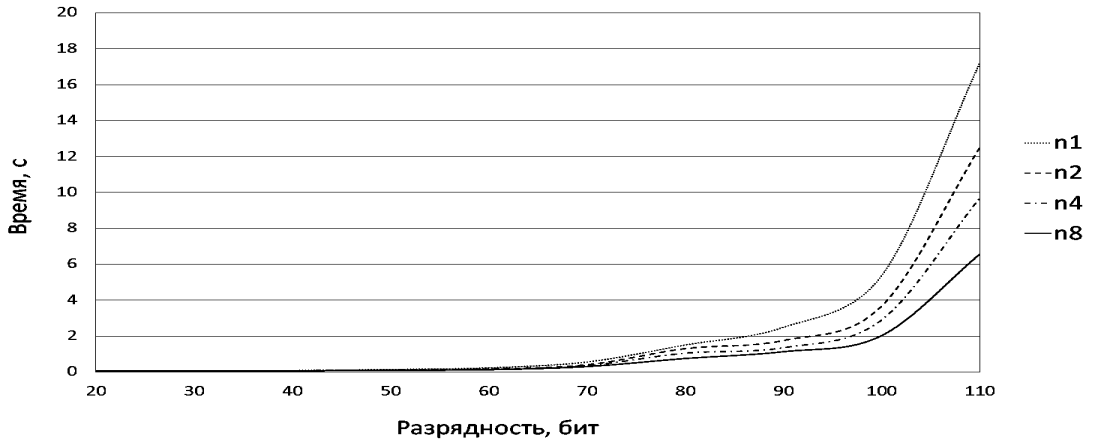

Рис. 11. Результаты измерений для метода квадратичного решета

2.7. Сравнение алгоритмов

Ввиду существенной разницы во времени выполнения экспоненциальных и субэкспоненциальных алгоритмов сравнение произведено в пределах своих категорий.

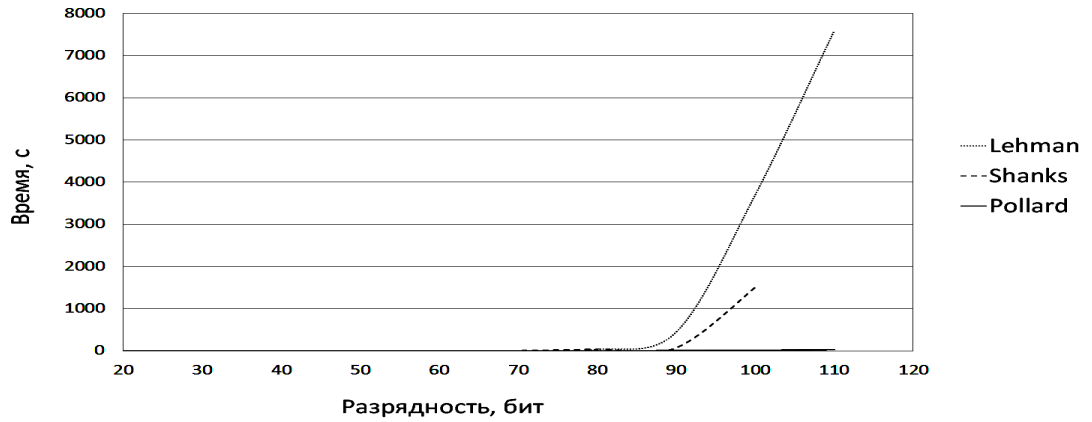

Рис. 12. Сравнение времени выполнения экспоненциальных алгоритмов

Исходя из вычислительной сложности алгоритмов, а также из практических результатов, можно сделать вывод, что в общем случае самым быстрым методом из группы экспоненциальных является метод Полларда (см. рис. 12).

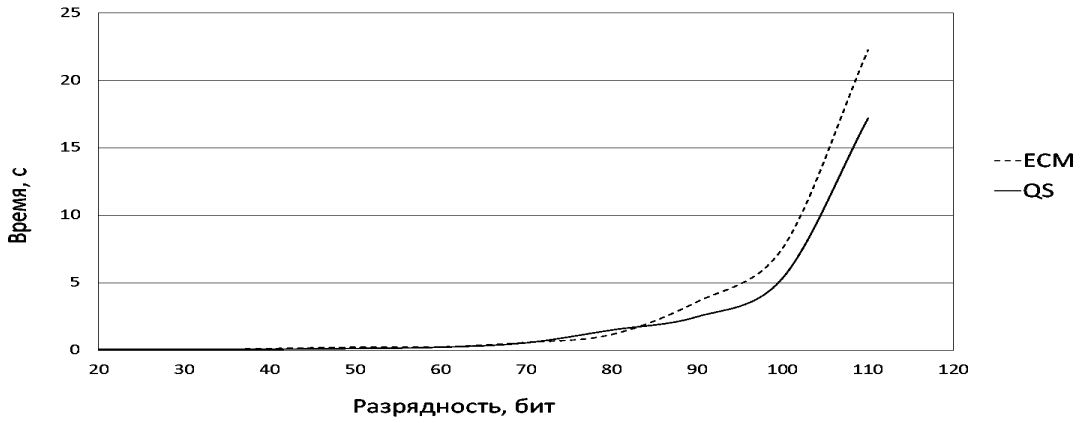

Рис. 13. Сравнение времени выполнения субэкспоненциальных алгоритмов

Очевидно, самым быстрым алгоритмом факторизации среди рассмотренных в исследовании является метод квадратичного решета (см. рис. 13). Погреш-

ность измерений, связанная с небольшим выигрышем метода эллиптических кривых на некоторых числах, исчезнет при увеличении количества измерений.

Заключение

Таким образом, в ходе исследования для большинства алгоритмов было достигнуто линейное ускорение при параллельной реализации. Также при изучении результатов работы алгоритмов было выяснено следующее: при наличии очень близких множителей наиболее эффективен метод Ферма, при относительно небольшом множителе — метод эллиптических кривых, при малых множителях — методы Шенкса и ρ-Полларда; в параллельной реализации наиболее простыми оказались методы эллиптических кривых и ρ-Полларда. В общем случае следует применять метод квадратичного решета.

Дальнейшие исследования могут строиться на основе улучшений существующих последовательных алгоритмов факторизации. Например, для метода эллиптических кривых достаточно эффективным является усовершенствование Монтгомери или использование только проективных координат, благодаря чему в алгоритме не будет использоваться ресурсоёмкое деление по модулю.

Метод квадратичного решета может быть эффективно доработан с помощью так называемой LP-стратегии. Также возможно использование различных полиномов $q(x)$ на разных процессах, участвующих в факторизации. Данные улучшения существенно уменьшат время выполнения алгоритмов.

Литература

1. Brent R.P. Some parallel algorithms for integer factorisation // Lect. Notes in Comp. Sci. 1999. V. 1685. P. 1–22.
2. Kanabar K.A. A Comparison Of Integer Factoring Algorithms. The University of Bath, 2007.
3. Lenstra H.W. Factoring integers with elliptic curves // Ann. Math. 1987. V. 126. P. 649–674
4. Василенко О.Н. Теоретико-числовые алгоритмы в криптографии. М. : МЦНМО, 2003.
5. Ишмухаметов Ш.Т. Методы факторизации натуральных чисел: учебное пособие. Казань : Казан. ун-т, 2011. 190 с.
6. Фороузан Б. А. Криптография и безопасность сетей: учебное пособие. М. : Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2010. 784 с.
7. Черемушкин А.В. Лекции по арифметическим алгоритмам в криптографии. М. : МЦНМО, 2002.