УДК 004.421

КОМПЬЮТЕРНАЯ МАТЕМАТИКА ДЛЯ ВЫЧИСЛИТЕЛЬНОЙ СЕТИ[1]

© Г. И. Малашонок

Ключевые слова: компьютерная математика, вычислительная сеть.
Предлагается проект системы компьютерной математики для вычислительной сети. Компонентами ядра такой системы являются основные математические объекты — числа, полиномы, функции, матрицы, последовательности, ряды. Обсуждается строение этих объектов, строится иерархия. Кроме основных классов, которые предполагают хранение данных в оперативной памяти, предлагается создание классов для данных во внешней памяти.

1 Введение

В истории развития вычислительных технологий выделяют два особо важных события — это появление Интернета и появление Суперкомпьютеров. Очередная технологическая революция в вычислительных технологиях происходит сегодня. Она связана с созданием Вычислительных сетей и Гридов, которые связывают многие удалённые суперкомпьютеры в единое вычислительное пространство, с появлением технологии, которые называют «облачные вычисления». Вычислительные сети позволяют не только накапливать информацию и обеспечивать к ней доступ, но и сохранять, поставлять и применять научные знания. Вычислительные сети позволяют получать новые научные знания в результате вычислительного эксперимента с виртуальной моделью реального объекта, задействуя при этом недоступные раннее вычислительные мощности.

Наличие высокоразвитых средств поддержки и применения математического знания определяет общий научный и технический потенциал общества. В той же мере, как обществу необходимы научные и образовательные институты, высокотехнологичные промышленные предприятия, в современном обществе должна функционировать система, которая не только сохраняет, но и позволяет применять математическое знание, опираясь на технические возможности современных вычислительных технологий — вычислительные сети. Электронная математика или e-математика — возможное название для такой системы.

Системы применения математического знания обычно называют системами компьютерной математики. Традиционно такие системы разделяли на численную математику и символьную математику, называемую также численно-аналитической математикой или компьютерной алгеброй. Системы, которые начинали развиваться как численные или символьные, сегодня уже во многом становятся смешанными. С появлением вычислительных сетей и гридов, естественно, ставится вопрос о создании системы компьютерной математики, которая обеспечивается мощностью вычислительной сети.

[1]Работа выполнена при поддержке программы «Развитие потенциала высшей школы» (проект 2.1.1/1853) и Темплана 1.12.09.

Прообразом такой будущей системы должна выступать система компьютерной алгебры как система, обеспечивающая точные и корректные вычисления. Включаемые в нее подсистемы вычислительной математики должны позволять проводить приближенные вычисления, в т. ч. и такие, в которых гарантируется некоторая допустимая погрешность решения конкретных задач, которая определяется пользователем. Время выполнения задания в такой системе определяется количеством процессоров, которые привлекаются к решению задачи. При увеличении мощности вычислительной сети это время может уменьшаться, а объёмы решаемых задач могут возрастать.

Настоящая работа является попыткой создания эскиза системы компьютерной математики для вычислительной сети.

2 Базис системы

Основные разделы математики, такие как теория множеств, алгебра, аналитическая и дифференциальная геометрия, математический анализ, теория функций, строятся на некотором запасе числовых множеств, построенных на них векторных пространствах и их отображениях (функциях и операторах). Эти же объекты должны составить основу системы компьютерной математики. Такая система должна опираться на поддержку основных числовых множеств современной математики, соответствующих векторных пространств и их отображений.

В понятие «поддержки» вкладывается смысл возможности оперирования с этими объектами привычным для математика образом. Это означает возможность определять такие множества, их подмножества, отдельные элементы и векторы, построенные на этих элементах, а также возможность определять операции на множествах и в векторных пространствах, т. е. задавать функции и операторы. Кроме того, необходимо иметь возможность именовать любые объекты и связывать их, подставляя одни объекты в другие, т. к. это общепринято в математике. Такое связывание должно сохраняться в пределах контекста решения одной задачи. Необходимо иметь возможность создавать композиции функций, вычислять значения функций в точках, выполнять замену переменных и тому подобные действия.

Выделим основные компоненты ядра такой системы.

— В основе системы должны лежать основные числовые множества, которые, вместе с операциями на них, образуют соответствующие алгебры. Эти числовые алгебры являются естественными базовыми *числовыми классами*, которые могут быть объединены в один *числовой пакет*. Информационные понятия и категории будем выделять курсивом, для отличия от математических.

— В силу особой роли, которую играют в математике рациональные функции: полиномы многих переменных и дробно-рациональные функции, эти объекты вместе с элементарными операциями на них образуют важные алгебры рациональных функций. Они могут образовать отдельный «*полиномиальный пакет*», куда будут входить *классы полиномов и рациональных функций* над числовыми множествами. Здесь должны решаться задачи нахождения полиномиальных НОД и НОК, нахождения результанта двух полиномов, вычисления базиса полиномиального идеала, выполнения операций над полиномиальными идеалами, задачи разложения полиномов на множители, нахождение корней полиномов и другие полиномиальные задачи.

— Трансцендентные функции, как элементарные, так и специальные, определенные на разных числовых множествах, образуют отдельный круг задач. Задачи вычисления пределов, производных, неопределённых и определённых интегралов для композиций трансцендентных функций, также как задачи упрощения, факторизации, разложения в суммы функций других видов, вычисления корней и другие образуют отдельный «*функциональный пакет*».

— Важным математическим инструментом являются линейные операторы на арифметических векторных пространствах и модулях. «*Матричный пакет*» могут образовать классы плотных и разреженных матриц, которые могут хранится либо в памяти, либо на диске. Векторные пространства и модули могут строиться как над числовыми, так и над функциональными алгебрами. Центральную роль играют задачи вычисления образа и прообраза вектора, полученного действием оператора, композиции и обращения операторов, вычисления ядра, эшелонной формы и спектра оператора, вычисления характеристического полинома. «*Матричный пакет*» играет центральную роль в технических и физических приложениях.

— Названные четыре пакета должны быть дополнены «*пакетом последовательностей и рядов*». Функциональные последовательности и ряды являются важной теоретической основой математического анализа. Пространства функциональных рядов активно применяются в теоретической физике. В последние годы они активно используются в связи с общим подходом к квантовой теории с точки зрения деформационного квантования.

3 Иерархия математических объектов

Основное математическое понятие «элемент» естественно поставить в вершине *иерархии всех объектов такой системы*. Это достаточно сильное требование, унифицирующее все *объекты* такой математической системы. При этом из системы сразу исключаются переменные, которые представляются *простыми типами данных*. Любой математический объект должен быть *наследником класса Element*, следовательно, сам должен быть не *простым типом, а объектом*.

Каждое число, каждый полином или трансцендентная функция, матрица или вектор является *наследником класса Element*. В свою очередь коэффициенты полиномов, аргументы трансцендентных функций и элементы матриц являются *некоторыми наследниками класса Element*. Это позволяет получать универсальный программный код, который не нужно дублировать для различных типов полей и колец. При этом вычислительные операции определяются *динамически* в зависимости от того, к какому *классу* относится конкретный математический объект или его коэффициент.

Element является главным *абстрактным классом*, и он не имеет *динамических полей*. Абстрактные *методы* в данном классе предназначены для задания основных операций с элементами алгебраических структур. В этом классе предусмотрены *методы*: add, subtract, multiply, divide, power, GCD, extendedGCD, mod, modInverse, modPower, divideAndRemainder, random, а также семейство *методов* типа valueOf, которые предназначены для отображения в ту алгебру, в которой лежит данный текущий представитель элемента, элементов других алгебр. Операций умножения может быть несколько. Например, умножение полиномов или матриц может выполняться разными алгоритмами. Поэтому операции умножения можно снабдить индексом, который будет указывать на

конкретный алгоритм.

Наследниками *класса* Element являются *классы*, которые образуют несколько подмножеств: подмножества чисел, полиномов, рациональных и трансцендентных функций, последовательностей и рядов. Кроме того, *наследниками являются классы матричного пакета* — это разреженные и плотные матрицы, а также векторы и тензоры.

4 Основные классы

4.1 Числовые классы

Можно выделить 12 основных числовых классов.

Это 6 точных числовых классов и 6 приближённых числовых классов. Набор точных числовых классов составляют классы Z, Q, Cz, Cq, Z_p и Z_p32. Здесь обозначено:

Z — целые числа;

Q — рациональные числа;

$Cz = \{a + ib : a, b \in Z, i^2 = -1\}$ — кольцо целых комплексных чисел;

$Cq = \{a + ib : a, b \in Q, i^2 = -1\}$ — поле рациональных комплексных чисел;

$Z_p = Z/pZ$ — простое поле характеристики p;

$Z_p32 = Z/pZ$ — простое поле, у которого характеристика не превосходит $2^{31} - 1$ — максимального положительного числа в четырехбайтном слове. Последний класс чисел часто используется в приложениях компьютерной алгебры, т. к. позволяет существенно ускорить вычисления по сравнению с Z_p.

Основные классы для приближённых чисел: R, C, $R64$, $C64$, $R128$ и $C128$. Здесь мы используем следующие обозначения:

$R64$ — это стандартные 64-разрядные числа с плавающей точкой для хранения приближённых действительных чисел;

R — это числа с плавающей точкой для хранения приближённых действительных чисел с произвольной мантиссой;

$R128$ — это числа с плавающей точкой для хранения приближённых действительных чисел со стандартной 52-разрядной мантиссой и отельным 32-разрядным полем для хранения порядка.

Три комплексных класса C, $C64$ и $C128$ образованы из классов R, $R64$ и $R128$ традиционным путем.

4.2 Полиномиальный класс

Класс полиномов многих переменных может быть один. В зависимости от того в каком из числовых классов находятся коэффициенты полиномов может быть 12 типов полиномов. С одной стороны, этот класс является наследником класса Element, с другой стороны, коэффициенты полиномов являются наследниками класса Element и могут принадлежать любому числовому классу.

4.3 Класс дробей

Класс дробей является наследником класса Element. У него есть два динамических поля типа Element для числителя и знаменателя дроби. Дочерним классом для дробей служит

класс рациональных чисел Q и класс комплексных рациональных чисел Cq. Числители и знаменатели в классе Q — это целые числа типа Z, а в классе Cq — целые комплексные числа. Динамические поля в этих классах наследуются из класса дробей.

4.4 Класс рациональных функций

Класс рациональных функций является дочерним по отношению к классу дробей. Числители и знаменатели рациональной функции — полиномы. Собственных динамических полей у него нет, а наследуются поля класса дробей.

4.5 Классы трансцендентных функций, последовательностей и рядов

Класс функций является самым широким объектом, аргументом функции может быть любой объект. Его наследником является класс композиций трансцендентных функций, у которых аргументами являются полиномы. Другими его наследниками являются классы функциональных последовательностей и рядов, соответствующие объекты получаются добавлением целочисленной переменной, отвечающей за номер члена последовательности.

4.6 Классы матриц, векторов и тензоров

Классы плотных и разреженных матриц – MatrixD и MatrixS, классы векторов и тензоров имеют элементы типа Element. Основные матричные методы это: adjoint, det, kernel, echelonForm, inverse, charPol. Центральным матричным методом является рекурсивный блочный метод adjDet. Этот метод возвращает определитель матрицы вместе с присоединённой и эшелонной матрицей. Ядро линейного оператора вычисляется исходя из эшелонной матрицы. Обратная матрица вычисляется на основе присоединённой матрицы и определителя. Отдельным методом является метод charPol, который обеспечивает вычисление характеристического полинома матрицы.

5 Структуры данных для внешней памяти

Кроме основных классов, которые предполагают хранение данных в оперативной памяти, необходимы для многих классов их аналоги, предусматривающие хранение данных во внешней памяти.

В первую очередь это классы файловых матриц, файловых полиномов и файловых функций. Эти классы позволят хранить большие объекты, которые не помещаются в оперативную память и ориентированы на параллельные вычисления. Для матричных и полиномиальных объектов хранение во внешней памяти представляется естественным. В то время как для композиций функций требуется наличие специального внутреннего регулярного представления, которое можно было бы продолжить до файлового объекта.

ЛИТЕРАТУРА

1. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14, вып. 4, 2009. С.744-748.

Malaschonok G.I. Computer mathematics for computational network. A project of the computer mathematics system for computational network is stated. The kernel components of this system are obtained by the basic mathematical objects (numbers, polynomials, functions, matrixes, sequences, series). These mathematical objects are kept in internal memory or out-of-core.

Key words: computer mathematics, computer network.

Поступила в редакцию 20 ноября 2009г.

УДК 004.421

О КОМПОЗИЦИИ ФУНКЦИЙ И МАШИННОМ ПРЕДСТАВЛЕНИИ [1]

© **Г. И. Малашонок, Р. А. Смирнов**

Ключевые слова: каноническая форма, формат хранения функций, композиция функций.

Обсуждается формат хранения композиции элементарных функций для системы компьютерной алгебры. Рассматривается задача вычисления производных и пределов композиций элементарных функций. Определяются некоторые специальные виды канонических форм композиции функций в заданном формате.

1 О представлении в компьютере композиции функций

В системах компьютерной алгебры выбираются определённые способы хранения функций. К способам хранения функций предъявляются разные требования, в зависимости от задач, которые должны решаться в такой системе.

Мы будем рассматривать универсальную систему компьютерной алгебры и будем выбирать такой формат хранения функции, который предполагает использование только оперативной памяти и проведения всего вычислительного процесса одним процессором.

В этом случае наиболее удобной структурой будет древовидная структура, которая повторяет строение функции. Она является наиболее гибкой и легко перестраиваемой, в отличие, например, от жёстких массивов. Она является естественной, т. к. повторяет реальную структуру композиции функции: связь функции с её аргументами повторяется как связь родительской вершины с её дочерними вершинами.

[1]Работа выполнена при поддержке программы "Развитие потенциала высшей школы" (проект 2.1.1/1853)

Отметим, что для параллельных вычислений такие структуры не подходят, т. к. процесс сбора всех фрагментов дерева для передачи другому процессору является очень трудоёмким. Поэтому для параллельных вычислений нужно применять специальный жёсткий формат для функции, в основе которого лежат не списки, а массивы. В универсальной системе необходимо иметь процедуры преобразования из одного формата в другой. Сложные однопроцессорные вычисления проводить в мягком древовидном формате, а для пересылок использовать жёсткий формат.

1.1 Рациональные функции

В силу особой роли, которую играют в математике рациональные и дробно-рациональные функции, для полиномов и полиномиальных дробей многих переменных выбирается отдельная компактная форма хранения. Она позволяет компактно хранить их и эффективно выполнять арифметические операции.

Класс Polynom содержит два динамических поля: массив степеней (powers) и массив коэффициентов(coeffs). Если в полиноме v переменных и n ненулевых коэффициентов, то в массиве коэффициентов хранится n коэффициентов, а в массиве степеней полинома хранятся vn целых чисел. Каждый отрезок из v чисел относится к одному моному — это степени при соответствующих переменных у этого монома. Мономы в полиноме всегда располагаются в обратном лексикографическом порядке, а переменные — в порядке, обратном старшинству. Каждый полином берется из определённого полиномиального кольца, которое определяет множество коэффициентов полинома. Например полином из кольца $\mathbb{Z}[x, y, z]$

$$p = -7x^2y^5z^5 + 9x^7y^3z^3 + 3x^5y^3z^3 - 8x^4y^3z + 5x^3y^2z$$

хранится в виде двух массивов:

powers = 2,5,5, 7,3,3, 5,3,3, 4,3,1, 3,2,1,

coeffs = -7, 9, 3, -8, 5.

Полиномы играют роль листовых вершин в дереве функции.

1.2 Общие композиции функции

Для создания дерева композиции функций определим *класс* TF(Tree Function), в котором заданы три *динамических поля*:

(1) целочисленное поле $name$ для имени функции,

(2) массив аргументов типа TF $args$,

(3) поле типа Element X для указания на листовую вершину дерева.

Каждая функция определяется именем, например, sin, cos, $plus$, $minus$ и т. д. Всё множество имен упорядочено некоторым образом. В поле $name$ хранится соответствующий номер функции.

По характеру аргументов все функции делятся на два типа — простые и составные.

У простой функции только один аргумент, на который указывает поле X, а массив аргументов $args$ пустой. У составных функций может быть один или несколько аргументов, при этом поле X пусто, а на аргументы указывают элементы массива $args$.

На рис. 1. представлена схема дерева для функции $\sin(\cos(x) + \mathrm{tg}(y))$.

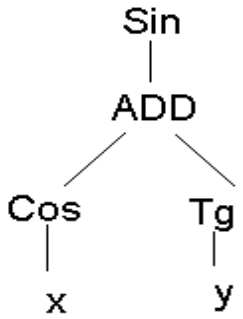

Рис. 1. Схема дерева для функции $\sin(\cos(x) + \mathrm{tg}(y))$

В этом примере вершины cos и tg простого типа, а вершины ADD и sin – составного.

2 Числа и предельные числа

Предопределено 12 разных числовых классов. Для каждого числового класса, в котором требуется вычислять пределы функций в точках, определены классы предельных чисел, в которых кроме предельного значения сохраняется еще и способ приближения к предельному числу.

Например, числовой класс NumberR64 предназначен для хранения приближённых действительных чисел в 64-разрядном слове. Его наследует класс предельных чисел NumberR64Lim, в котором кроме родительского динамического поля определено еще одно динамическое поле rORl — целочисленная переменная, принимающее значение -1, 1 или 0, в зависимости от того, с какой стороны требуется найти предел в точке: слева, справа или двусторонний.

3 Каноническая форма

Существует большое множество представлений одной и той же композиции элементарных функций в разных, но в то же время эквивалентных видах. Каждый фрагмент функции, образованный цепочкой произведений, будем называть мультипликативным фрагментом. Мы выделили следующие формы, к которым будем приводить композиции функций.

1. Форма, которая соответствует входному выражению, заданному пользователем (начальная форма).

2. Форма, которая образуется из начальной в результате четырех действий: замена вычитания на сложение с данным выражением, умноженным на минус один; замена деления на умножение на данное выражение в степени минус один; удаление ассоциативных скобок внутри операции сложения; удаление ассоциативных скобок внутри операции умножения (базовая форма).

3. Форма, полученная из базовой формы путём приведения подобных слагаемых и приведения произведения сомножителей к стандартному виду путём перестановки полиномов-сомножителей в мультипликативных фрагментах и группировки одинаковых сомножителей под одну степень (приведенная базовая форма).

4. Форма, полученная из формы 3 после удаление сомножителей в степени ноль (сокращенная базовая форма).

5. Форма, полученная из формы 4 путём перемножения всех полиномов, стоящих в каждом мультипликативном фрагменте, после чего в каждом фрагменте останется только один полином-сомножитель (форма с перемноженными полиномами).

6. Форма, полученная из формы 5 путём формального сложения всех дробей без сокращения (форма с одной дробью).

7. Форма, полученная из формы 5 путём раскрытия всех скобок в мультипликативных фрагментах целых слагаемых и приведения целой части к форме 5 (форма с без скобок в целой части).

8. Форма, полученная из форм 6 и 7 факторизацией числителя дроби и свободного слагаемого (форма с одной дробью и факторизованным знаменателем).

9. Форма, в которой открыты все скобки и факторизованы знаменатели (форма полного разложения).

10. Факторизованная форма без учета дробей. Для получения этой формы производится группировка слагаемых и выражение раскладывается на множители.

11. Полная факторизованная форма. Получается вынесением общего множителя у всех числителей дробей и у целой части выражения.

12. Выходная форма. Получается эта форма из любой формы путём замены умножения на минус единицу вычитанием и отрицательных степеней — делением.

4 Вычисление производной и предела композиции функций

Производная композиции функций вычисляется по известной из анализа формуле:

$$(F(g(x)))' = F'(g(x))g'(x).$$

Если $F(g)$ — композиция функций многих переменных, то приведённая формула может рассматриваться как формула вычисления частной производной.

Вычисление пределов опирается на применение правила Лопиталя: если предел отношения производных двух функций существует, то предел отношения этих функций будет ему равен. Большинство случаев раскрытия неопределённостей можно свести к отношению функций. Например, в случае разности двух бесконечно больших величин можно вынести за скобки одно из слагаемых. В случае произведения бесконечно малой на бесконечно большую можно перейти к отношению одной из них на величину, которая обратна к другой из них. И так далее.

ЛИТЕРАТУРА

1. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14, вып. 4, 2009. С.744-748.

Malaschonok G.I., Smirnov R.A. About composition of functions and a machine forms. The form of the function compositions for computer algebra system are discussed. The problem of calculation of limits and derivations of a composition of functions is considered.

Key words: canonical form, function compositions, machine forms of functions.

Поступила в редакцию 20 ноября 2009г.

УДК 004.421

ОБ АЛГОРИТМЕ ФАКТОРИЗАЦИИ ПОЛИНОМОВ МНОГИХ ПЕРЕМЕННЫХ[1]

© **Г. И. Малашонок, Д. С. Ивашов**

Ключевые слова: факторизация полиномов, полиномы многих переменных, компьютерная алгебра.

Приводится алгоритм факторизации полиномов многих переменных. Алгоритм опирается на алгоритм факторизации полиномов одной переменной и на методы гомоморфных образов в кольцах полиномов.

Одной из очень важных задач компьютерной алгебры является задача факторизации полиномов многих переменных над рациональными числами.

Для факторизации полиномов одной переменной известно много замечательных алгоритмов. С обзором работ по факторизации можно познакомиться по учебнику Е.В. Панкратьева [1]. Будем полагать, что имеется некоторый алгоритм разложения на множители полиномов одной переменной. Требуется построить алгоритм факторизации для полиномов многих переменных. Рассмотрим сначала случай двух переменных.

Пусть $F(x,y)$ раскладывается на взаимнопростые множители $f_i(x,y)$: $F(x,y) = \prod_{i=0}^{s} f_i(x,y)^{n_i}$, $n_0 < n_1 < ... < n_s$. Тогда это разложение можно получить вычисляя НОД $F(x,y)$ и $F'(x,y)$ необходимое количество раз.

Пусть $f(x,y)$ — полином от двух переменных, у которого нет кратных сомножителей и который раскладывается на взаимнопростые множители: $f(x,y) = \prod_{j}^{s} f_j(x,y)$. Сомножители $f_j(x,y)$ требуется найти. Обозначим $S_x = deg_x f(x,y)$. Возьмём множество разных точек $Y \subseteq \mathbb{Q}$, такое что $|Y| = S_x + 1 = k$. Обозначим $\varphi_i(y) = f(x_i,y)\ \forall x_i \in Y$.

[1]Работа выполнена при поддержке программы «Развитие потенциала высшей школы» (проект 2.1.1/1853).

Разложим $\varphi_i(y)$ на множители: $\varphi_i(y) = \prod\limits_s \varphi_{si}(y)$. Воспользуемся китайской теоремой об остатках для полиномов. Тогда полином $f_j(x,y)$ можно восстановить по его k образам: $\varphi_{j1}(y), ..., \varphi_{jk}(y)$, поднимая каждый коэффициент при степени y в полином степени $k-1$ от x по его k числовым значениям. Таким образом мы восстановим все множители $f_j(x,y)$ полинома $f(x,y)$.

Рассмотрим $f(x_1, ..., x_n)$ — полином от n переменных, у которого нет кратных сомножителей и который раскладывается на взаимнопростые множители: $f(x_1, ..., x_n) = \prod\limits_j f_j(x_1, ..., x_n)$. Сомножители $f_j(x_1, ..., x_n)$ требуется найти. Пусть Y_i $(i = 1, ..., n-1)$ конечные подмножества $\mathbb{Q}$, такие что $|Y_i| = k_i + 1$, где $k_i = deg_{x_i} f(x_1, ..., x_n)$. В каждом из этих множеств все элементы различные. Введём обозначение элементов Y_i: $Y_i = \{t_i^0, ..., t_i^{k_i}\}$. Обозначим $\varphi_{v_1,...,v_{n-1}}(x_n) = f(t_1^{v_1}, ..., t_{n-1}^{v_{n-1}}, x_n)$ полином, который получается подстановкой вместо переменных x_i чисел $t_i^{v_i}$ $(i = 1, ..., n-1;\ \ v_i \in [0, ..., k_i])$. Разложим каждый из $\varphi_{v_1,...,v_{n-1}}(x_n)$ на множители: $\varphi_{v_1,...,v_{n-1}}(x_n) = \prod\limits_j \varphi_{j,v_1,...,v_{n-1}}(x_n)$. Обозначим $\rho_{v_1,...,v_{n-1}} = \prod\limits_{i=1}^{n-1}(x_i - t_i^{v_i})$. Отметим, что $\varphi_{j,v_1,...,v_{n-1}}(x_n)$ является образом $f_j(x_1, ..., x_n)$ при отображении $\mathbb{Q}[x_1, ..., x_n] \to \mathbb{Q}[x_1, ..., x_n]/\rho_{v_1,...,v_{n-1}}\mathbb{Q}[x_1, ..., x_n]$. Перейдём к восстановлению искомых функции $f_j(x_1, ..., x_n)$.

Шаг 1.

Отметим, что $\varphi_{j,v_1,...,v_{n-1}}(t_n^{v_n}) = f_j(t_1^{v_1}, ..., t_n^{v_n})\ \ \forall v_1 \in [0, ..., k_1], ..., v_n \in [0, ..., k_n]$. Введём обозначения для коэффициентов полиномов $\varphi_{j,v_1,...,v_{n-2},g}(x_n)$. Пусть

$$\varphi_{j,v_1,...,v_{n-2},g}(x_n) = \sum_{g_n=0}^{k_n} \alpha^0_{j,v_1,...,v_{n-2},g,g_n} x_n^{g_n}, \qquad g = 0, ..., k_{n-1}.$$

Восстановим полином степени k_{n-1} по его значениям: $\alpha^0_{j,v_1,...,v_{n-2},0,g_n}, ...,\ \ \alpha^0_{j,v_1,...,v_{n-2},k_{n-1},g_n}$ в точках $t_{n-1}^0, ..., t_{n-1}^{k_{n-1}}$ по формуле Лагранжа:

$$f_{j,v_1,...,v_{n-2},g_n} = \sum_{g=0}^{k_{n-1}} \alpha^0_{j,v_1,...,v_{n-2},g,g_n} \frac{(x_{n-1} - t_{n-1}^0)...(x_{n-1} - t_{n-1}^{g-1})(x_{n-1} - t_{n-1}^{g+1})...(x_{n-1} - t_{n-1}^{k_{n-1}})}{(t_{n-1}^g - t_{n-1}^0)...(t_{n-1}^g - t_{n-1}^{g-1})(t_{n-1}^g - t_{n-1}^{g+1})...(t_{n-1}^g - t_{n-1}^{k_{n-1}})}.$$

Сгруппируем коэффициенты при неизвестных:

$$f_{j,v_1,...,v_{n-2},g_n} = \sum_{s=0}^{k_{n-1}} \alpha^1_{j,v_1,...,v_{n-2},g_n,s} x_{n-1}^s.$$

Составим полином от двух переменных:

$$\varphi_{j,v_1,...,v_{n-2}}(x_{n-1}, x_n) = \sum_{g_{n-1}=0}^{k_{n-1}} \sum_{g_n=0}^{k_n} \alpha^1_{j,v_1,...,v_{n-2},g_{n-1},g_n} x_{n-1}^{g_{n-1}} x_n^{g_n},$$

который в точках $t_{n-1}^{v_{n-1}}, t_n^{v_n}$ равен значению полинома $f_j(t_1^{v_1}, ..., t_{n-2}^{v_{n-2}}, x_{n-1}, x_n)$:

$$\varphi_{j,v_1,...,v_{n-2}}(t_{n-1}^{v_{n-1}}, t_n^{v_n}) = f_j(t_1^{v_1}, ..., t_n^{v_n}).$$

Шаг r.

Пусть уже получено $(k_1+1)...(k_{r-1}+1)$ полиномов:

$$\varphi_{j,v_1,...,v_{r-1},g}(x_{r+1},...,x_n)=\sum_{g_{r+1}=0}^{k_{r+1}}...\sum_{g_n=0}^{k_n}\alpha^{n-1-r}_{j,v_1,...,v_{r-1},g,g_{r+1},...,g_n}x_{r+1}^{g_{r+1}}...x_n^{g_n},$$

$g=0,...,k_r; v_1\in[0,...,k_1],...,v_{r-1}\in[0,...,k_{r-1}]$.

Будем восстанавливать полином степени k_r по его значениям: $\alpha^{n-1-r}_{j,v_1,...,v_{r-1},0,g_{r+1},...,g_n}$, ..., $\alpha^{n-1-r}_{j,v_1,...,v_{r-1},k_r,g_{r+1},...,g_n}$ в точках $t_r^0,...,t_r^{k_r}$ по формуле Лагранжа:

$$f_{j,v_1,...,v_{r-1},g_{r+1},...,g_n}=\sum_{g=0}^{k_r}\alpha^{n-1-r}_{j,v_1,...,v_{r-1},g,g_{r+1},...,g_n}\frac{(x_r-t_r^0)...(x_r-t_r^{g-1})(x_r-t_r^{g+1})...(x_r-t_r^{k_r})}{(t_r^g-t_r^0)...(t_r^g-t_r^{g-1})(t_r^g-t_r^{g+1})...(t_r^g-t_r^{k_r})}.$$

Сгруппируем коэффициенты при неизвестных:

$$f_{j,v_1,...,v_{r-1},g_{r+1},...,g_n}=\sum_{s=0}^{k_r}\alpha^{n-r}_{j,v_1,...,v_{r-1},g_{r+1},...,g_n,s}x_r^s.$$

Составим полином от $n-r+1$ переменной:

$$\varphi_{j,v_1,...,v_{r-1}}(x_r,...,x_n)=\sum_{g_r=0}^{k_r}\sum_{g_{r+1}=0}^{k_{r+1}}...\sum_{g_n=0}^{k_n}\alpha^{n-r}_{j,v_1,...,v_{r-1},g_r,g_{r+1},...,g_n}x_r^{g_r}x_{r+1}^{g_{r+1}}...x_n^{g_n},$$

который в точках $t_r^{v_r},...,t_n^{v_n}$ равен значению полинома $f_j(t_1^{v_1},...,t_{r-1}^{v_{r-1}},x_r,...,x_n)$:

$$\varphi_{j,v_1,...,v_{r-1}}(t_r^{v_r},...,t_n^{v_n})=f_j(t_1^{v_1},...,t_n^{v_n}).$$

Шаг $n-1$.

Пусть уже получено k_1+1 полиномов:

$$\varphi_{j,g}(x_2,...,x_n)=\sum_{g_2=0}^{k_2}...\sum_{g_n=0}^{k_n}\alpha^{n-2}_{j,g,g_2,...,g_n}x_2^{g_2}...x_n^{g_n},\qquad g=0,...,k_1.$$

Восстановим полином степени k_1 по его значениям: $\alpha^{n-2}_{j,0,g_2,...,g_n}$, ..., $\alpha^{n-2}_{j,k_1,g_2,...,g_n}$ в точках $t_1^0,...,t_1^{k_1}$ по формуле Лагранжа:

$$f_{j,g_2,...,g_n}=\sum_{g=0}^{k_1}\alpha^{n-2}_{j,g,g_2,...,g_n}\frac{(x_1-t_1^0)...(x_1-t_1^{g-1})(x_1-t_1^{g+1})...(x_1-t_1^{k_1})}{(t_1^g-t_1^0)...(t_1^g-t_1^{g-1})(t_1^g-t_1^{g+1})...(t_1^g-t_1^{k_1})}.$$

Сгруппируем коэффициенты при неизвестных:

$$f_{j,g_2,...,g_n}=\sum_{s=0}^{k_1}\alpha^{n-1}_{j,g_2,...,g_n,s}x_1^s.$$

Составим полином от n переменных:

$$\varphi_j(x_1,...,x_n)=\sum_{g_1=0}^{k_1}\sum_{g_2=0}^{k_2}...\sum_{g_n=0}^{k_n}\alpha^{n-1}_{j,g_1,g_2,...,g_n,}x_1^{g_1}...x_n^{g_n},$$

который в точках $t_1^{v_1}, ..., t_n^{v_n}$ равен значению полинома $f_j(t_1^{v_1}, ..., t_n^{v_n})$:

$$\varphi_j(t_1^{v_1}, ..., t_n^{v_n}) = f_j(t_1^{v_1}, ..., t_n^{v_n}).$$

Получен искомый полином $f_j(x_1, ..., x_n)$. Вычисляя последовательно все полиномы $f_j(x_1, ..., x_n)$, $i = 0, 1, .., s$, получим искомое разложение: $f(x_1, ..., x_n) = \prod_j f_j(x_1, ..., x_n)$.

ЛИТЕРАТУРА

1. *Панкратьев Е.Г.* Элементы компьютерной алгебры. Учебное пособие. М. Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2007.

Malaschonok G.I., Ivashov D.S. An algorithm of factorization of polynomials of several variables.

Key words: factorization of polynomials, computer algebra.

Поступила в редакцию 20 ноября 2009г.

УДК 004.421

О ПАРАЛЛЕЛЬНОМ ВЫЧИСЛЕНИИ ДИСКРЕТНОГО ПРЕОБРАЗОВАНИЯ ФУРЬЕ И ПРОВЕДЁННЫХ ЭКСПЕРИМЕНТАХ[1]

© **А. О. Лапаев**

Ключевые слова: параллельная компьютерная алгебра, дискретное преобразование Фурье, быстрое преобразование Фурье, полиномиальные алгоритмы.

Предлагается алгоритм параллельного вычисления многомерного дискретного преобразования Фурье полинома нескольких переменных в простом поле. Приводятся результаты экспериментов на кластере МСЦ РАН.

1 Введение

Пусть $f \in \mathbb{Z}_p[x_1, x_2, \ldots, x_d]$, p — простое число. Пусть наибольшая степень переменной x_i в полиноме f равна $n_i - 1$, $n_i = 2^{N_i}$. Обозначим $n = n_1 n_2 \ldots n_d$. Тогда полином f можно записать в виде:

$$f = \sum_{i_1=0}^{n_1-1} \sum_{i_2=0}^{n_2-1} \ldots \sum_{i_d=0}^{n_d-1} f_{i_1 i_2 \ldots i_n} x_1^{i_1} x_2^{i_2} \ldots x_d^{i_d}.$$

[1]Работа выполнена при поддержке программы «Развитие потенциала высшей школы» (проект 2.1.1/1853).

Пусть простое число p такое, что каждое из чисел n_i делит $p-1$. Тогда в $\mathbb{Z}_p$ существует корень из 1 степени n_i, который будем обозначать ω_i. Введём определение дискретного преобразования Фурье полинома f.

О п р е д е л е н и е. Дискретным преобразованием Фурье (ДПФ) для полинома f называется d-мерная таблица чисел $\mathcal{F}(f) = (\hat{f}_{j_1 \dots j_d})$, где $1 \leqslant j_1 \leqslant n_1,\ 1 \leqslant j_2 \leqslant n_2,\ \dots, 1 \leqslant j_d \leqslant n_d$, где

$$\hat{f}_{j_1 j_2 \dots j_d} = \sum_{i_1=0}^{n_1-1} \sum_{i_2=0}^{n_2-1} \dots \sum_{i_d=0}^{n_d-1} f_{i_1 i_2 \dots i_n} \omega_1^{j_1 i_1} \omega_2^{j_2 i_2} \dots \omega_d^{j_d i_d}. \tag{1}$$

Правая часть (1) содержит n слагаемых. В левой части находится элемент d-мерной таблицы, всего таких элементов n. Следовательно, общее число операций для вычисления ДПФ полинома f будет $O(n^2)$. Рассмотрим способ быстрого вычисления дискретного преобразования Фурье.

Запишем формулу (1) в виде:

$$\hat{f}_{j_1 j_2 \dots j_d} = \sum_{i_d=0}^{n_d-1} \omega_d^{j_d i_d} \sum_{i_{d-1}=0}^{n_{d-1}-1} \omega_{d-1}^{j_{d-1} i_{d-1}} \dots \sum_{i_k=0}^{n_k-1} \omega_k^{j_k i_k} \dots \sum_{i_1=0}^{n_1-1} f_{i_1 i_2 \dots i_d} \omega_1^{j_1 i_1}.$$

При фиксированных параметрах $i_2, \dots, i_d$ выражение

$$F^1_{j_1, i_2, i_3, \dots, i_d} = \sum_{i_1=0}^{n_1-1} f_{i_1 i_2 \dots i_d} \omega_d^{j_1 i_1}$$

— j_1-й элемент одномерного дискретного преобразования Фурье на n_1 точках, которое может быть посчитано по алгоритму Cooley-Tukey [1] за $O(n_1 \log_2 n_1)$ операций.

Обозначим

$$F^{k+1}_{j_1, \dots, j_{k+1}, i_{k+2}, \dots, i_d} = \sum_{i_{k+1}=0}^{n_{k+1}-1} F^k_{j_1, \dots, j_k, i_{k+1} \dots i_d} \omega_d^{j_{k+1} i_{k+1}}.$$

При последовательном вычислении $F_1, F_2, \ \dots, F_d$ элемент F_d будет содержать ДПФ полинома f.

Схема распараллеливания будет состоять из d последовательных шагов, каждый из которых выполняется параллельно. Схему вычисления каждого шага можно представить в виде бинарного дерева, у которого данные распределяются от корневой вершины к листовым, а результат вычислений собирается снова в корневой вершине. На каждом шаге параллельно вычисляются одномерные ДПФ, при этом все вычисления происходят в листовых вершинах. При переходе к следующему шагу меняется порядок переменных, а после шага d будет получен искомый d-мерный вектор дискретного преобразования Фурье. Приведём алгоритм.

2 Алгоритм

Будем проводить вычисления в следующем порядке: на шаге k параллельно вычисляются n/n_k дискретных преобразований Фурье при фиксированных значениях индексов $j_1 \dots j_{d-k-1} j_{d-k+1} \dots j_d$. Пусть F_k — результат вычислений k-го шага, и пусть $F_0 = f$.

Приведем схему вычислений на шаге k. В корневой вершине d-мерный массив, полученный на предыдущем шаге, делится на две равные части по индексу. И каждая часть отправляется в соответствующую дочернюю вершину. В дочерних вершинах повторяется разделение это массива по тому же индексу на две половины. Такой процесс выполняется рекурсивно до тех пор, пока имеются свободные процессоры либо пока возможно деление по данному индексу, т. е. число процессоров, вовлеченных в вычисления, меньше n/n_k. На листовом уровне вычисляются одномерные ДПФ и результат возвращается в обратном порядке. После чего, при переходе к следующему шагу вычислений, порядок индексов массива меняется $[j_{d-k+2}j_1 \dots j_{d-k+1}]$ на $[j_{d-k+1}j_1 \dots j_{d-k}]$. Отметим, что на шаге k требуется не больше, чем n/n_k процессоров.

Пусть имеется m компьютерных модулей (КМ) с номерами $1, \dots, m$. Пусть $k = 1$.

1)КМ с номером 1 получает d-мерный массив и список свободных КМ. Вычисляются степени $w_{d-k+1}^{i_{d-k+1}j_{d-k+1}}$. Массив коэффициентов и список свободных процессоров делятся пополам. Одна половина массива коэффициентов и списка свободных процессоров вместе с массивом степеней отсылается КМ с номером $(n/2+1)$, а вторые половины остаются в данном КМ.

2)Каждый КМ, получивший свою часть задачи, продолжает такое деление дальше. Процесс продолжается до достижения листового уровня либо исчерпания всех свободных процессоров.

3)Выполняется параллельное вычисление одномерных преобразований Фурье.

4)Каждый КМ пересылает результат обратно по дереву тому процессору, от которого получил данные.

5)КМ 1 собирает результат и меняет порядок индексации результирующего массива с $[j_{d-k+2}j_1 \dots j_{d-k+1}]$ на $[j_{d-k+1}j_1 \dots j_{d-k}]$.

6)Увеличивается номер шага k. Если $k = d+1$, то конец всех вычислений, иначе переходим на п.1.

3 Результаты экспериментов

По приведённому выше алгоритму был разработан программный комплекс. Эксперименты проводились на кластере МСЦ РАН.

При переходе от вычислений на s_0 процессорах к вычислениям на s, $s > s_0$, процессорах ускорение достигает максимально возможных 100%, когда $\frac{T_{s_0}}{T_s} = \frac{s}{s_0}$. Ускорение равно нулю, когда $T_{s_0} = T_s$. Чтобы определить ускорение вычислений при других значениях $\frac{T_{s_0}}{T_s}$, определим ускорение, как линейную функцию величины $\frac{T_{s_0}}{T_s}$.

О п р е д е л е н и е. Ускорением вычислений при переходе от s_0-процессорного кластера, где время вычислений равно T_{s_0}, к s-процессорному кластеру, где время вычислений равно T_s, $s > s_0$, называется функция

$$\alpha(T_s) = \frac{\frac{T_{s_0}}{T_s} - 1}{\frac{s}{s_0} - 1} \cdot 100\%.$$

Таблица 1

Время выполнения параллельного алгоритма ДПФ для полиномов двух переменных

Количество переменных = 2					
Время, мс					
Степень	511	1023	2047	4095	8191
Процессоры					
1	817	6838	47466	371882	2919529
2	608	3918	31059	190410	2267073
4	424	1755	11470	79170	715310
8	374	1109	4220	30643	210488
16	323	730	2785	11514	83500
32	294	649	1792	6372	36665
64	367	576	1454	4732	18416
128	454	571	1341	3749	11462
256	516	1083	1423	4571	12469

Таблица 2

Ускорение параллельного алгоритма ДПФ для полиномов двух переменных при переходе от одного процессора к нескольким

Ускорение, %					
Степень	511	1023	2047	4095	8191
Процессоры					
1	–	–	–	–	–
2	34,38	74,53	52,83	95,31	28,78
4	30,9	96,54	104,61	123,24	102,71
8	16,92	73,8	146,4	159,09	183,86
16	10,2	55,78	100,96	208,65	226,43
32	5,74	30,76	82,22	185,04	253,64
64	1,95	17,26	50,23	123,16	250,05
128	0,63	8,64	27,08	77,32	199,77
256	0,23	2,08	12,69	31,51	91,43

Таблица 3

Время выполнения параллельного алгоритма ДПФ для полиномов трех переменных

Количество переменных = 3				
Время, мс				
Степень	31	63	127	255
Процессоры				
1	345	6916	212756	8140475
2	263	3446	111121	2955568
4	175	1731	45085	1055536
8	129	888	12420	358084
16	86	490	4726	127972
32	68	290	2466	66859
64	58	316	1495	17918
128	433	184	945	7554

Таблица 4

Ускорение параллельного алгоритма ДПФ для полиномов трех переменных при переходе от одного процессора к нескольким

Ускорение, %				
Степень	31	63	127	255
Процессоры				
1	–	–	–	–
2	31, 18	100, 7	91, 46	175, 43
4	32, 18	99, 85	123, 97	123, 74
8	32, 92	96, 98	230, 43	310, 48
16	20, 08	87, 43	293, 45	417, 41
32	13, 14	73, 7	175, 08	289, 54
64	7, 85	33, 15	224, 3	719, 55
128	−0, 16	28, 817	176, 49	847, 75

По результатам вычислительных экспериментов можно сделать вывод о высокой эффективности параллельного алгоритма БПФ для больших вычислительных задач. Лучшее ускорение составляет 847 %. Это можно объяснить тем, что при увеличении количества процессоров части задачи на каждом процессоре становятся достаточно малы для того, чтобы поместиться в кэш.

ЛИТЕРАТУРА

1. *Ноден П., Китте К.* Алгебраическая алгоритмика (с упражнениями и решениями). // Пер. с франц. М.: Мир, 1999.

2. *Кнут Д.Э.* Исскуство программирования, т.2. Получисленные алгоритмы, 3-е изд. М.: Издательский дом «Вильямс», 2001.

3. *Кормен, Томас Х., Лейзерсон, Чарльз И., Ривест, Рональд Л., Штойн, Клиффорд.* Алгоритмы: построение и анализ, 2-е издание: Пер. с англ. М.: Издательский дом «Вильямс», 2005.

4. *Лапаев А.О.* О вычислении многомерного дискретного преобразования Фурье в простом поле. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14, вып. 4, 2009. С.729-731.

5. *Лапаев А.О.* Параллельное вычисление дискретного преобразования Фурье полинома в простом поле. Материалы 9-й международной конференции-семинара «Высокопроизводительные параллельные вычисления на кластерных системах». Владимир — 2009. С. 272-273.

6. *Малашонок Г.И., Лапаев А.О.* Статистическая схема распараллеливания вычисления определителя, присоединённой матрицы и решения систем линейных уравнений в кольце целых чисел. XI Державинские чтения ИМФИ им. Г.Р. Державина. Тамбов, 3 февраля 2006. С. 53-56.

Lapaev A.O. A parallel computation of multidimensional discrete Fourier transform and experiments' results. Algorithm of parallel computation of multidimensional discrete Fourier transform of polynomial of many vars if described. Experiments' results on MVS cluster are stated.

Key words: parallel computer algebra, discrete Fourier transform, fast Fourier transform, polynomial algorithms.

Поступила в редакцию 20 ноября 2009г.

УДК 004.421

РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ С КУСОЧНО-НЕПРЕРЫВНЫМИ ПРАВЫМИ ЧАСТЯМИ С ПОМОЩЬЮ ПРЕОБРАЗОВАНИЯ ЛАПЛАСА [1]

© **М. А. Рыбаков**

Ключевые слова: алгоритм решения систем дифференциальных уравнений, система дифференциальных уравнений, преобразование Лапласа.
В работе рассматривается алгоритм решения систем линейных дифференциальных уравнений с кусочно-непрерывными правыми частями с помощью преобразования Лапласа, приводятся примеры решения таких систем.

1 Введение

Одной из актуальных задач компьютерной алгебры является задача решения систем линейных дифференциальных уравнений с постоянными коэффициентами. В работе [1] был рассмотрен случай, когда в правой части системы стоят непрерывные функции. С практической точки зрения наибольший интерес представляет система уравнений с кусочно-непрерывными правыми частями. Эта задача решается в системе компьютерной алгебры ParCA.

2 Алгоритм решения систем линейных дифференциальных уравнений с кусочно-непрерывными правыми частями

Приведём схему алгоритма.

Прямое преобразование Лапласа.

1. Преобразование левой части системы дифференциальных уравнений. В результате прямого преобразования Лапласа левая часть системы дифференциальных уравнений преобразуется в матрицу полиномов $A(p)$ одной действительной переменной p.

2. Преобразование правой части системы дифференциальных уравнений. Каждая функция в правой части разбивается на отдельные слагаемые и для каждого слагаемого применяется табличная функция для вычисления прямого преобразования Лапласа. Результатом будут целые и дробно-рациональные выражения с действительной переменной p. Причем в полученные выражения могут входить Гамма-функции.

3. Формирование объектов (K) для хранения дробно-рациональных выражений с Гамма-функциями.

[1]Работа выполнена при поддержке программы «Развитие потенциала высшей школы» (проект 2.1.1/1853).

4. Добавление в правую часть слагаемых, соответствующих начальным условиям для системы дифференциальных уравнений. Эти слагаемые являются полиномами переменной p.

5. Вычисление для матрицы $A(p)$ присоединённой матрицы $A(p)^*$ и определителя $det(A(p))$.

6. Вычисление комплексных корней полинома $det(A(p))$ с заданной точностью и разложение дроби $1/det(A(p))$ в сумму простых дробей в комплексной области.

7. Формирование столбца V, каждый элемент которого состоит из сумм преобразованных правых частей системы и преобразованных начальных условий.

8.Умножение присоединённой матрицы $A(p)^*$ на столбец V. Результатом будет столбец W, элементы которого состоят из сумм рациональных дробей.

9. Разложение дробей в столбце W в суммы простых дробей в комплексной области.

10.Умножение столбца W на выражение $1/det(A(p)$, которое записано в виде суммы простых дробей и приведение подобных членов. В результате каждый элемент вектора W будет суммой простых дробей в комплексной области.

Обратное преобразование Лапласа.

11. Нахождение прообразов для простых дробей из вектора W при преобразовании Лапласа, используя табличные функции. И восстановление прообразов по массиву объектов K.

3 Пример

Решить систему уравнений:

$$\begin{cases} x'''(t) - x'(t) - 2x(t) - y'''(t) + y(t) = (t^2e^{2t} - e^t)UnitStep(t-1) + e^tUnitStep(t), \\ 3x'''(t) + x''(t) - 2x'(t) + y'''(t) + y(t) = (e^{2t} - te^t)UnitStep(t-1) + te^tUnitStep(t). \end{cases}$$

Начальные условия: $x(0) = 5; x'(0) = 10; x''(0) = 30;\ y(0) = 4; y'(0) = 14; y''(0) = 20$.

Описание задачи на входном языке:

$systLDE($
$D(x,t,3)-D(x,t)-2x-D(y,t,3)+y = t^2e^{2t}UnitStep(t-1)-e^tUnitStep(t-1)+e^tUnitStep(t),$
$3D(x,t,3) + D(x,t,2) - 2D(x,t) + D(y,t,3) + y = e^{2t}UnitStep(t-1) - te^tUnitStep(t-1) + te^tUnitStep(t))),$
$InitCond(D(x,t,0,0) = 5, D(x,t,0,1) = 10, D(x,t,0,2) = 30,$
$D(y,t,0,0) = 4, D(y,t,0,1) = 14, D(y,t,0,2) = 20).$

Решение системы дифференциальных уравнений:

$x(t) = t^2(10.031)e^{-t} - (1.25)e^t + (5.539)e^{1.228t} + 2e^{0.356t}(-3.736\ cos(0.513t) +$
$15.530\ sin(0.513t)) + 2e^{-0.595}(-0.924\ cos(0.831t) + 0.061\ sin(0.830t)),$

$y(t) = (10.031)e^{-t} + (0.5)e^t - (8.948)e^{1.228t} + 0.5e^tt + 2e^{0.356t}(-0.493\ cos(0.513t) +$
$33.959\ sin(0.513t)) + 2e^{-0.595t}(1.702\ cos(0.831t) + 0.930\ sin(0.831t)).$

ЛИТЕРАТУРА

1. *Рыбаков М.А.* Решение систем линейных дифференциальных уравнений с постоянными коэффициентами с помощью преобразования Лапласа. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14, вып. 4, 2009, 791-792.

2. *Malaschonok N.A.* An Algorithm for Symbolic Solving of Differential Equations and Estimation of Accuracy. Computer Algebra in Scientific Computing. LNCS 5743. Springer, Berlin, 2009, 213-225.

3. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14, вып. 4, 2009. С.744-748.

Ribakov M.A. Solving systems of linear differential equations with a piecewise continuous right-hand parts by means transformation Laplace. An algorithm for solving systems of linear differential equations with a piecewise continuous right-hand parts by means transformation Laplace is considered. Examples for solving such systems are received.

Key words: algorithm solution systems of differential equations, systems of differential equations, Laplace transform.

Поступила в редакцию 20 ноября 2009г.

УДК 004.421

ЭКСПЕРИМЕНТЫ С ПАРАЛЛЕЛЬНЫМ АЛГОРИТМОМ ВЫЧИСЛЕНИЯ ПРИСОЕДИНЁННОЙ МАТРИЦЫ И ПАРАЛЛЕЛЬНЫМ УМНОЖЕНИЕМ ФАЙЛОВЫХ МАТРИЦ[1]

© **А. А. Бетин**

Ключевые слова: вычисление присоединённой матрицы, параллельный алгоритм, кластер, файловые матрицы, произведение матриц.
Приводятся и обсуждаются результаты с параллельным алгоритмом вычисления присоединённой матрицы и параллельным умножением файловых матриц.

1 Эксперименты с параллельным алгоритмом вычисления присоединённой матрицы

В работе [1] был рассмотрен параллельный алгоритм вычисления присоединённой матрицы. Рассмотренный алгоритм был программно реализован для многопроцессорных вычислительных систем. Эксперименты с параллельным алгоритмом проводились на вычислительном кластере МВС-100К в МСЦ РАН.

[1]Работа выполнена при поддержке программы «Развитие потенциала высшей школы» (проект 2.1.1/1853) и Темплана 1.12.09.

В экспериментах использовались матрицы размеров 512×512, 1024×1024, 2048×2048 и 4096×4096. Присоединенная матрица вычислялась в конечном поле $\mathbb{Z}_p$, где $p<2^{32}$. Результаты экспериментов приведены в табл. 1 и 2 и на рис. 1 и 2. За ускорение вычислений на k процессорах по сравнению с вычислением на n процессорах примем число

$$a_{n,k}=(\frac{t_n}{t_k}-1)/(\frac{k}{n}-1).$$

Для вычисления ускорения в процентах будем умножать $a_{n,k}$ на 100%. Отметим, что при $t_n=t_k$ ускорение равно 0, а при $\dfrac{t_n}{t_k}=\dfrac{k}{n}$ ускорение равно 100%.

На рис. 1 и 2 кроме графиков ускорения изображены еще две прямые, которые показывают 50% и 100% ускорение.

В табл. 1 и на рис. 1 представлены результаты экспериментов, в которых использовались плотные матрицы.

Таблица 1

Время в секундах и ускорение вычислений присоединённой матрицы для плотной матрицы в зависимости от ее размера и количества процессоров

Количество процессоров	2	4	8	16	$a_{2,4}$	$a_{4,8}$	$a_{8,16}$
Размер матрицы							
512x512	11.463	11.571	12.521	13.512	-0.9%	-7.6%	-7.9%
1024x1024	71.332	58.945	67.312	71.478	21.4%	-12.4%	-5.8%
2048x2048	741.269	390.46	348.921	338.832	84.9%	11.9%	2.9%
4096x4096	5635.616	2919.601	2392.472	2201.979	93.0%	22.4%	8.7%

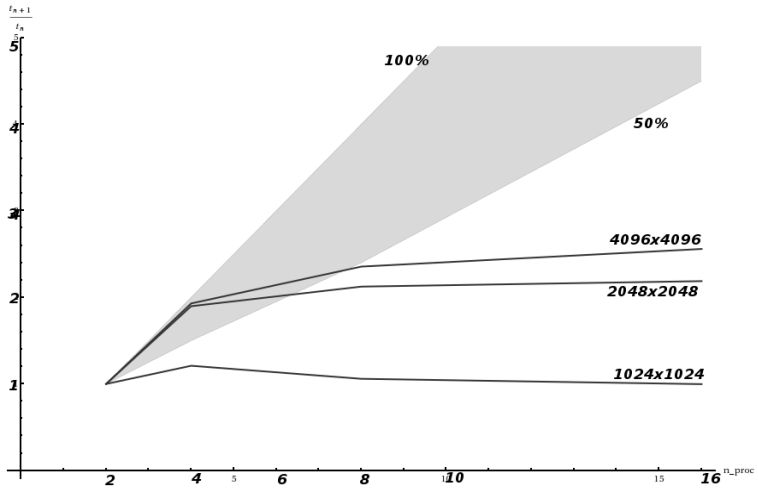

Рис. 1. Ускорение вычислений присоединённой матрицы для плотной матрицы в зависимости от количества процессоров *proc* по отношению к 2 процессорам

В табл. 2 и на рис. 2 представлены результаты экспериментов, в которых использовались матрицы с плотностью 1%.

Таблица 2

Время в секундах и ускорение вычислений присоединённой матрицы для матрицы плотностью 1% в зависимости от ее размера и количества процессоров

Количество процессоров	2	4	8	16	$a_{2,4}$	$a_{4,8}$	$a_{8,16}$
Размер матрицы							
512x512	3.737	4.572	5.098	5.500	-18.3%	-10.2%	-7.4%
1024x1024	40.014	31.512	27.919	28.478	27.0%	12.8%	-2.4%
2048x2048	470.877	346.985	258.497	264.100	35.8%	34.0%	-2.2%
4096x4096	3855.991	1929.697	1339.389	1241.251	100%	43.9%	7.7%

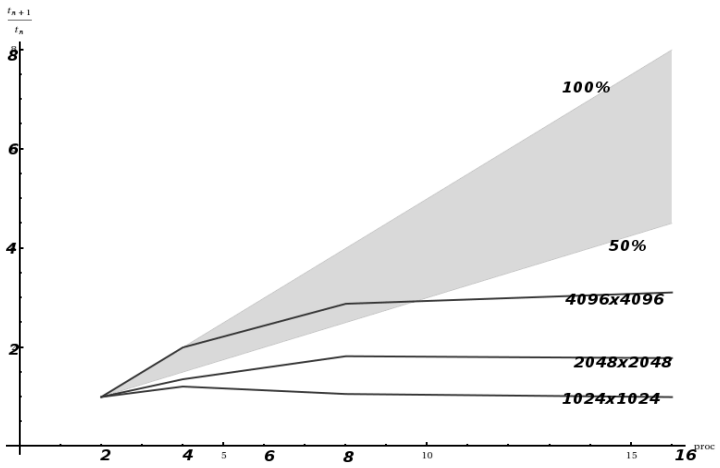

Рис. 2. Ускорение вычислений присоединённой матрицы для матрицы с плотностью 1% в зависимости от количества процессоров *proc* по отношению к 2 процессорам

Из табл. 1 и 2 видно, что вычисление присоединённой матрицы для матрицы размера 512x512 неэффективно делать параллельно, т. к. при увеличении числа процессоров время вычисления не уменьшается. Для матриц размера 1024x1024 и более ускорение наблюдается при увеличении числа процессоров с 2 до 16. При дальнейшем увеличении количества процессоров ускорение вычислений становится равным 0. Это связано с тем, что дальнейшее распараллеливание не выгодно, т. к. пересылаемые блоки становятся на столько малыми, что время пересылки приближается к времени вычислений.

Эксперименты показали, что с увеличением размера матриц параллельное вычисление становится более эффективным. Но с ростом размера матриц возникает проблема нехватки оперативной памяти. Для решения данной проблемы можно хранить матрицы не в оперативной памяти, а в виде файлов на жёстком диске, считывая только необходимые блоки матрицы. Размер максимального блока выбирается с учетом доступной оперативной памяти узла.

2 Эксперименты с параллельным умножением файловых матриц

В табл. 3 приведено время (время указано в секундах) вычисления произведения файловых матриц. Элементы матриц числа типа BigInteger 10 бит, плотность 100%.

Таблица 3

Время вычисления произведения файловых матриц

Количество процессоров	2	4	8	16
Размер				
2048x2048	970.827	475.192	266.895	142.546
4096x4096	13350.297	6286.368	3245.467	2622.430

На рис. 3 представлена зависимость ускорения вычислений с ростом количества процессоров по отношению к 2 процессорам для матриц размера 2048x2048 и 4096x4096 соответственно.

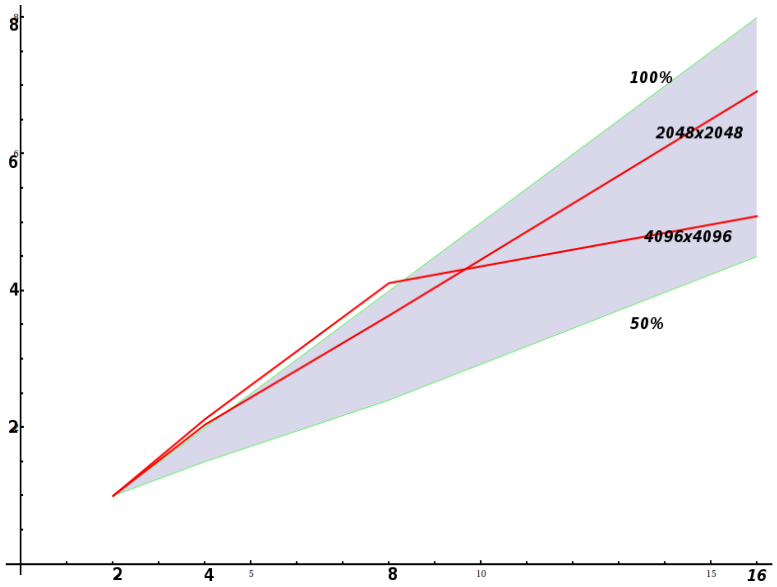

Рис. 3. Ускорение вычислений произведения файловых матриц в зависимости от количества процессоров *proc* по отношению к 2 процессорам

На обоих графиках кривая зависимости ускорения находится между прямыми, показывающими 50% и 100% ускорение, что говорит об эффективном распараллеливании.

ЛИТЕРАТУРА

1. *Бетин А.А.* Параллельное вычисление присоединённой матрицы. International conference Polynomial Computer Algebra. St.Petersburg, PDMI RAS, 2009, С. 123-128.

2. *Малашонок Г.И.* О вычислении ядра оператора, действующего в модуле. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 13, вып. 1, 2008. С. 129-131.

Betin A.A. Experiments with a parallel algorithm for calculation of adjoint matrix and with a parallel algorithm for multiplication of file matrices. Results of experiments with a parallel algorithm

for calculation of adjoint matrix and with a parallel algorithm for multiplication of file matrices are stated.

Key words: calculation of an adjoint matrix, parallel algorithm, file matrix.

Поступила в редакцию 20 ноября 2009г.

УДК 004.421

РЕАЛИЗАЦИЯ МЕТОДА ФУЖЕРА ВЫЧИСЛЕНИЯ БАЗИСА ПОЛИНОМИАЛЬНЫХ ИДЕАЛОВ[1]

© **М. В. Старов**

Ключевые слова: вычисление базиса Гребнера, параллельный алгоритм, кластер, метод Фужера, алгоритм F4.
Приводятся и обсуждаются результаты с параллельным алгоритмом вычисления базиса полиномиальных идеалов.

В работе [1] был рассмотрен параллельный алгоритм вычисления базиса Гребнера, основанный на методе гомоморфных образов в кольце полиномов с целочисленными коэффициентами.

В основе параллельного алгоритма лежит метод F4 [2], предложенный французским математиком Ж.-Ш. Фужером. Параллелизм основан на методе гомоморфных образов в кольце полиномов с целочисленными коэффициентами. На основе приведённого алгоритма были разработаны программы для параллельного вычисления базиса Гребнера полиномиальных идеалов, с которыми были проведены эксперименты на кластере ТГУ им. Г.Р. Державина.

В экспериментах использовались разреженные полиномы от пяти переменных.

Таблица 1

Время вычисления в секундах базиса Гребнера с использованием параллельного алгоритма

Количество процессоров k	2	4	8
Время t_k**, с**	259.5	134	77.5
t_2/t_k		1.9	3.34

[1]Работа выполнена при поддержке программы «Развитие потенциала высшей школы» (проект 2.1.1/1853)

Как видно из таблицы, при увеличении количества процессоров с 2 до 4 время уменьшилось в $1,9$ раза, при увеличении с 2 до 8 — в $3,34$ раза. Следовательно, для данного эксперимента эффективно использовать рассмотренный параллельный алгоритм для вычисления базиса Гребнера полиномиальных идеалов на 8 процессорах.

ЛИТЕРАТУРА

1. *Г.И. Малашонок, М.В. Старов, А.А. Бетин, О.Н. Переславцева, А.Г. Поздникин* Параллельная компьютерная алгебра. Часть 1. Учебное пособие. Тамбов: Издательский дом ТГУ им. Г.Р. Державина, 2009.

2. *Faugere J.-C.* A new efficient algorithm for computing Groebner bases (F4), J. Pure Appl. Algebra, 139:1–3 (1999), 61–88.

Starov M.V. Realization of Fougere's method for calculation of polynomial basis. Experiments with a parallel algorithm for calculation for ideals of polynomial basis are stated.

Key words: calculation of a Groebner basis, parallel algorithm, algorithm F4, Faugere's method, cluster.

Поступила в редакцию 20 ноября 2009г.

УДК 004.421

ЭКСПЕРИМЕНТЫ С ПАРАЛЛЕЛЬНЫМ АЛГОРИТМОМ ВЫЧИСЛЕНИЯ ХАРАКТЕРИСТИЧЕСКИХ ПОЛИНОМОВ МАТРИЦ В КОЛЬЦЕ ПОЛИНОМОВ[1]

© **О. Н. Переславцева**

Ключевые слова: вычисление характеристического полинома, параллельный алгоритм, кластер.

Приводятся и обсуждаются результаты экспериментов с параллельным алгоритмом вычисления характеристических полиномов полиномиальных матриц.

В работе [1] был рассмотрен параллельный алгоритм вычисления характеристических полиномов матриц, основанный на методе гомоморфных образов [2] в кольце полиномов с целочисленными коэффициентами. На основе приведённого алгоритма был разработан программный комплекс для параллельного вычисления характеристических полиномов

[1]Работа выполнена при поддержке программы «Развитие потенциала высшей школы» (проект 2.1.1/1853) и Темплана 1.12.09.

матриц в кольце полиномов, с которым были проведены эксперименты на кластере ТГУ им. Г.Р. Державина и на кластере МСЦ РАН.

Для оценки эффективности параллельного алгоритма введем понятие ускорения. Обозначим через t_k время вычисления задачи на кластере с k процессорами. При переходе от кластера с n процессорами к кластеру с k процессорами, $k > n$, ускорение достигает 100%, когда $t_n/t_k = k/n$. Ускорение равно нулю, когда $t_k = t_n$. Чтобы определить ускорение вычислений при других значениях t_n/t_k определим ускорение как функцию времени t_k.

О п р е д е л е н и е. Ускорением вычислений при переходе от n-процессорного кластера, где время вычислений равно t_n, к k-процессорному кластеру, где время вычислений равно t_k, $k > n$, называется функция $\alpha_{n,k} = (t_n/t_k - 1)/(k/n - 1) \cdot 100\%$.

Эксперимент 1 проводился на кластере ТГУ им. Г.Р. Державина. В эксперименте использовались плотные полиномиальные матрицы размера 40×40 над полиномами от двух переменных с 10-разрядными коэффициентами, старшая степень каждой переменной равна 2.

Зависимость времени вычисления от количества процессоров и ускорение вычислений при переходе от 2-процессорного кластера к 4-процессорному и 8-процессорному кластерам приведены в табл. 1.

Таблица 1

Время и ускорение вычислений характеристического полинома полиномиальной матрицы с использованием алгоритма, основанного на методе гомоморфных образов, в котором в конечном поле используется алгоритм Данилевского; порядок матрицы $n = 40$, разрядность числовых коэффициентов $b = 10$ бит, старшие степени переменных $deg = [2, 2]$.

Количество процессоров k	2	4	8
Время t_k**, с**	4480	2424	1631
Ускорение $\alpha_{2,k}$		85%	58%

Как показывают эксперименты, результаты которых приведены в табл. 1, ускорение вычислений составило 85% при переходе от 2-процессорного кластера к 4-процессорному кластеру и 58% при переходе к 8-процессорному кластеру.

Эксперимент 2 проводился на кластере МВС-100К в МСЦ РАН. В эксперименте использовались плотные полиномиальные матрицы размера 50×50 над полиномами от двух переменных с 10-разрядными коэффициентами. Старшие степени переменных равны 2: $deg = [2, 2]$.

График зависимости времени вычисления от количества процессоров приведён на рис. 1. В табл. 2 приведено ускорение вычислений при переходе от 1-процессорного кластера.

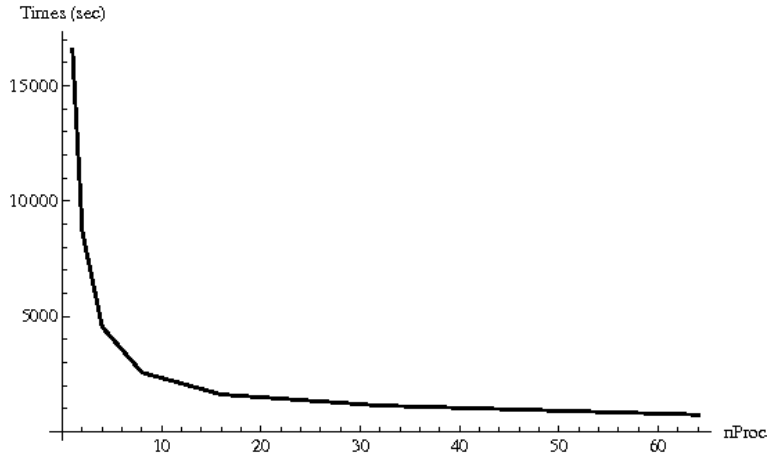

Рис. 1. Зависимость времени вычисления характеристического полинома с использованием алгоритма, основанного на методе гомоморфных образов, в котором в конечном поле используется алгоритм Данилевского [3], от количества процессоров $nProc$; порядок матрицы $n = 50$, разрядность числовых коэффициентов $b = 10$ бит, старшие степени переменных $deg = [2, 2]$.

Таблица 2

Время и ускорение вычислений характеристического полинома полиномиальной матрицы с использованием алгоритма, основанного на методе гомоморфных образов, в котором в конечном поле используется алгоритм Данилевского; порядок матрицы $n = 50$, разрядность числовых коэффициентов $b = 10$ бит, старшие степени переменных $deg = [2, 2]$.

Количество процессоров k	1	2	4	8	16	32	64	128	256
Время t_k**, с**	16558	8676	4548	2651	1626	1146	748	513	510
Ускорение $\alpha_{1,k}$		91%	88%	75%	61%	43%	34%	25%	12%

Как показывают эксперименты, при увеличении количества процессоров ускорение вычислений уменьшается и для некоторого числа процессоров дальнейшее распараллеливание становится не выгодно. Это объясняется тем, что пересылаемые блоки становятся на столько малыми, что время пересылки приближается к времени вычислений на граничном уровне. Как видно из табл. 2, для данного эксперимента неэффективно использовать рассмотренный параллельный алгоритм для вычисления характеристических полиномов матриц на 256 и более процессорах, т. к. время вычислений не уменьшается.

Проведённые эксперименты показали, что для полиномиальных матриц для некоторого числа процессоров пересылаемые блоки становятся на столько малыми, что время пересылки приближается к времени вычислений. Поэтому при увеличении количества процессоров ускорение вычислений становится близким к 0, начиная с некоторого числа процессоров. Это говорит о том, что дальнейшее распараллеливание не выгодно. Так, лучшее время вычисления характеристических полиномов матриц размера 50×50 над полиномами от двух переменных, старшие степени которых равны 2, с 10-разрядными коэффициентами на кластере МВС-100К в МСЦ РАН будет около 10 минут.

ЛИТЕРАТУРА

1. *Переславцева О.Н.* Параллельное вычисление характеристического полинома матрицы // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новгород, 30 марта – 3 апреля 2009 г.). Челябинск: Изд. ЮУрГУ, 2009. С. 817-821.

2. Компьютерная алгебра: Символьные и алгебраические вычисления. Пер. с англ./ Под ред. Б. Бухбергера, Дж. Коллинза, Р. Лооса. М.: Мир, 1986.

3. *Данилевский А.М.* О численном решении векового уравнения // Матем. сб. 1937. Т.2(44). N1. 169-172.

Pereslavtseva O.N. Experiments with a parallel algorithm for calculation of the characteristic polynomial for polynomials matrixes. Experiments with a parallel algorithm for calculation of the characteristic polynomial for polynomials matrixes on the MVS and TSU clusters are stated and discussed.

Key words: calculation of a characteristic polynomial, parallel algorithm.

Поступила в редакцию 20 ноября 2009г.

УДК 004.421

УМНОЖЕНИЕ ФАЙЛОВЫХ ПОЛИНОМОВ[1]

© **А. Г. Поздникин**

Ключевые слова: файловый полином, умножение файловых полиномов, параллельный алгоритм.

Предлагается параллельный алгоритм умножения полиномов, которые не помещаются в оперативную память. Такие полиномы назовём файловыми полиномами, т. к. они хранятся в файлах на жёстком диске. Приводятся и обсуждаются результаты с параллельным алгоритмом вычисления произведения файловых полиномов.

1 Введение

Большинство математических систем позволяют производить операции с полиномами, которые хранятся в оперативной памяти компьютера. А когда полином не помещается в оперативную пямять, такие системы оказываются непригодными. Так как объёмы используемой оперативной памяти обычно меньше дисковой, то можно хранить полиномы в файлах на жёстком диске. Полином, записанный в файл на жёстком диске в определённом формате, будем называть файловым.

[1]Работа выполнена при поддержке программы «Развитие потенциала высшей школы» (проект 2.1.1/1853)

В работе [1] описан формат хранения файловых полиномов, рассмотрены алгоритмы, реализующие выполнение основных арифметических операций с ними, и приведены результаты экспериментов. Очевидно, что время выполнения операций с файловыми полиномами, особенно операции умножения, очень быстро растет с ростом степеней исходных полиномов. Поэтому возникает задача разработки эффективного параллельного алгоритма умножения файловых полиномов.

2 Параллельный алгоритм умножения файловых полиномов

Рассматриваемый алгоритм умножения файловых полиномов является рекурсивным. В основе алгоритма лежит дихотомическое деление полиномов на части. Деление полинома на части происходит до тех пор, пока части станут таких размеров, которые позволяют выполнить умножение в оперативной памяти. Графом алгоритма является бинарное дерево. На листах дерева происходит умножение частей полиномов. В параллельном алгоритме деление на части происходит одновременно с распределением процессоров. Если свободные процессоры заканчиваются, а умножение частей полиномов по-прежнему не возможно произвести в памяти, то выполняется вызов однопроцессорного рекурсивного алгоритма.

Рассмотрим параллельный алгоритм умножения файловых полиномов A и B. Граф алгоритма изображен на рис. 1.

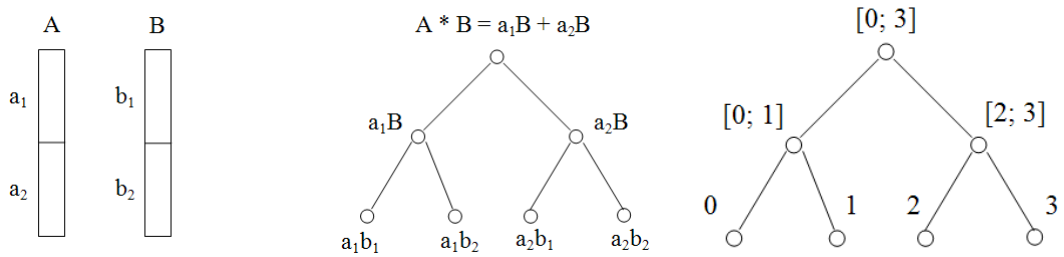

Рис. 1. Граф параллельного алгоритма умножения полиномов A и B и распределение четырех процессоров на узлах данного дерева

Выбирается больший из полиномов и делится на две части, которые будут занимать в оперативной памяти примерно равные объёмы. Пусть в нашем случае полином $A > B$, т. е. занимает большее количество памяти, чем B. На первом шаге делим полином A на две части a_1 и a_2, т. е. $A = a_1 + a_2$. Интервал с номерами процессоров [0;3] делится на два интервала [0;1] и [2;3]. Получим два узла на слое, на которых необходимо выполнить операции умножения $a_1 * B$ и $a_2 * B$. Эти операции не могут быть выполнены в оперативной памяти, поэтому необходимо продолжать деление полиномов на части. Для этого сравниваем a_1 с полиномом B, a_2 с полиномом B и выбираем из них большие. Пусть $B > a_1$ и $B > a_2$. Тогда делим B на части b_1, b_2 и на следующем слое на каждом процессоре вычисляем произведения $a_1 * b_1$, $a_1 * b_2$, $a_2 * b_1$, $a_2 * b_2$. Пусть мы достигли листовых вершин, когда данные операции возможно произвести в оперативной памяти соответственно на процессорах 0, 1, 2, 3. Следующими шагами станет подъём. Сначала это сумма полиномов $a_1 * B = a_1 * b_1 + a_1 * b_2$ и $a_2 * B = a_2 * b_1 + a_2 * b_2$. Результатом умножения станет сумма $A * B = a_1 * B + a_2 * B$.

3 Эксперименты

По приведенному выше алгоритму был разработан программный комплекс. Эксперименты проводились на кластере из 14 процессоров Intel Xeon 3 ГГц, 1 Гб. В экспериментах использовались полиномы от двух переменных, полученные случайным образом, с целыми 8-разрядными коэффициентами, с количеством мономов $16 * 10^4$. Принято, что максимально допустимое количество мономов в полиномах, когда операцию умножения возможно произвести в оперативной памяти, равно 10^4.

Ускорением вычислений при переходе от n_0-процессорного кластера, где время вычислений равно T_0, к k–процессорному кластеру, где время вычислений равно T_k, $k > n_0$, принята функция $\alpha(T_k) = (T_0/T_k - 1)/(k/n_0 - 1) * 100\%$.

Результаты экспериментов приведены в табл. 1. К примеру, при переходе от однопроцессорной машины к двухпроцессорному кластеру ускорение равно 88,64 %. При переходе от однопроцессорных вычислений к четырехпроцессорным ускорение равно 89,84 %. Когда мы используем 6, 10, 12, 14 процессоров, ускорение заметно снижается. Это связано с тем, что числа 6, 10, 12, 14 не являются степенями двойки, а у нас используется граф алгоритма в виде бинарного дерева.

Таблица 1

Значения времени выполнения операции умножения полиномов на n процессорах и ускорения вычислений на n процессорах по сравнению с вычислениями на одном процессоре

Число процессоров	1	2	4	6	8	10	12	14
Время умножения, с	11156	5914	3019	2753	1688	1701	1754	1660
Ускорение		88,64	89,84	61,05	80,13	61,76	48,73	44

На следующих графиках представлена зависимость времени вычислений от количества процессоров и ускорение вычислений на n процессорах по сравнению с вычислениями на одном процессоре.

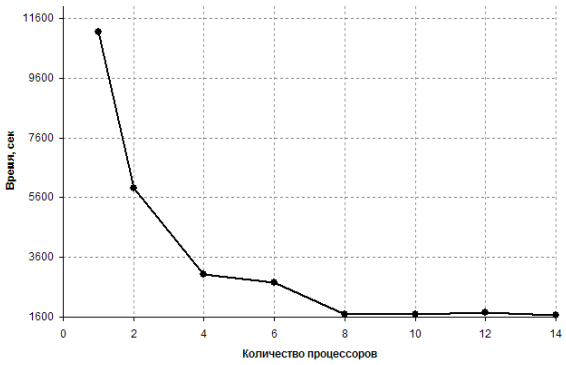

Рис. 2. График зависимости времени выполнения операции умножения полиномов от количества процессоров

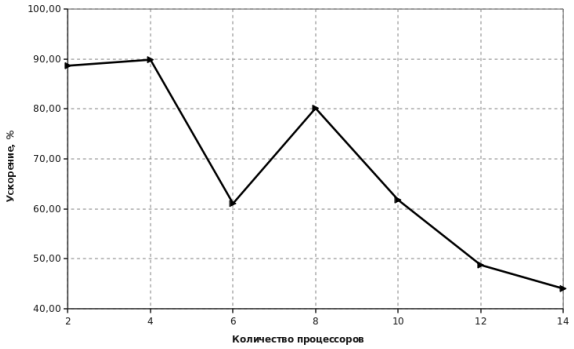

Рис. 3. Ускорение выполнения операции умножения на n процессорах по сравнению с умножением на одном процессоре

4 Заключение

Реализованный параллельный алгоритм умножения файловых полиномов показал свою эффективность и может быть применён в задачах, которые используют умножение полиномов больших размеров. Так как в основе алгоритма лежит дихотомическое деление полиномов на части, для вычислений рекомендуется использовать количество процессоров $n = 2^k$, где k — натуральное число. Оптимальное число процессоров для быстрого умножения файловых полиномов можно подобрать экспериментально, учитывая ускорение вычислений.

ЛИТЕРАТУРА

1. *Поздникин А.Г.* Файловые полиномы. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14, вып. 4, 2009. С.783-785.

2. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14, вып. 4, 2009. С.744-748.

Pozdnikin A.G. Multiplication of file polynomials. An algorithm for multiplication of file polynomials is described. Experimental results with that parallel algorithm on TSU cluster are stated.

Key words: file polynomials, multiplication of file polynomials, parallel algorithm.

Поступила в редакцию 20 ноября 2009г.

УДК 004.421

ВЫЧИСЛЕНИЕ НЕОПРЕДЕЛЁННОГО ИНТЕГРАЛА ДРОБНО-РАЦИОНАЛЬНОЙ ФУНКЦИИ[1]

© **С. М. Тарарова**

Ключевые слова: дробно-рациональные функции, интеграл дробно-рациональной функции.

Рассматривается задача вычисления неопределённого интеграла дробно-рациональной функции. Предлагается метод, который позволяет избежать приближённых вычислений и получить точный ответ.

Теория интегрирования дробно-рациональных функций известна с XIX в. При традиционном подходе знаменатель функции раскладывается на линейные множители, используя вычисление всех корней знаменателя в поле комплексных чисел. Полученные корни являются приближёнными. В результате получается неопределённый интеграл, в котором присутствуют приближённые числа.

Однако во многих случаях можно избежать приближённых вычислений. Опишем способ вычисления неопределённого интеграла дробно-рациональных функций, который позволяет получить точное решение, если оно существует.

Пусть $f(x) = \frac{q(x)}{r(x)}$, где $q(x),\ r(x) \in \mathbb{Q}[x],\ deg\ q(x) < deg\ r(x)$, правильная дробно-рациональная функция.

Разложим полином $r(x)$ на неприводимые множители $r(x) = r_1^{n_1}(x) r_2^{n_2}(x) \dots r_k^{n_k}(x)$ в $\mathbb{Q}[x]$. Тогда функцию $f(x)$ можно представить в виде суммы правильных дробей

$$f(x) = \sum_{i=1}^{k} \sum_{j=1}^{n_i} \frac{q_{ij}(x)}{r_i^j(x)},$$

где $q_{ij} \in \mathbb{Q}[x],\ r_i \in \mathbb{Q}[x],\ deg\ q_{ij} < deg\ r_i,\ r_i(x)$ – неприводимые полиномы, $j = 1, \dots, n_i,\ i = 1, \dots, k$.

Таким образом, задача сводится к интегрированию правильных дробей вида $\frac{q(x)}{r^j(x)}$, где $r(x)$ — неприводимый полином.

Так как $r(x)$ и $r'(x)$ — взаимно простые, то существуют полиномы $a \in \mathbb{Q}[x],\ b \in \mathbb{Q}[x]$, удовлетворяющие соотношению $ar + br' = 1$ [1]. Тогда

$\int \frac{q}{r^n} dx = \int \frac{q(ar+br')}{r^n} dx = \int \frac{qa}{r^{n-1}} dx + \int \frac{qbr'}{r^n} dx = \int \frac{qa}{r^{n-1}} dx + \int \left(\frac{(qb/(n-1))'}{r^{n-1}} - \left(\frac{qb/(n-1)}{r^{n-1}} \right)' \right) dx =$
$- \frac{qb/(n-1)}{r^{n-1}} + \int \frac{qa + (qb/(n-1))'}{r^{n-1}} dx.$

Продолжим аналогичные преобразования получившегося интеграла до тех пор, пока в знаменателе подинтегральной дроби получим r^1. Тогда

$$\int \frac{q}{r^n} dx = -\frac{q_n b/(n-1)}{r^{n-1}} - \frac{q_{n-1} b/(n-2)}{r^{n-2}} - \dots - \frac{q_k b/(k-1)}{r^{k-1}} - \dots - \frac{q_2 b}{r} + \int \frac{q_1}{r} dx =$$

[1]Работа выполнена при поддержке программы «Развитие потенциала высшей школы» (проект 2.1.1/1853)

$$= -b\sum_{j=1}^{n-1}\frac{q_{j+1}}{jr^j} + \int\frac{q_1}{r}dx,$$

где

$$q_n = q,$$
$$q_k = \frac{1}{k}(q_{k+1}(ak+b') + q'_{k+1}b), k = n-1,\ldots,1,$$
$$q'_k = \frac{1}{k}(q_{k+1}(ka'+b'') + q'_{k+1}(ak+2b') + q''_{k+1}b).$$

Итак, получили интеграл от правильной дроби, в знаменателе которой стоит неприводимый полином в первой степени.

Если $q_1 = cr'$, то искомый интеграл равен

$$\int\frac{q_1}{r}dx = c\ ln(r).$$

Если c не является константой, то исходная функция неинтегрируема без алгебраических расширений. Следовательно, можем найти только приближённое значение интеграла, если он существует [2]. Например, даже такой интеграл

$$\int\frac{Mx+N}{x^2+px+q}\,dx = \frac{M}{2}ln\ \mid x^2+px+q\mid + \frac{2N-Mp}{2\sqrt{p^2-4q}}ln\left|\frac{2x+p-\sqrt{p^2-4q}}{2x+p+\sqrt{p^2-4q}}\right| + C$$

вычисляется в расширении поля $\mathbb{Q}$.

Описанный метод позволяет как можно дольше выполнять действия в поле рациональных чисел и только при необходимости перейти в поле действительных чисел или, если это необходимо, в поле комплексных чисел.

ЛИТЕРАТУРА

1. *Панкратьев Е.В.* Элементы компьютерной алгебры (Конспекты спецкурса). - М.:Механико-математический факультет МГУ. - 2007.

2. *Дэвенпорт Д., Сирэ И., Турнье Э.* Компьютерная алгебра: Пер с франц. - М.: Мир,1991.

3. *Фихтенгольц Г.М.* Курс дифференциального и интегрального исчисления. Т. 1, 2. - М.: ФИЗМАТЛИТ, 2001.

4. *Калинина Е.А., Утешев А.Ю.* Теория исключения: Учеб. пособие. - СПб.: Изд-во НИИ химии СПбГУ, 2002.

5. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14, вып. 4, 2009. С.744-748.

Tararova S.M. Calculation of integral of the fractional-rational function. The problem of calculation of integral of the fractional-rational function is considered. The method which allows to avoid the approached calculations and to receive the exact answer is offered.

Key words: fractional-rational functions, integral of fractional-rational functions.

Поступила в редакцию 20 ноября 2009г.

УДК 004.421

ВИЗУАЛИЗАЦИЯ ФУНКЦИИ ПЛОТНОСТИ СРЕДЫ [1]

© **Ю. Ю. Юрин**

Ключевые слова: функция плотности, машинная графика.

Излагается один из методов визуализации функции плотности среды.

Визуализация графиков функций имеет не только эстетическое, но и практическое значение. После того, как функция становится зримой, анализировать её гораздо проще. На графиках можно проследить такие особенности, как положения экстремальных точек, точек бифуркаций и др. Превращение любой информации в визуальную является актуальной задачей, т. к. для человека зрение — основной высокоскоростной канал получения информации.

Сейчас существует огромное множество компьютерных программ для построения графиков функций, как самостоятельных, так и включенных в состав специализированных математических пакетов, таких как Maple [1], Mathcad [2], Mathematica [3], MATLAB [4] и др. Они предоставляют возможность построения 2D- и 3D-графиков функций в параметрическом и явном виде.

В задачах естествознания и техники часто возникает потребность проникнуть внутрь трёхмерного объекта и показать его строение. При этом непосредственно увидеть внутреннее строение изучаемого объекта невозможно. Примерами таких объектов могут быть галактики, различные физические поля, трёхмерное распределение плотности в жидких и твёрдых средах, а также микроструктуры и наноструктуры реального мира. Вычислительные средства, обеспечивающие такую визуализацию, представляют интерес при исследовании различных явлений в физике, медицине, геологии, метеорологии и других областях.

Возникает потребность в новых средствах визуализации, которые бы предоставили пользователю следующие возможности:

1)построение трёхмерного изображения в виде полупрозрачных изоповерхностей;

2)выбор цвета, количества изоповерхностей и правил построения пользователем, исходя из стоящей перед ним задачи;

3)произвольное вращение, перемещение, масштабирование сцены;

4)построение изображения сечений объекта произвольными плоскостями.

Трёхмерные данные могут быть получены различными способами, и от этого зависит их входное представление в памяти компьютера. Когда мы получаем данные о реальных объектах от каких-либо приборов, то получаем не явно заданную функцию, а лишь набор значений в некоторых точках, т. е. табличное представление. В зависимости от алгоритма визуализации может потребоваться переход от табличного задания к аналитическому (например, с помощью кусочно-гладких функций).

[1]Работа выполнена при поддержке программы «Развитие потенциала высшей школы» (проект 2.1.1/1853)

Невозможно получить, а тем более хранить абсолютно всю информацию об объекте. Следовательно, в памяти нужно хранить информацию в сжатом или аппроксимированном виде. От способа представления трёхмерных данных зависит количество требуемой памяти, скорость работы алгоритма визуализации и точность получаемой проекции. Эти данные являются входными для алгоритма визуализации. Преобразование входных трёхмерных данных в выходные (двухмерный массив пикселей) должно происходить по определённым правилам, суть которых состоит в том, чтобы зрительное ощущение человека, наблюдающего двухмерное изображение, построенное алгоритмом визуализации, соответствовало ощущению, которое наблюдатель получил бы, если бы рассматривал сцену в реальной действительности. Для того чтобы изображения, сформированное на сетчатке глаза от реального объекта и от его проекции на экране, максимально совпадали, необходимо обеспечить сохранность всей информации о сцене.

Архитектура приложения должна представлять собой несколько компонент, последовательно преобразующих исходные данные, а также предусматривать наличие интерфейса пользователя для управления преобразованиями. Основные формы представления объекта в памяти ЭВМ:

1) входной образ оъекта (набор изображений плоскостных срезов);

2) трёхмерный массив;

3) совокупность изоповерхностей;

4) изображение в z-буфере [6];

5) изображение на экране.

Опишем подробнее компоненты и происходящие в них преобразования.

1. С помощью некоторого прибора информация о физических свойствах реального объекта, например, распределение плотности вещества, преобразуется в набор изображений плоскостных срезов. Это будут входные данные.

2. Изображения срезов «склеиваются» друг с другом, образуя трёхмерный массив, элементами которого являются значения плотности в точках пространства.

3. Трёхмерный массив преобразуется в совокупность поверхностей одинакового уровня — изоповерхностей. Для хранения в памяти изоповерхности аппроксимируются с помощью полигонов (обычно треугольников), или с помощью кусочно-гладких функций.

4. Далее происходит проективное преобразование объекта на плоскость экрана по правилам центрального проецирования. Удаление невидимых и наложение полупрозрачных частей происходит по алгоритму z-буфера.

5. На завершающем этапе двухмерный массив z-буфера хранит в себе готовое изображение, которое выводится на экран.

ЛИТЕРАТУРА

1. *Дьяконов В. П.* Maple 9 в математике, физике и образовании. М.: СОЛОН-Пресс, 2004.

2. *Очков. В. Ф.* Mathcad 14 для студентов и инженеров: русская версия. СПб.: BHV, 2009.

3. *Шмидский Я. К.* Mathematica 5. Самоучитель. М.: Диалектика. 2004.

4. *Дьяконов В. П.* MATLAB 6/6.1/6.5 + Simulink 4/5. Основы применения. Полное руководство пользователя. –М.: СОЛОН-Пресс, 2002.

5. *Порев В. Н.* Компьютерная графика. – СПб.: БХВ-Петербург, 2002.

6. *Никулин Е. А.* Компьютерная геометрия и алгоритмы машинной графики. — СПб: БХВ-Петербург, 2003.

Yurin Yu.Yu. Visualization of density function of medium. A method of visualization of density function of medium is stated.

Key words: density function, the machine drawing.

Поступила в редакцию 20 ноября 2009г.