УДК 51-37+519.61
DOI: 10.25559/SITITO.15.201902.274-282

Градуированные вычисления

Р. Р. Айдагулов[1,2], **С. Т. Главацкий**[1*]
[1] Московский государственный университет имени М.В. Ломоносова, г. Москва, Россия
119991, Россия, г. Москва, ГСП-1, Ленинские горы, д. 1
[*] glavatsky_st@mail.ru
[2] Институт машиноведения РАН имени А.А. Благонравова, г. Москва, Россия
119334, Россия, г. Москва, ул. Бардина, д. 4

Аннотация

Вычисления с большим количеством данных (или данных большой размерности) обычно сводят к наборам вычислений с достаточно малыми их объемами. Одним из способов такого сведения является фильтрация, когда разбиение всего объема данных на мелкие составляющие сводится к рассмотрению ветвлений на графе (дереве). Метод фильтрации вычислений, кратко формулируемый как метод «разделяй и властвуй», часто позиционируется как наиболее эффективный метод сокращения количества операций в сложных вычислениях. Мы рассмотрим здесь более эффективный метод, метод градуированных вычислений, суть которого ранее не была опубликована.

Основное содержание нашего метода заключается в разбиении множества элементов вычисления по их значениям. Это позволяет избежать множества повторов в промежуточных вычислениях. Градуировка линейного пространства предполагает разбиение его на подпространства так же, как при представлении его в виде тензорного произведения компонент малых размерностей. Значениями в этом случае являются линейные функционалы, которые вычисляются как многочлены, сопоставляющие определенные значения базисным элементам компонентов тензорного произведения.

В задачах сортировки выигрыш скорее не так существенен, а в задачах умножения выигрыш метода градуированных вычислений проявляется явно. Даже в случае произведения больших чисел, представление многочлена от одной переменной как многочлена от нескольких переменных, соответствующее тензорному произведению пространств малых размерностей (малых степеней в компонентах), приводит к более эффективным алгоритмам, нежели алгоритм Карацубы, соответствующий вычислению с фильтрацией. Наш алгоритм позволяет выполнить вычисления за $N\log(N)$ операций при количестве данных N. Метод фильтрации приводит, вообще говоря, к N^{α} операций с экспонентой (умножения) $\alpha > 1$.

Ключевые слова: алгоритм, алгоритм Карацубы, алгоритм Штрассена, градуировка, фильтрация.

Для цитирования: *Айдагулов Р. Р., Главацкий С. Т.* Градуированные вычисления // Современные информационные технологии и ИТ-образование. 2019. Т. 15, № 2. С. 274-282. DOI: 10.25559/SITITO.15.201902.274-282

Graded Computing

R. R. Aidagulov[a,b], **S. T. Glavatsky**[a*]
[a] Lomonosov Moscow State University, Moscow, Russia
1, Leninskie gory, Moscow 119991, Russia
[*] glavatsky_st@mail.ru
[b] Institute of Mechanical Engineering RAS after A.A. Blagonravov, Moscow, Russia
4 Bardina St., Moscow 119334, Russia

Abstract

Calculations with a large amount of data (or data of a large dimension) are usually reduced to sets of calculations with fairly small amounts of them. One of the methods of such information is filtering, when splitting the entire volume of data into small components is reduced to the consideration of branching on a graph (tree). The filtering method of computations, briefly formulated as the "divide and rule" method, is often positioned as the most effective method of reducing the number of operations in complex computations. We consider here a more efficient method, the method of graded calculations, the essence of which has not been previously published.
The main content of our method is to split the set of elements of the calculation according to their values. This avoids many repetitions in intermediate calculations. Graduation of a linear space implies splitting it into subspaces in the same way as when representing it as a tensor product of components of small dimensions. In this case, the values are linear functionals, which are calculated as polynomials that associate certain values with the basis's elements of the components of the tensor product.
In sorting problems, the gain is probably not so significant, and in multiplication problems, the gain in the method of graded calculations appears explicitly. Even in the case of the multiplication of large numbers, the representation of a polynomial in one variable as a polynomial in several variables, corresponding to the tensor product of spaces of small dimensions (small degrees in the components), leads to more efficient algorithms than the Karatsuba algorithm, corresponding to the calculation with filtering. Our algorithm allows us to perform calculations for Nlog (N) operations with the amount of data N. The filtering method leads, generally speaking, to N^{α} exponential operations (multiplication) $\alpha > 1$.

Keywords: algorithm, Karatsuba multiplication algorithm, Strassen algorithm, grading, filtering.

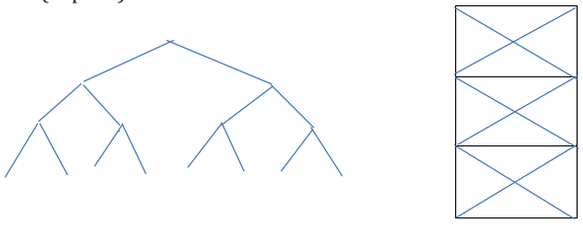
For citation: Aidagulov R.R., Glavatsky S.T. Graded Computing. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2019; 15(2):274-282. DOI: 10.25559/SITITO.15.201902.274-282

Введение. Вычисления с фильтрацией и градуированные вычисления

Вычисления с большим количеством данных (или данных большой размерности) обычно сводят к наборам вычислений с достаточно малыми их объемами. Одним из способов такого сведения является фильтрация, когда разбиение всего объема данных на мелкие составляющие сводится к рассмотрению ветвлений на графе (дереве).

Вычисления при этом начинаются с нижних листьев, результаты передаются вышерасположенному уровню, где они агрегируются (сливаются), и т.д., вплоть до самого верхнего уровня дерева и результирующей агрегации. Это хорошо описано в методе сортировки слиянием. Общеизвестная мастер-теорема относится к оценке сложности вычисления методом фильтрации (не обязательно с бинарным деревом). При умножении больших чисел (или при умножении с большой точностью) к этому методу относится алгоритм Карацубы, при умножении матриц – его аналог, алгоритм Штрассена.

Под градуированным вычислением мы понимаем вычисление по схеме, когда результаты промежуточных вычислений передаются во все продолжения градуировки по схеме из рисунка 1 ниже (справа).



Р и с. 1. Схема вычислений фильтрацией (слева) и градуированных вычислений (справа)

F i g. 1. Filtration calculation scheme (left) and graded calculations (right)

За счёт более широкого использования промежуточных вычислений повторы в вычислениях минимизируются, и достигается большая эффективность, нежели при вычислении с фильтрацией. При вычислении методом фильтрации элементами вычислений являются конечные вершины на графе – листья. Результаты промежуточных вычислений передаются только единственному верхнему узлу. При градуированном же вычислении — все пути, проходящие через один узел на каждом уровне. Результаты промежуточных вычислений передаются на продолжение градуировки (на рисунке с бинарной градуировкой) как правому, так и левому продолжению. В отличие от дерева, упорядочивание уровней носит условный характер, и их можно перенумеровать. На дереве порядок уровней определен жестко через граф связей (смотри рисунок слева). Отличие градуированного вычисления от вычисления с фильтрацией лучше объяснить на конкретных примерах.

В качестве первого примера рассмотрим задачу упорядочивания, называемую сортировкой. При решении задачи методом фильтрации множество элементов разбивается на два подмножества, они, в свою очередь, – на два подмножества поменьше, и т.д. Вычисление (упорядочивание) начинается от отдельных элементов (листьев на графе), являющихся терминальными множествами, не разбивающимися далее. В задаче сортировки метод фильтрации называется сортировкой слиянием. Имея два отсортированных подмножества с $m_1 8 m_2$ элементами (вначале для листьев $m_1 = 1 = m_2$), за $m_1 + m_2 - 1$ сравнений получаем упорядоченное (слитое) подмножество из $m_1 + m_2$ элементов. Поднимаясь по бинарному дереву, получаем упорядочение всего массива из n элементов примерно за $nlog(n)$ сравнений (логарифм без указания основания в информатике означает логарифм по основанию 2). Здесь следует указать, что одно сравнение может состоять из большого количества операций (с точки зрения процессора). Такое происходит, когда сравниваются почти идентичные тексты или числа, совпадающие на многих разрядах. В качестве примера рассмотрим массив целых чисел

$$x_i = \sum_{j=0}^{k-1} i_{k-1-j} 2^{2^j},\ i = i_0 + i_1 2 + \ldots + i_{k-1} 2^{k-1}\ \ i_j = 0 \text{ или } i_j = 1.$$

При упорядочении этого массива из $n = 2^k$ чисел в среднем приходится считывать и сравнивать $n/3$ бита с каждой парой сравниваемых чисел. Соответственно, упорядочивание выполняется за $O(n^2 \log n)$ битовых операций.

Рассмотрим теперь другой метод упорядочивания. Вначале разделим множества на положительные и отрицательные числа (определяется одним битом). Главное здесь то, что любое положительное число больше любого отрицательного. В нашем случае отрицательных чисел нет, и этот этап пропускаем. Далее разбиваем на подмножества по их порядку. Опять-таки, любое число, имеющее больший порядок, больше любого числа, имеющего меньший порядок. Далее в полученных подмножествах продолжаем разбиение по порядку, полученному обнулением основного бита, определяющего их порядок, и т.д. Полученные новые значения чисел могут иметь совершенно разные порядки и могут опять разбиваться на подклассы по новому порядку. Если порядки чисел одинаковые, то разбиваем их по значению следующего определяющего бита. Так мы действительно получим упорядочивание за $O\left(nlog\left(n\right)\right)$ операций, по крайней мере, для упорядочивания указанного массива. Преимуществом этого способа является разбиение упорядочиваемого множества на подмножества по значениям их элементов. Подмножества с близкими значениями разбиваются на малые подмножества после вычета одного из определяющих битов, или по тому, насколько имеется отличие от одного из элементов. При этом убирается сравнение множества пар по неотличающимся битам. Пусть числа задаются m битами. Тогда количество выполняемых битовых операций равно $O\left(\left(n+m\right)\log\left(n\right)\right)$ операций. В предыдущем случае было $O\left(nm\log\left(n\right)\right)$ операций.

Этот метод во многом сходен с первой реализованной одним из авторов подпрограммой упорядочивания. Тогда за $O\left(n\right)$ операций находилось минимальное и максимальное значение в массиве чисел. Далее по их значениям определялся номер корзины:

$$j\left(i\right) = \left[\frac{(x_i - x_{min})m}{x_{max} - x_{min}}\right].$$

Когда данные распределены равномерно, и $m \sim 2n$, то номер корзины уже определяет порядок числа без всяких сравнений. В случае неравномерности распределения требуется доработать вышеуказанным способом. Номер корзины (полученный по значению) представляет градуировку элемента. Существенным здесь является упомянутое выше свойство разбиения по значениям: если номер *j(i)>j(k),* то $x_i > x_k$ без всякого сравнения. Разбиение по значениям ассоциируется с инте-

гральной суммой интеграла Лебега, в то время как сортировка слиянием ассоциируется с интегральной суммой интеграла Римана. Допустим, значения x_i равны 0 или 1, как у характеристической функции некоторого подмножества. Тогда в нашем случае, как и при вычислении интегральной суммы Лебега, мы сразу отделяем это подмножество по значениям 0 или 1. В последнем случае (сортировка слиянием) упорядоченные множества расширяются за счет слияния нулевых значений с нулевыми, единичных с единичными, как при интегральной сумме Римана.

В качестве второго примера рассмотрим умножение больших чисел. Перемножаемые числа записываются в системе исчисления с основанием $N = 2^l$: $x = x_0 + x_1 N + \ldots + x_{k-1} N^{n-1}, y = y_0 + y_1 N + \ldots + y_{k-1} N^{n-1}$. Здесь количество цифр у обоих множителей *n*. Это не принципиально, всегда можно при необходимости дополнить нулевыми старшими цифрами. При умножении методом фильтрации (метод Карацубы для умножения больших чисел) числа разделяются на два примерно одинаковых числа $x = X_1 + X_2 M, y = Y_1 + Y_2 M$. Произведение сводится к произведению трех пар чисел вдвое меньшего разряда:

$$xy = X_1 Y_1 + \left(X_1 Y_2 + X_2 Y_1\right) M + X_2 Y_2 M^2, X_1 Y_2 + X_2 Y_1$$
$$= \left(X_1 + X_2\right)\left(Y_1 + Y_2\right) - X_1 Y_1 - X_2 Y_2.$$

Таким образом, произведение *n*-разрядных чисел вычисляется за $O\left(n^{\log(3)}\right)$ операций. Градуированному вычислению произведения соответствует умножение больших чисел методом Кули-Тьюки с использованием дискретного преобразования Фурье. Умножение чисел сводится к умножению многочленов с коэффициентами, соответствующими коэффициентам в первоначальном представлении. Количество разрядов *n* увеличивается до необходимого уровня, чтобы их произведение так же могло быть записано как *n*-разрядное число. При необходимости *n* увеличивается до такого значения, чтобы оно имело разложение на малые множители $n = p_1 p_2 \ldots p_k$. Градуировка определяется, по сути, этим разложением и значениями многочленов соответствующих значений. Умножение многочленов вычисляется через произведения их значений. Остается эффективно вычислить значения многочлена в корнях из 1. Пусть $n = pm$. Тогда вычисление *n* значений на корнях степени *n* для многочлена:

$$f\left(x\right) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$$

сводится к вычислению *m* значений для p многочленов степени *m*:

$$f_0\left(x^p\right) = a_0 + a_p x^p + \ldots + a_{(m-1)p} x^{(m-1)p},$$
$$f_1\left(x^p\right) = a_1 + a_{p+1} x^p + \ldots + a_{(m-1)p+1} x^{(m-1)p},$$

……,

$$f_{p-1}\left(x^p\right) = a_{p-1} + a_{2p-1} x^p + \ldots + a_{(m-1)p+p-1} x^{(m-1)p}.$$

По этим вычисленным величинам значения нашего многочлена получаются по формуле:

$$f\left(x\right) = f_0\left(x^p\right) + x f_1\left(x^p\right) + \ldots x^{p-1} f_{p-1}\left(x^p\right).$$

Здесь нам потребуется дополнительно складывать *(p-1)n* раз и умножать примерно столько же раз (на самом деле чуть меньше, при вычислении случая $E = 1$ обходимся без умножения). Таким образом, количество $T\left(n\right)$ операций для преобразования Фурье на *n* точках удовлетворяет рекуррентному соотношению

$$T\left(n\right) = pT\left(\frac{n}{p}\right) + \left(p-1\right)n.$$

Когда все множители малые, то $T\left(n\right) \approx O\left(nlog\left(n\right)\right)$ операций.

Если сделать оценку более точной, то для преобразования Фурье с $n = p_1 p_2 \ldots p_k$ точками требуется выполнить N умножений и N сложений, где

$$N = p_1 p_2 \ldots p_k \left(p_1 + p_2 + \ldots p_k - k\right) = n\left(p_1 + p_2 + \ldots + p_k - k\right).$$

Для поиска минимального $N\left(n\right)$ следует найти минимум этого выражения при условии $p_1 p_2 \ldots p_k \geq n$. Например, для *n=45* величина $N = 360$ меньше, чем в случае $p_i = 2, k = 6, n = 64$, *N(64)=384*. Если *n=81*, то величина *N=648* меньше чем $N\left(96\right) = 672 < 896 = N\left(128\right)$.

Заметим, что корни из 1 в целочисленном варианте берутся из колец Z_q таких, для которых q – простое, и *q-1* делится на *n*. Обратное преобразование совпадает по своему виду с преобразованием Фурье, осуществляется совершенно аналогично и занимает примерно такое же количество операций, что и само преобразование Фурье, так как отличие состоит только в том, что вместо корня ω используется корень ω^{-1} и общий множитель для всех $\frac{1}{n}$.

Таким образом, достигается эффективность, пропорциональная объему данных n с коэффициентом $\log\left(n\right)\log log\left(n\right)$ на каждое данное. Последний множитель связан с увеличением длины представления коэффициентов произведения при увеличении степени многочлена.

Заметим, что в последним случае мы опять прибегли к вычислению произведения через значения, и получили результат примерно за $nlog\left(n\right)$ операций, что характерно для градуированных вычислений. Градуированность вычисления здесь проявляется в умножении многочленов по модулю многочлена $x^n - 1$, с введением промежуточных переменных $x_1 = x, x_2 = x_1^{p_1}, \ldots, x_k = x_{k-1}^{p_{k-1}}, x_k^{p_k} = 1$. Градуированность вычисления демонстрируется более наглядно в случае умножения многочленов от *k* переменных $x_1, x_2, \supset, x_k$ с независимыми соотношениями $x_i^{p_i} = 1$. Представителем градуированного элемента является набор степеней $\left(i_1, i_2, \ldots, i_k\right) : x_1^{i_1} x_2^{i_2} \ldots x_k^{i_k}, 0 \leq i_j < p_j$. Градуированный элемент является тензорным произведением градуированных элементов с каждого из *k* уровней. Если все $p_i = p$, то количество базисных градуированных элементов равно $p^k = n$ (растет по экспоненциальному закону от количества уровней *k*). Вычисление значения производится с переходом с одного уровня на другой за $O\left(nk\right) = O\left(n log\, n\right)$ операций. Хотя при вычислении произведения величины x_i не являются независимыми, все равно вычисления производятся по канонам градуированного вычисления, описанного ранее.

Градуированные вычисления произведения матриц

Впервые более быстрый, нежели стандартный, алгоритм умножения матриц был разработан в 1969г. Штрассеном [1]. Этот алгоритм является аналогом алгоритма Карацубы, и имеет сложность $O\left(n^{\log 7}\right)$ при умножении квадратных матриц размера $n\times n$. В работе[1] [2] рассмотрены почти все методы эффективного вычисления произведения матриц. Они сводятся к оценке ранга умножения и вычисления экспоненты умножения $\alpha=\log_n r\left(n\right)$ для алгебры матриц порядка $n\times n$ с рангом умножения $r\left(n\right)$. Наилучшие оценки экспоненты умножения получены Виноградом и Копперсмитом в [3]. Этот метод более развёрнуто описан в [4]. При этом ранг в алгебре тензорного произведения оценивается через ранги алгебр сомножителей. Так, оценены ранги произведения матриц порядков, являющихся некоторыми степенями 2. Виноградом и Копперсмитом получена оценка с $\alpha=2.373,$ и в работе [5] с помощью компьютерных вычислений их же методом оценка уточнена для порядка $n=2^8=256$ до $\alpha=2.3728639$.

Следует отметить, однако, что в этом методе оценки ранга имеется следующий изъян (недостаток): экспонента умножения для тензорного произведения получается больше $log_2 5 \oplus 2.321928$, если экспоненты умножения для сомножителей больше этого числа. При этом отсутствует эффективный способ вычисления значений функционалов, как при преобразовании Фурье. Это приводит к нереально большим коэффициентам для количества операций даже при значениях $n<10^9$. При больших значениях *n* количество операций и ресурсов памяти таковы, что вычисления невыполнимы и на самых мощных суперкомпьютерах. Действительно, если нет эффективного способа вычисления функционалов, то количество операций при умножении матриц порядков $n=m^k$ оценивается величиной $n^{\alpha_m+\frac{1}{k}}$. В методе Штрассена функционалы и постоянные матрицы более разреженные, и поэтому количество операций получается несколько меньше этой оценки. Однако и здесь это вычисление произведения эффективнее стандартного метода лишь начиная с *k*>9, и не более чем в 2 раза, пока *k*<14. Для лучшей оценки на сегодня с $\alpha=2.3728639, m=256$ такое вычисление произведения эффективнее стандартного вычисления (с n^3 произведениями) только при *k*>3. Когда k≥4, m=256 размер *n* матриц не меньше недостижимого значения даже для суперкомпьютеров: $n=2^{32}$.

Как видно из этого краткого обзора, все успехи в быстром вычислении произведения матриц связаны с методом фильтрации. Успех в градуированном методе вычисления сразу привел бы к алгоритму со сложностью $O\left(n^2 log^d n\right)$, с *d=1* или *d=2*. Настоящая работа задумана как теоретическая статья для описания такого алгоритма. Ранее одним из авторов были опубликованы две алгебраические статьи на эту тему [6,7]. Они, как и настоящая статья, представляют и самостоятельный интерес. В них вводится структура бигрупповой алгебры на алгебре матриц, являющейся алгеброй некоммутативных многочленов от переменных x_i, y_i с соотношениями:

$$x_i^{n_i}=1=y_i^{n_i}, x_i y_i=\theta_i y_i x_i,$$

где θ_i – примитивный корень степени n_i, а переменные разного уровня (с разными индексами) между собой коммутируют. Как и в случае умножения больших чисел, более удобной является структура с $n_i=2, \theta_i=-1$. При k уровнях получаем алгебру квадратных матриц $n\times n, n=2^k$. Умножение прямоугольных матриц сводится к умножению в такой алгебре при некотором k (при расширении матриц нулевыми элементами). Эта алгебра является *k*-й тензорной степенью алгебры матриц второго порядка.

Рассмотрим вначале подробнее структуру этой алгебры. Указанный базис состоит из следующих элементов:

$$1=\begin{pmatrix}1&0\\0&1\end{pmatrix}, x=\begin{pmatrix}1&0\\0&-1\end{pmatrix}, y=\begin{pmatrix}0&1\\1&0\end{pmatrix}, z=xy=-yx=\begin{pmatrix}0&1\\-1&0\end{pmatrix}.$$

Переменные x, y, z являются антикоммутирующими. В случае базиса Вейля-Швингера [6], получающегося в этом случае заменой z на $iz, где\, i^2=-1$ (*i* – мнимая единица), переменные *x, y, z* симметричны относительно циклической перестановки как мнимые антикоммутирующие единицы кватернионов. В качестве образующих алгебры можно принять любую пару из тройки переменных x, y, z. В качестве базиса e_0, e_1, e_2, e_3 алгебры можно принять любые линейные комбинации исходного базиса, составляющие базис линейного пространства. Любой линейный функционал $f: A\to R$ на алгебре определяется по формуле:

$$f\left(a_0e_0+a_1e_1+a_2e_2+a_3e_3\right)=a_0\lambda_0+a_1\lambda_1+a_2\lambda_2+a_3\lambda_3, a_i, \lambda_i\in R.$$

По сути, его значения получаются, когда базисным элементам e_i сопоставляют числа λ_i. При представлении элементов алгебры в другом базисе:

$$e_i=\sum_j c_i^j e'_j$$

новые координаты представления вычисляются по формулам:

$$a'_j=\sum_i a_i c_i^j.$$

Новые координаты функционала связаны со старыми по формулам:

$$\lambda_i=\sum_j c_i^j\lambda'_j, \lambda'_j=\sum_i d_j^i\lambda_i, \sum_j c_i^j d_j^k=\delta_i^k.$$

В итоге получается, что функционалы задают значения или коэффициенты в разложении элемента алгебры в некотором (сопряженном) базисе.

При умножении матриц надо уметь представлять их в разных базисах. Матрицы задаются обычно в таком базисе:

$$A=a_{00}\begin{pmatrix}1&0\\0&0\end{pmatrix}+a_{01}\begin{pmatrix}0&1\\0&0\end{pmatrix}+a_{10}\begin{pmatrix}0&0\\1&0\end{pmatrix}+a_{11}\begin{pmatrix}0&0\\0&1\end{pmatrix}.$$

Базис $e_0=1, e_1=x, e_2=y, e_3=z$ назовем стандартным. Координаты матрицы в стандартном базисе имеют вид:

$$a_0=\frac{a_{00}+a_{11}}{2}, a_1=\frac{a_{00}-a_{11}}{2}, a_2=\frac{a_{01}+a_{10}}{2}, a_3=\frac{a_{01}-a_{10}}{2}.$$

Одним из удобных базисов является базис с проекторами:

1 *Holtz O.* Fast and stable matrix multiplication [Электронный ресурс]. URL: https://people.eecs.berkeley.edu/~oholtz/Talks/mit.pdf (дата обращения: 6.05.2019).

$1, p=\frac{1-x}{2}, e=\frac{(1-x)(1-y)}{2}, t=\frac{(1+x)(1-y)}{2}.$

Проверим, что оператор $z=\frac{1+\varepsilon_1 x+\varepsilon_2 y+\varepsilon_3 xy}{2}$ является проектором, учитывая значения попарных антикоммутирующих произведений и квадратов $x^2=1=y^2=-(xy)^2$:

$$z^2=\frac{1+\varepsilon_1^2+\varepsilon_2^2-\varepsilon_3^2}{4}+\frac{\varepsilon_1 x+\varepsilon_2 y+\varepsilon_3 xy}{2}=z, \varepsilon_1^2+\varepsilon_2^2-\varepsilon_3^2=1.$$

Здесь члены с $\varepsilon_i\varepsilon_j$, $i\neq j$ исчезают из-за антикоммутирования переменных. Это доказывает, что введенные переменные и их дополнения $\overline{p}=1-p, \overline{e}=1-e, \overline{t}=1-t$ являются проекторами. Представляя первый сомножитель в базисе $1, p, e, t$, а другой сомножитель – в базисе $1, \overline{p}, \overline{e}, \overline{t}$, получим результат в виде линейной комбинации семи операторов $1, p, e, t, \overline{p}, \overline{e}, \overline{t}$.

$$C=AB=tr(A)tr(B)*1-A(0,1,1)B(0,0,0)p-$$
$$-A(1,1,1)B(1,0,0)\overline{p}-A(1,0,0)B(0,0,1)e-$$
$$-A(1,1,0)B(0,1,1)\overline{e}-A(0,0,0)B(1,1,0)t-$$
$$-A(0,0,1)B(1,1,1)\overline{t}.$$

Здесь $A(a,b,c), B(a,b,c)$ – значения (функционалов) в базисе 1,*p,e,t*, когда вместо операторов подставляются числа:

$1-1, p-a, e-b, t-c.$

В стандартном базисе $1, x, y, z=xy$ это соответствует:

$$tr(A)=f^7=2f_0=2f(0,0,0)=a_{00}+a_{11},$$
$$f^1=A(1,1,1)=f(-1,-1,-1)=a_{11}-a_{01},$$
$$f^2=A(1,0,0)=f(-1,1,-1)=a_{11}+a_{10},$$
$$f^3=A(1,1,0)=f(-1,0,0)=a_{11},$$
$$f^4=A(0,0,1)=f(1,0,0)=a_{00},$$
$$f^5=A(0,1,1)=f(1,-1,1)=a_{00}-a_{10},$$
$$f^6=A(0,0,0)=f(1,1,1)=a_{00}+a_{01}.$$

Обозначим соответствующие значения через φ^i, а для произведения – через ψ^i. Произведение определяется четырьмя значениями, получаемыми по формулам:

$$\psi^1=f^7\varphi^7-f^2\varphi^4-f^6\varphi^3-f^5\varphi^6,$$
$$\psi^2=f^7\varphi^7-f^3\varphi^5-f^4\varphi^1-f^5\varphi^6,$$
$$\psi^3=f^7\varphi^7-f^2\varphi^4-f^4\varphi^1-f^5\varphi^6,$$
$$\psi^4=f^7\varphi^7-f^3\varphi^5-f^6\varphi^3-f^1\varphi^2.$$

Эти функционалы удовлетворяют условию дополнения:

$$f^i+f^{7-i}=f^7, i=1,2,3,4,5,6. \quad (1)$$

Заметим, что встречаются только 7 произведений $f^i\varphi^{j(i)}$, где

$$j(7)=7, j(i)=2^{\left(\frac{i}{7}\right)} imod 7, j(1)=2, j(2)=4, j(4)=1,$$
$$j(6)=3, j(3)=5, j(5)=6.$$

Элементы произведения матриц находятся из формул:

$$c_{00}=\psi^4, c_{11}=\psi^3, c_{01}=\psi^3-\psi^1, c_{10}=\psi^2-\psi^3.$$

Это задает вычисление произведение двух матриц второго порядка.

Множество значений, по которым можно восстановить элементы матрицы, назовем полным. Покажем, что полный комплект значений (функционалов) вычисляются за $O(4^k k)=O(n^2\log(n))$ операций, как и в случае значений коммутативных многочленов, переходя с одного уровня на другой. Полный комплект значений составляют 4^k значений $f^{i_1 i_2\ldots i_k}, 1\le i_j\le 4$. Они вычисляются от коэффициентов матрицы за k шагов, переходя с одного уровня на другой. На первом шаге вычисляем:

$$f^1_{I_1J_1}=a_{2I_1+1,2J_1+1}-a_{2I_1,2J_1+1},$$
$$f^2_{I_1J_1}=a_{2I_1+1,2J_1+1}+a_{2I_1+1,2J_1},$$
$$f^3_{I_1J_1}=a_{2I_1+1,2J_1+1},$$
$$f^4_{I_1J_1}=a_{2I_1,2J_1},$$

На следующем этапе вычисляем:

$$f^{i_1 1}_{I_2J_2}=f^{i_1}_{2I_2+1,2J_2+1}-f^{i_1}_{2I_2,2J_2+1},$$
$$f^{i_1 2}_{I_2J_2}=f^{i_1}_{2I_2+1,2J_2+1}+f^{i_1}_{2I_2+1,2J_2},$$
$$f^{i_1 3}_{I_2J_2}=f^{i_1}_{2I_2+1,2J_2+1},$$
$$f^{i_1 4}_{I_2J_2}=f^{i_1}_{2I_2,2J_2}.$$

Далее реализуем третий, ..., k-й этап. Количество операций при вычислении составляет $\frac{kn^2}{4}$ сложений и столько же вычитаний. Имея такой полный комплект значений, коэффициенты матрицы (произведения) вычисляем за $O(n^2k)$ операций.

Полный комплект n^2 значений произведения матриц можно вычислить через N пар произведений различных значений сомножителей, переходя с одного уровня на другой. При одном типе организации вычислений $N=n^2\left(1+\frac{3k}{4}\right)$. В этом случае общее количество операций для вычисления произведения не превосходит $O(n^2 log^2 n)$ операций. Можно вычислить экономнее, обойдясь всего N умножениями, где $N=4^k+3*4^{k-1}+3^2*4^{k-2}+\ldots+3^k=4^{k+1}-3^{k+1}<4n^2$. Здесь к 4^k умножениям первого уровня добавляется 3 умножения для каждого продолжения градуировки. Далее к поправочным произведениям в количестве $3*4^{k-1}$ потребуется еще $3*3*4^{k-2}$ дополнительных умножений и т.д. Общее количество операций при этом равно $O(n^2\log(n))$.

В работе [6] высказывается, что ранг умножения может быть $2n^2-1$. Однако при такой экономии в количестве функциональных произведений мы теряем в количестве суммирований и умножений на коэффициенты. При вычислении любого полного базиса значений произведения, каждое значение в среднем состоит из суммы не менее n произведений значе-

ний. При этом не удается объединить их в последовательные сложения и умножения на коэффициенты, как описано выше, в *k* шагов. Из-за описанного в [6] разложения $(n-1)(n+1)$ алгебры автоморфизмов, в коэффициентах появятся комплексные корни соответствующих степеней. Поэтому, экономия в функциональных произведениях приводит к еще большему количеству умножений на коэффициенты. Соответственно, сложность вычисления только увеличится. Исходя из таких соображений, авторы не стали разрабатывать до конца теорию о ранге умножения. Больше информации о современных достижениях в построении эффективных алгоритмов умножения можно найти в работах [7–20].

Скажем немного о методе фильтрации при умножении матриц. В этом случае размер $n = m^k$. Пусть для умножения матриц порядка $m \times m$ получены формулы:

$$! _i = \sum_{j=1}^{r(m)} \alpha_{ij} f_j \varphi_j .$$

Количество операций на одной ветке составит (без сокращения, как в случае градуированного вычисления) $O(r(m)\log n)$ операций для вычисления попарных произведений $f_j\varphi_j$ и $m^2(r(m)-1)$ сложений и $m^2 r(m)$ умножений на коэффициенты. Для количества операций при умножении матриц $n \times n$ получаем рекуррентное соотношение:

$$T(m^k) = r(m)T(m^{k-1}) +$$
$$+m^{2k}\left[r(m) \text{ (умножений на коэффициенты)} + \right.$$
$$\left. +(r(m)-1) \text{ сложений} \right].$$

Как и в мастер-теореме, получаем: $r(m)m^2 \frac{r(m)^k - m^{2k}}{r(m) - m^2} = \theta n^{\log_m r(m) + \frac{2}{k}}$ умножений, и примерно столько сложений. За счет величины $\frac{2}{k}$ в показателе, получаем, что количество операций будет меньше n^3 только при больших *n*, для некоторых формул это – при $n > 10^9$. При таких размерностях даже самые мощные суперкомпьютеры пока бессильны.

Заключение

В работе описывается относительно новый тип организации вычислений, названной градуированным вычислением, опирающийся на градуировку данных, и позволяющий выполнить вычисления за $Nlog(N)$ операций, при количестве данных N. Этот метод сравнивается с распространенным методом «разделяй и властвуй», приводящим, вообще говоря, к N^α операций с экспонентой (умножения) $\alpha > 1$.

Список использованных источников

[1] *Strassen V.* Gaussian elimination is not optimal // Numerische Mathematik. 1969. Vol. 13, Issue 4. Pp. 354-356. DOI: 10.1007/BF02165411

[2] *Demmel J., Dumitriu I., Holtz O.* Fast linear algebra is stable // Numerische Mathematik. 2007. Vol. 108, Issue 1. Pp. 59-91. DOI: 10.1007/s00211-007-0114-x

[3] *Coppersmith D., Winograd S.* Matrix multiplication via arithmetic progressions // Journal of Symbolic Computation. 1990. Vol. 9, Issue 3. Pp. 251-280. DOI: 10.1016/S0747-7171(08)80013-2

[4] *Жданович Д. В.* Экспонента сложности матричного умножения // Фундаментальная и прикладная математика. 2011/2012. Т. 17, № 2. С. 107-166. URL: https://elibrary.ru/item.asp?id=23616768 (дата обращения: 6.05.2019).

[5] *François Le Gall.* Powers of tensors and fast matrix multiplication // Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC '14) / Katsusuke Nabeshima (eds). ACM, New York, NY, USA, 2014. Pp. 296-303. DOI: 10.1145/2608628.2608664

[6] *Айдагулов Р. Р.* Бигрупповые алгебры и их автоморфизмы // Дневник науки. 2019. № 1(25). С. 25. URL: https://elibrary.ru/item.asp?id=36901640 (дата обращения: 6.05.2019).

[7] *Айдагулов Р. Р.* Бигрупповые алгебры и квантовая комбинаторика // Дневник науки. 2019. № 1(25). С. 26. URL: https://elibrary.ru/item.asp?id=36901641 (дата обращения: 6.05.2019).

[8] *Harvey D., van der Hoeven J.* Integer multiplication in time O(*n* log *n*). 2019. hal-02070778. URL: https://hal.archives-ouvertes.fr/hal-02070778/file/nlogn.pdf (дата обращения: 6.05.2019).

[9] *Harvey D.* Faster truncated integer multiplication // CoRR. 2017. Vol. abs/1703.00640. URL: http://arxiv.org/abs/1703.00640 (дата обращения: 6.05.2019).

[10] *Harvey D., van der Hoeven J.* Faster integer and polynomial multiplication using cyclotomic coefficient rings // CoRR. 2017. Vol. abs/1712.03693. URL: http://arxiv.org/abs/1712.03693 (дата обращения: 6.05.2019).

[11] *Harvey D., van der Hoeven J.* On the complexity of integer matrix multiplication // Journal of Symbolic Computation. 2018. Vol. 89. Pp. 1-8. DOI: 10.1016/j.jsc.2017.11.001

[12] *Harvey D., van der Hoeven J.* Faster integer multiplication using plain vanilla FFT primes // Mathematics of Computation. 2019. Vol. 88. Pp. 501-514. DOI: 10.1090/mcom/3328

[13] *Harvey D., van der Hoeven J.* Faster integer multiplication using short lattice vectors // ANTS XIII. Proceedings of the Thirteenth Algorithmic Number Theory Symposium / R. Scheidler, J. Sorenson (eds). Mathematical Sciences Publishers, Berkeley, 2019, pp. 293-310. DOI: 10.2140/obs.2019.2.293

[14] *Harvey D., van der Hoeven J.* Faster polynomial multiplication over finite fields using cyclotomic coefficient rings // Journal of Complexity. 2019. Vol. 54. 101404. DOI: 10.1016/j.jco.2019.03.004

[15] *Harvey D., van der Hoeven J.* Polynomial multiplication over finite fields in time O(*n* log *n*). 2019. hal-02070816. URL: https://hal.archives-ouvertes.fr/hal-02070816 (дата обращения: 6.05.2019).

[16] *Harvey D., van der Hoeven J., Lecerf G.* Even faster integer multiplication // Journal of Complexity. 2016. Vol. 36. Pp. 1-30. DOI: 10.1016/j.jco.2016.03.001

[17] *Harvey D., van der Hoeven J., Lecerf G.* Faster Polynomial Multiplication over Finite Fields // Journal of the ACM. 2017. Vol. 63, Issue 6, Article 52. 23 p. DOI: 10.1145/3005344

[18] *van der Hoeven J.* Faster relaxed multiplication //

Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC '14) / Katsusuke Nabeshima (ed). ACM, New York, NY, USA, 2014. Pp. 405-412. DOI: 10.1145/2608628.2608657

[19] *Brent R. P., Zimmermann P.* Modern Computer Arithmetic, Cambridge Monographs on Applied and Computational Mathematics. Vol. 18. Cambridge University Press, Cambridge, 2010. 236 p.

[20] *De A., Kurur P., Saha C., Saptharishi R.* Fast Integer Multiplication Using Modular Arithmetic // SIAM Journal on Computing. 2013. Vol. 42, Issue 2. Pp. 685-699. DOI: 10.1137/100811167

Поступила 6.05.2019; принята к публикации 10.07.2019; опубликована онлайн 25.07.2019.

Об авторах:

Айдагулов Рустем Римович, старший научный сотрудник кафедры теоретической информатики, отделение математики, механико-математический факультет, Московский государственный университет имени М.В. Ломоносова (119991, Россия, г. Москва, ГСП-1, Ленинские горы, д. 1), Институт машиноведения РАН имени А.А. Благонравова (119334, Россия, г. Москва, ул. Бардина, д. 4), кандидат физико-математических наук, ORCID: http://orcid.org/0000-0002-6579-429X, a_rust@bk.ru

Главацкий Сергей Тимофеевич, доцент кафедры теоретической информатики, отделение математики, механико-математический факультет, Московский государственный университет имени М.В. Ломоносова (119991, Россия, г. Москва, ГСП-1, Ленинские горы, д. 1), кандидат физико-математических наук, доцент, ORCID: http://orcid.org/0000-0003-1857-6158, glavatsky_st@mail.ru

Все авторы прочитали и одобрили окончательный вариант рукописи.

References

[1] Strassen V. Gaussian elimination is not optimal. *Numerische Mathematik*. 1969; 13(4):354-356. (In Eng.) DOI: 10.1007/BF02165411

[2] Demmel J., Dumitriu I., Holtz O. Fast linear algebra is stable. *Numerische Mathematik*. 2007; 108(1):59-91. (In Eng.) DOI: 10.1007/s00211-007-0114-x

[3] Coppersmith D., Winograd S. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation.* 1990; 9(3):251-280. (In Eng.) DOI: 10.1016/S0747-7171(08)80013-2

[4] Zhdanovich D.V. The matrix capacity of a tensor. *Journal of Mathematical Sciences.* 2012; 186(4):599-643. (In Eng.) DOI: 10.1007/s10958-012-1009-7

[5] François Le Gall. Powers of tensors and fast matrix multiplication. In: Katsusuke Nabeshima (eds). *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC '14).* ACM, New York, NY, USA. 2014; p. 296-303. (In Eng.) DOI: 10.1145/2608628.2608664

[6] Aidagulov R.R. B and Group Algebras and their Automorphisms. *Dnevnik Nauki*. 2019; 1(25):25. Available at: https://elibrary.ru/item.asp?id=36901640 (accessed 6.05.2019). (In Russ., abstract in Eng.)

[7] Aidagulov R.R. Group Algebras and Quantum Combinatorics. *Dnevnik Nauki*. 2019; 1(25):26. Available at: https://elibrary.ru/item.asp?id=36901641 (accessed 6.05.2019). (In Russ., abstract in Eng.)

[8] Harvey D., van der Hoeven J. Integer multiplication in time O(*n* log *n*). 2019; hal-02070778. Available at: https://hal.archives-ouvertes.fr/hal-02070778/file/nlogn.pdf (accessed 6.05.2019). (In Eng.)

[9] Harvey D. Faster truncated integer multiplication. *CoRR.* 2017; abs/1703.00640. Available at: http://arxiv.org/abs/1703.00640 (accessed 6.05.2019). (In Eng.)

[10] Harvey D., van der Hoeven J. Faster integer and polynomial multiplication using cyclotomic coefficient rings. *CoRR.* 2017; abs/1712.03693. Available at: http://arxiv.org/abs/1712.03693 (accessed 6.05.2019). (In Eng.)

[11] Harvey D., van der Hoeven J. On the complexity of integer matrix multiplication. *Journal of Symbolic Computation*. 2018; 89:1-8. (In Eng.) DOI: 10.1016/j.jsc.2017.11.001

[12] Harvey D., van der Hoeven J. Faster integer multiplication using plain vanilla FFT primes. *Mathematics of Computation.* 2019; 88:501-514. (In Eng.) DOI: 10.1090/mcom/3328

[13] Harvey D., van der Hoeven J. Faster integer multiplication using short lattice vectors. In: Scheidler R., Sorenson J. (eds). *ANTS XIII. Proceedings of the Thirteenth Algorithmic Number Theory Symposium*. Mathematical Sciences Publishers, Berkeley. 2019; 293-310. (In Eng.) DOI: 10.2140/obs.2019.2.293

[14] Harvey D., van der Hoeven J. Faster polynomial multiplication over finite fields using cyclotomic coefficient rings. *Journal of Complexity*. 2019; 54:101404. (In Eng.) DOI: 10.1016/j.jco.2019.03.004

[15] Harvey D., van der Hoeven J. Polynomial multiplication over finite fields in time O(*n* log *n*). 2019; hal-02070816. Available at: https://hal.archives-ouvertes.fr/hal-02070816 (accessed 6.05.2019). (In Eng.)

[16] Harvey D., van der Hoeven J., Lecerf G. Even faster integer multiplication. *Journal of Complexity*. 2016; 36:1-30. (In Eng.) DOI: 10.1016/j.jco.2016.03.001

[17] Harvey D., van der Hoeven J., Lecerf G. Faster Polynomial Multiplication over Finite Fields. *Journal of the ACM.* 2017; 63(6):52. 23 p. (In Eng.) DOI: 10.1145/3005344

[18] van der Hoeven J. Faster relaxed multiplication. In: Katsusuke Nabeshima (ed). *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC '14)*. ACM, New York, NY, USA. 2014; 405-412. (In Eng.) DOI: 10.1145/2608628.2608657

[19] Brent R.P., Zimmermann P. *Modern Computer Arithmetic*, Cambridge Monographs on Applied and Computational Mathematics. Vol. 18. Cambridge University Press, Cambridge, 2010. 236 p. (In Eng.)

[20] De A., Kurur P., Saha C., Saptharishi R. Fast Integer Multiplication Using Modular Arithmetic. *SIAM Journal on Computing.* 2013; 42(2):685-699. (In Eng.) DOI: 10.1137/100811167

Submitted 16.05.2019; revised 20.06.2019; published online 25.07.2019.

About the authors:

Rustem R. Aidagulov, Senior Researcher of Department of The-

oretical Informatics, Faculty of Mechanics and Mathematics, Lomonosov Moscow State University (1, Leninskie gory, Moscow 119991, Russia), Institute of Mechanical Engineering RAS after A.A. Blagonravov (4 Bardina St., Moscow 119334, Russia), Ph.D. (Phys.-Math.), ORCID: http://orcid.org/0000-0002-6579-429X, a_rust@bk.ru

Sergei T. Glavatsky, Associate Professor of Department of Theoretical Informatics, Faculty of Mechanics and Mathematics, Lomonosov Moscow State University (1, Leninskie gory, Moscow 119991, Russia), Ph.D. (Phys.-Math.), Associate Professor, ORCID: http://orcid.org/0000-0003-1857-6158, glavatsky_st@mail.ru

All authors have read and approved the final manuscript.