Параллельные алгоритмы компьютерной алгебры

Г.И. Малашонок, А.И. Аветисян, Ю.Д. Валеев, М.С. Зуев.

Аннотация. В работе рассматривается разрабатываемая в рамках среды ParJava система компьютерной алгебры. Цель разрабатываемой системы – предоставить возможность эффективного использования параллельных вычислительных систем для проведения аналитических расчетов. В силу разреженности данных система включила в себя несколько версий каждой решаемой задачи за счет различных методов распределения данных между процессами параллельной программы. Статья состоит из 5 разделов. В разделе 2 содержится краткое описание среды ParJava. Перечисляются входящие в нее инструментальные средства. В разделе 3 рассматривается применение инструментов среды для решения задач оптимизации программы, возникающих во время ее разработки. Подробно описывается применение нового метода анализа профилей параллельной программы, позволяющего работать с иерархическим представлением численных характеристик. В разделе 4 описываются разработанные программы компьютерной алгебры, приводятся результаты модельных расчетов, проводимых на кластере ИСП РАН. Раздел 5 подводит итоги проведенной работы.

1. Введение

Используемые сегодня системы компьютерной алгебры, такие как MAPLE, Mathematica, CoCoA, Reduce, Derive и другие, находят применение при решении задач во многих разделах физики, химии, биологии, в задачах моделирования в электротехнике, робототехнике и других областях.

Эти системы предназначены для работы на персональных рабочих станциях и поэтому ограничены задачами небольшой вычислительной сложности. Они опираются на парадигмы программирования 80х-90х годов, когда закладывались основы для этих систем, и ориентированы на последовательную организацию вычислений – их базовые структуры данных являются последовательными структурами.

Аналитические расчеты носят характер задач высокой вычислительной сложности, у которых происходит быстрый рост сложности вычислений при росте объемов входных данных. По этой причине для проведения аналитических вычислений требуется разработка параллельных алгоритмов и проведение расчетов на многопроцессорных вычислительных комплексах, требуется создание системы параллельной компьютерной алгебры. Создание

такой параллельной системы представляется перспективным в связи с развитием вычислительных сетей и технологий GRID.

Цель этой статьи – обсудить результаты экспериментов с параллельными алгоритмами компьютерной алгебры, реализованными в среде ParJava и проведенными на вычислительном кластере ИСП РАН.

Каждая параллельная программа выполнялась на высокопроизводительном кластере ИСП РАН, состоящем из восьми узлов, связанных сетями Myrinet и Fast Ethernet. Вычислительный узел кластера работает под управлением операционной системы Linux с ядром версии 2.4.20-8, содержит два процессора Athlon с тактовой частотой 1533 MHz и один гигабайт оперативной памяти. Для передачи данных во время работы параллельной программы используется сеть Myrinet, обеспечивающая полнодуплексный канал с пропускной способностью 2 Gbit/sec; топология сети Myrinet представляет собой сеть Клоза (Clos network). Сеть Fast Ethernet используется только для управления работой кластера: монтирование файловых систем, запуск процессов параллельных программ, сбор данных системного мониторинга вычислительных узлов.

2. Краткое описание среды ParJava

Среда ParJava обеспечивает возможность разработки Java-программ, параллельных по данным, расширяя среду Java стандартным интерфейсом MPI, реализующим симметричные коммуникации. От реализации MPI в среде Java требуется, чтобы она минимизировала латентность и другие накладные расходы на организацию коммуникаций. Этим требованиям удовлетворяет реализация функций MPI в среде Java в виде «привязки» к соответствующим функциям реализации MPI в среде C. Такой подход позволяет использовать высокоэффективные реализации MPI, учитывающие специфику аппаратуры используемого коммуникационного оборудования. Реализация MPI в среде Java удовлетворяет этим условиям.

Среда ParJava предоставляет прикладному программисту набор инструментов, позволяющих во время разработки Java-программы на инструментальном компьютере исследовать динамические характеристики как программы в целом, так и ее частей, и использовать эту информацию для улучшения своей программы. В среде ParJava имеется три группы инструментов: анализаторы (меню *Analyze*), преобразователи (меню *Transform*) и интерпретаторы (меню *Run*). В текущей версии среды ParJava доступны следующие инструменты: (а) *анализаторы*: *Sequential* – выделение последовательной части параллельной программы; *AmdahlRatio* – вычисление отношения Амдаля [1]; *ForLoop* – анализ возможности распараллеливания заданного цикла *for* с помощью Омега-теста или теста расстояний [2]; *Slice* – построение обратного динамического слайса для заданного набора переменных в заданной точке программы; (б) *преобразователи*: *Transform to IC* – построение внутреннего представления параллельной программы; *Compile* – компиляция внутреннего представления отдельного файла параллельной программы в байт-код; *Build Project* – сборка

параллельной программы; *Instrumentate* – инструментирование параллельной программы с компенсацией обращений к инструментальным функциям; (в) интерпретаторы: *Exec* – выполнение параллельной программы; *Simulate* – интерпретация модели параллельной программы.

Во время интерпретации модели параллельной программы активизируется диалоговое окно, в котором пользователь может задавать параметры интерпретации: описание узлов целевого кластера, описание его коммуникационной сети, количество вычислительных узлов кластера. В результате интерпретации получается оценка времени работы программы, определяются границы области масштабируемости (по Амдалю [1] или по Густафсону [3]).

В настоящее время в среду ParJava интегрируется механизм создания контрольных точек параллельных программ [4]. Поскольку существующие реализации JVM не обеспечивают возможности сохранения своего состояния, была разработана и реализована методика сохранения локальных (стековых) переменных и содержимого кучи JVM. С рядом ограничений (одна нить в каждом процессе, отсутствие пользовательских native-методов), такая методика позволяет восстанавливать состояние JVM по данным, сохраненным на диске. Целостность восстановленного состояния обеспечивается за счет использования коммуникационно-управляемого протокола [5] создания контрольных точек. Реализован механизм, позволяющий оптимизировать расположение контрольных точек в программе за счет использования профиля используемой памяти: если короткоживущие переменные занимают достаточно много места, создание контрольной точки откладывается.

3. Использование инструментов среды ParJava

Рассмотрим некоторые инструментальные средства среды ParJava, применявшиеся при разработке программ компьютерной алгебры. Средства использовались для анализа динамических характеристик программы и для выявления фрагментов кода, оптимизация которых дает максимальный эффект.

Разрабатывая прикладную параллельную программу в рамках среды ParJava, прикладной программист взаимодействует с графическим интерфейсом среды. Он вводит исходный код программы в окне редактора и помещает введенные файлы в проект, в рамках которого хранятся исходный код параллельной программы, ее байт-код, ее внутреннее представление (статические модели классов), ее модель, ее трассы и профили.

Инструментальные средства среды становятся доступными после того, как пользователь выполнит трансляцию исходного кода во внутреннее представление (строятся статические модели классов): с помощью мыши отмечает транслируемые файлы в диалоговом окне управления проектом и вызывает транслятор. В результате трансляции во внутреннее представление для каждого указанного файла исходного кода (содержащего один или несколько классов) строится абстрактное синтаксическое дерево, которое

затем передается преобразователю, строящему набор статических моделей классов – объектов класса `FileInfo`. Трансляция каждого класса выполняется независимо. Это позволяет в случае изменения исходного кода отдельного класса Java-программы повторно транслировать только файл, содержащий данный класс. После трансляции исходного кода в окне управления проектом начинает отображаться информация о структуре класса: каждый файл исходного кода отображается как корень дерева, потомок которого – класс, описанный в данном файле, для каждого класса в виде потомков выводятся методы и поля данного класса.

Выявление критических фрагментов кода выполняется при доводке параллельной программы, когда прикладной программист ищет код, работа которого сильнее всего влияет на характеристики всей программы. Влияние возникает за счет того, что код выполняется достаточно много раз (тела циклов) или занимает большую часть времени работы программы (например, из-за использования библиотек). Ручная оптимизация такого кода позволяет ощутимо улучшить всю программу в целом. Поскольку относительно всей программы объем критического кода достаточно мал (как правило, он составляет 10-20%), оптимизировать только его проще и эффективней по времени, чем всю программу. Для выявления критического кода необходим профиль параллельной программы. Для облегчения работы пользователя с профилем, среда ParJava предоставляет возможность визуализации – числовые значения профиля отображаются на набор цветов (оттенки серого: от белого до черного), а исходный код программы подсвечивается определенным цветом, согласно полученному для него числовому значению характеристики. Если программа достаточно большая, такая методика уже не дает требуемой простоты и комфортности в работе. Поэтому в большинстве систем [6, 7], применяемых для доводки параллельных программ, используется иерархическая навигация по профилю. В среде ParJava также были разработаны и реализованы средства, позволяющие изучать профиль поэтапно, переходя от крупных фрагментов программы к более мелким. Рассмотрим порядок поиска критических фрагментов на примере анализа частотного профиля.

Сбор и анализ частотного профиля программы заключается в последовательном использовании инструментов среды: применение инструментатора к файлам исходного кода, отладочное выполнение программы, работа со средствами визуализации. Вопросы, связанные с техникой сбора профилей параллельной программы освещены в работе [8].

Собранный частотный профиль визуализируется с помощью инструментов среды ParJava в основном окне, содержащем исходный код программы, и окне визуализатора профиля. Окно визуализатора разделено на четыре части, три из которых используются для отображения данных, а четвертое содержит элементы управления (см. рис 1).

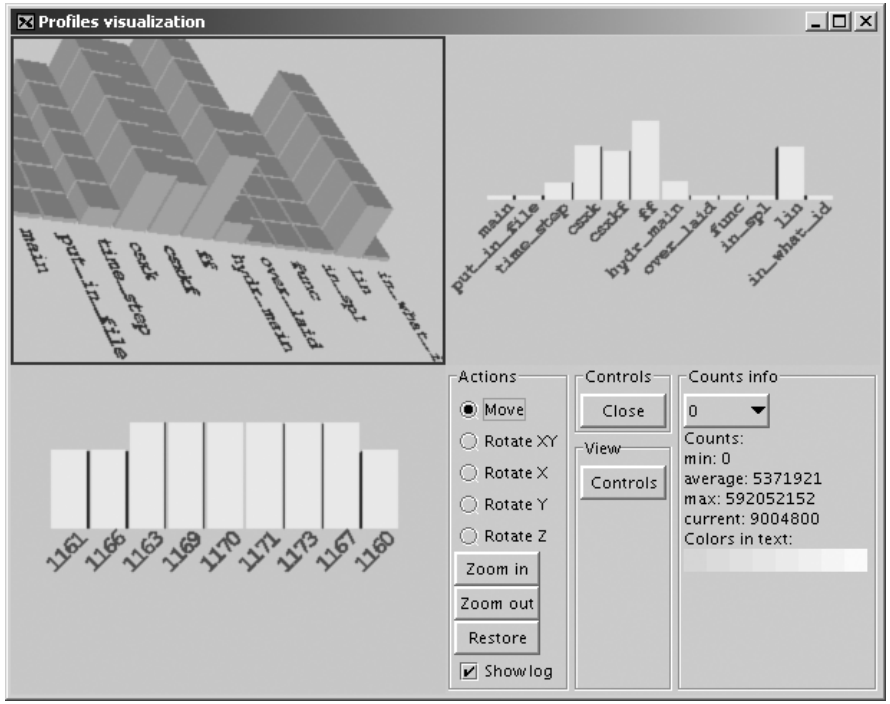

Рис. 1. Окно визуализатора с загруженным профилем.

Верхнее левое поле, *поле программы*, отображает частный профиль методов параллельной программы. Значения показываются в виде трехмерной гистограммы, столбцы которой расположены в виде матрицы. Строки соответствуют методам классов, входящих в программу, столбцы – процессам параллельной программы. Пользователь может двигать, вращать и приближать/удалять гистограмму. Такое представление данных предпочтительней цветового отображения, за счет того, что пользователь лучше видит «пики» и «провалы». Цветовое отображение предполагает, что одному цвету соответствует целый диапазон значений, и такая методика отображения результатов не позволяет сразу увидеть различие в характеристиках методов (или базовых блоков, если рассматривать профили более подробно). Выделение строки или столбца позволяет выбрать характеристики только одного метода во всех процессах (строка) программы или всех методов в одном процессе (столбец) и отобразить выделенные данные в верхнем правом поле окна – поле *среза*. В первом случае подписи гистограммы показывают номер процесса, во втором – имя метода.

Если профилирование выполнялось на уровне базовых блоков, то становится доступно для использования нижнее левое поле – *поле базовых блоков*. Данное

поле отображает гистограмму для значений характеристик базовых блоков выбранного метода в выбранном процессе. Выбор можно сделать либо через поле программы, либо через поле среза. Подписи столбцов гистограммы в поле базовых блоков показывают числовые идентификаторы базовых блоков. Нажав мышью на столбец гистограммы, пользователь получает в главном окне среды выделенный цветом соответствующий базовый блок в исходном коде, а в окне визуализатора отображается частота выполнения данного блока.

Каждое из полей отображения гистограмм может быть переведено в расширенный режим, когда оно занимает все окно визуализатора. Помимо обычной шкалы, поддерживается логарифмическая шкала, повышающая информативность гистограммы, в случае, когда высота столбцов существенно различается.

Получение профиля параллельной программы может выполняться как на целевом вычислительном кластере, так и на инструментальном компьютере с помощью интерпретации. В интерпретации используется модель параллельной программы, строящаяся на основе внутреннего представления. Использование интерпретации позволяет избежать простоев в процессе доводки программы, возникающих в силу ожидания доступа к целевому кластеру, являющемуся ресурсом коллективного пользования. Подробно вопросы построения модели и ее интерпретации рассмотрены в работе [9].

4. Параллельные программы компьютерной алгебры

Можно выделить три типа параллельных программ компьютерной алгебры: статический, динамический и комбинированный.

Пусть вычислительный граф задачи разбит на узловые подграфы, т.е. подграфы, вычисление которых происходит на одном процессорном узле. Если закрепление узлового подграфа за конкретным узлом происходит до начала вычислений, то такую параллельную программу называем статической. Если закрепление узлового подграфа за конкретным узлом происходит непосредственно перед вычислением по этому подграфу, в зависимости от состояния загруженности узлов, то такую программу называем динамической. Программа комбинированного типа – это программа, в которой предусмотрено переключение статического режима в динамический.

Задачи компьютерной алгебры имеют дело, как правило, с существенно разреженными данными. Заранее неизвестно, как будут загружены отдельные узлы. Поэтому статические программы могут оказаться неэффективными. Если же данные однородные и загрузку узлов можно рассчитать заранее, то, очевидно, статические программы предпочтительнее любых других типов.

4.1. Умножение полиномов

Были разработаны программы комбинированного типа для умножения полиномов многих переменных. Вычислительный граф параллельной программы имеет вид бинарного дерева: каждый из полиномов сомножителей

представляется суммой двух полиномов, а произведение представляется суммой четырех произведений, в которых участвуют эти слагаемые.

Ниже, на рис. 2 и 3, показаны результаты двух вычислительных экспериментов с параллельными программами умножения полиномов. На рисунках представлены графики роста скорости вычислений при увеличении числа процессоров, участвующих в вычислениях.

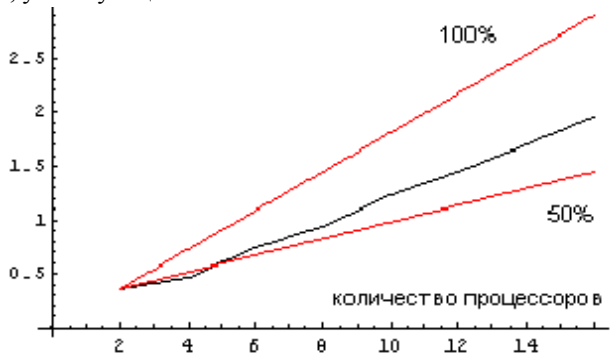

Рис. 2. Вычисление произведения полиномов в конечном поле. Коэффициент ускорения равен 76%.

На этих графиках и на всех графиках дальше по горизонтальной шкале откладывается количество процессоров, участвующих в вычислениях, а по вертикальной шкале – величина, обратная времени вычислений. За единицу времени принято 100 сек. Верхний луч соответствует 100% роста скорости (теоретический предел), нижний луч соответствует росту скорости, равному 50%.

В первом эксперименте вычислялось произведение двух полиномов от трех переменных, содержащих по 17 тысяч мономов. Коэффициенты брались из конечного поля, характеристика которого не превосходила $2 \cdot 10^8$. Коэффициент ускорения при переходе от двух к шестнадцати процессорам составил 76%.

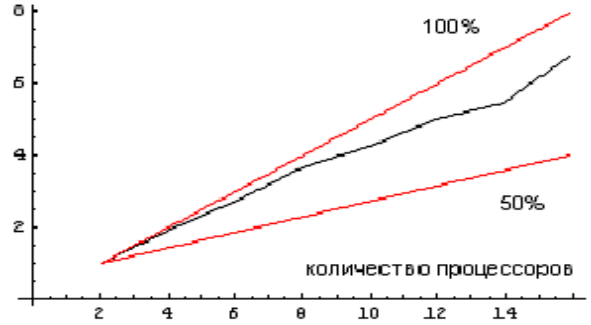

Рис. 3. Вычисление произведения полиномов с целыми коэффициентами. Коэффициент ускорения равен 85%.

Во втором эксперименте вычислялось произведение таких же по размеру полиномов, коэффициентами которых были случайные целые числа, не превосходящие по модулю числа 2^{28}. Коэффициент ускорения при переходе от двух к шестнадцати процессорам составил 85%.

4.2. Матричные операции

Самыми распространенными формами хранения матриц являются стандартная построчная форма хранения всех коэффициентов матрицы (DM), которая применяется для хранения плотных матриц, а также построчная форма хранения ненулевых коэффициентов матрицы и их индексов (SM), которая применяется для хранения разреженных матриц.

Для организации эффективных рекурсивных параллельных матричных операций необходимо иметь соответствующий формат для хранения матриц.

Таким естественным форматом может служить формат кватернарного дерева (QT). Дерево строится следующим образом. Если порядок матрицы 2^n, то она может быть разбита на четыре квадратных блока порядка 2^{n-1}, которые также могут быть разбиты на четыре равных блока, и так – вплоть до элементов. Если блокам поставить в соответствие вершины, а ребрами соединить вершины, соответствующие блокам, с вершинами, соответствующими его подблокам, то в результате получим кватернарное дерево. Если некоторый блок нулевой, то соответствующее ему ребро в дереве отсутствует. Это обеспечивает эффективный способ хранения разреженных матриц (см. также [10, [11]). Отметим, что при необходимости всякая матрица может быть дополнена до матрицы размера 2^n введением дополнительных нулевых или единичных элементов. Заметим, что обозначения QT, DM и SM происходят от названий: quaternary tree, density matrix и sparse matrix.

Если время вычисления суммы и произведения элементов матриц примерно одинаковое, то применение схемы умножения Штрассена не приводит к ускорению вычислений, если порядок матрицы менее 1287 (см. оценки в [12]). Поэтому в первую очередь были разработаны параллельные программы для стандартного алгоритма умножения матриц. При этом использовались две схемы – простая параллельная схема умножения и рекурсивная блочная схема умножения.

4.3. Простая параллельная схема умножения матриц

Вычисление произведения двух матриц можно осуществить с помощью представления первого матричного сомножителя в виде вектора матричных блоков. В результате умножения каждого такого блока на второй сомножитель получаем соответствующий блок матрицы, являющейся произведением. Здесь использован естественный параллелизм задачи умножения матриц.

Был реализован статический вариант такой параллельной программы, в которой число блоков, на которые разбивается первый сомножитель, просто

выбирается равным числу процессоров, участвующих в вычислении, а размеры всех блоков равны между собой или отличаются не более чем на одну строку. В каждом из процессоров выполняется умножение одного блоков первой матрицы на вторую матрицу.

Эксперименты с этой программой показали, что при умножении матриц порядка 1024 в конечном поле коэффициент ускорения равен 77% при использовании 12 процессоров.

На рис. 4 показана зависимость скорости вычислений этой параллельной программы от количества процессоров, участвующих в вычислении.

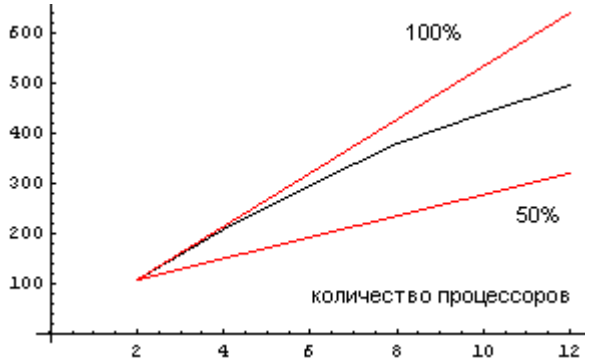

Рис. 4. Вычисление произведения матриц в конечном поле с помощью простой параллельной схемы умножения. Коэффициент ускорения равен 77%.

4.4. Рекурсивная блочная схема умножения матриц

Когда порядки квадратных матриц сомножителей равны 2^n, то матрицы разбиваются на четыре одинаковых квадратных блока и вычисление произведения матриц сводится к вычислению 8 умножений таких блоков, с последующим парным суммированием. Для вычисления произведения блоков снова используется эта же схема. Так можно продолжать вплоть до элементов.

Был реализован динамический вариант такой программы. Одно умножение матриц, которые разбиты на четыре блока, сводится к восьми блочным умножениям и может выполняться параллельно на 8 процессорах. Если при этом еще не все процессоры участвуют в вычислениях, то каждое такое блочное умножение может, в свою очередь, быть реализовано с помощью 8 умножений подблоков. Для управления свободными процессорами и организации блочных операций умножения на конкретных процессорах используется программа диспетчера.

Эксперименты для матриц 512 порядка с небольшими целыми коэффициентами ($\sim 2^{28}$) показали, что при умножении в кольце целых чисел

коэффициент ускорения равен 85%, а при умножении в конечном поле коэффициент ускорения равен 86.4%. Использовались 2 и 14 процессоров.

На рис. 5 и 6 показаны результаты двух экспериментов, проведенных для динамических параллельных программ умножения матриц. В этих программах использовался QT-формат хранения коэффициентов. В каждом эксперименте выполнялось умножение матриц 512 порядка. В первом случае коэффициенты матриц – это целые числа в пределах 2^{28}. Коэффициент ускорения при переходе с двух на 14 процессоров был равен 85%. Во втором случае коэффициенты матриц берутся из конечного поля, характеристика которого не превосходит $2 \cdot 10^8$. Коэффициент ускорения при переходе с двух на 14 процессоров составляет 85%.

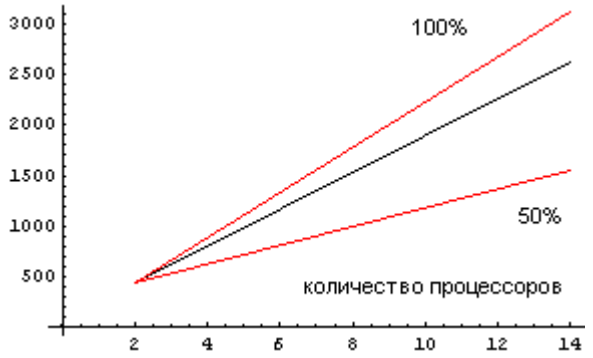

Рис. 5. Вычисление произведения целочисленных матриц в QT-формате с помощью рекурсивной схемы умножения. Коэффициент ускорения равен 85%.

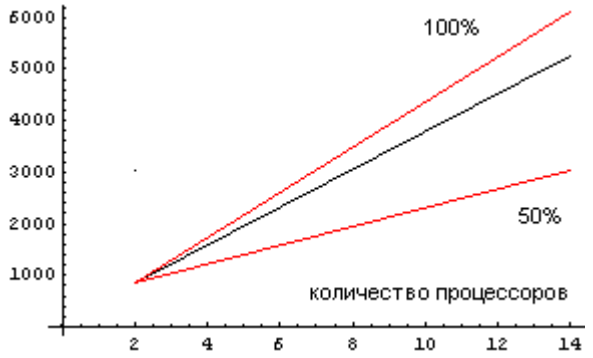

Рис. 6. Вычисление произведения матриц в конечном поле в QT-формате с помощью рекурсивной схемы умножения. Коэффициент ускорения равен 86%.

4.5. Рекурсивная блочная схема вычисления присоединенной и обратной матриц

Для QT-формата был реализован рекурсивный алгоритм вычисления присоединенной и обратной матриц, а также решения систем линейных уравнений (см. [13] и [14]).

Вычисления производились с плотной случайной целочисленной матрицей 128 порядка, с 28-битовыми целыми коэффициентами. Была составлена программа с помощью рекурсивного алгоритма [14] для QT-формата.

Этот алгоритм вычисления присоединенной матрицы в коммутативной области имеет такую же сложность, что и алгоритм матричного умножения, который в нем используется. Для построения параллельного алгоритма был применен последовательный алгоритм вычисления присоединенной матрицы, в котором все процедуры умножения выполнялись параллельно.

Результаты экспериментов приведены на рис 7. Коэффициент ускорения при переходе с двух на 14 процессоров равен 60%.

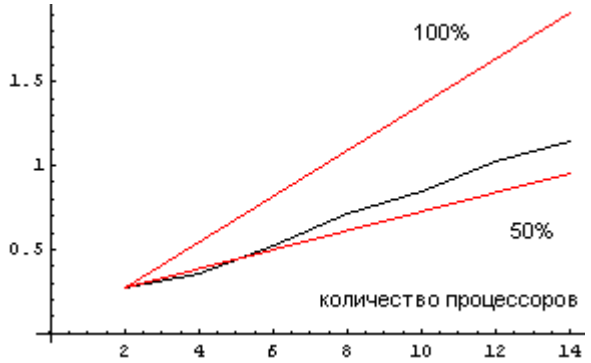

Рис. 7. Вычисление присоединенной (обратной) матрицы в случае целых коэффициентов в QT формате с помощью рекурсивного алгоритма. Коэффициент ускорения равен 60%.

5. Заключение

Результаты экспериментов демонстрируют достаточно высокую эффективность полученных программ компьютерной алгебры и закладывают основу для разработки аналогичных программ для других задач. Можно говорить, что сделаны первые шаги на пути создания параллельных алгоритмов компьютерной алгебры.

Литература

1. G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. // Proc. AFIPS Conference, Reston, VA, vol. 30, pp. 483-485, April 1967.

2. E. Walker. Extracting data flow information for parallelizing FORTRAN nested loop kernels. / Submission for the degree of Doctor of Philosophy. Department of Computer Science, Heslington, the University of York, England. 1994.
3. J. L. Gustafson. Reevaluating Amdahl's law. // Communications of the ACM, vol. 31, no. 5, pp. 532-533, May 1988.
4. Victor Ivannikov, Serguei Gaissaryan, Arutyun Avetisyan, Vartan Padaryan and Hennadiy Leontyev. Dynamic Analysis and Trace Simulation for Data Parallel Programs in the ParJava Environment. M. Estrada, A. Gelbukh (Eds.) // Avances en la Ciencia de la Computacion, ENC'04, Colima, Mexico, pp. 481-488.
5. M. Elnozahy and L. Alvisi and Y.~M. Wang and D.~B. Johnson. A survey of rollback-recovery protocols in message passing systems. / Technical report, Carnegie Mellon University, CMU-CS-96-181. October 1996.
6. W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. // Supercomputer, 12(1): 69--80, January 1996.
7. B. Carpenter, G. Zhang, G. Fox, Xiaoming Li, Xinying Li, and Y. Wen. Towards a Java environment for SPMD programming. // D. Pritchard and J. Reeve, (eds.), 4th Intern. Europar Conf., LNCS vol. 1470, 1998.
8. Виктор Иванников, Сергей Гайсарян, Арутюн Аветисян, Вартан Падарян. Применение среды ParJava для разработки параллельных программ. // Труды Института СистемногоПрограммирования. 2004. с. 41-62.
9. Victor Ivannikov, Serguei Gaissaryan, Arutyun Avetisyan, Vartan Padaryan. Improving properties of a parallel program in ParJava Environment // The 10th EuroPVM/MPI conference, Venice, Sept. 2003, LNCS v. 2840, pp 491-494.
10. Г.И.Малашонок, А.В.Красиков. Пакет процедур для работы с би-матрицами. Вестник ТГУ, Т.7, вып.1 VII Державинские чтения. 2002, с.76.
11. М.С.Зуев. Параллельные матричные алгоритмы в коммутативных областях. Дипломная работа. ИМФИ ТГУим.Г.Р.Державина, 2004.
12. Malaschonok G.I. Complexity Considerations in Computer Algebra. / Computer Algebra in Scientific Computing, CASC 2004/ Techn. Univ. Munchen, Garching, Germany, 2004. с. 325—332.
13. Malaschonok G.I. Recursive Method for the Solution of systems of Linear Equations. / Computational Mathematics. A. Sydow Ed. Proc. 15th IMACS World Congress. V.I. Berlin, August 1997. Wissenschaft and Technik Verlag, Berlin, 1997. pp. 475-480.
14. Malaschonok G.I. Effective Matrix Methods in Commutative Domains. Formal Power Series and Algebraic Combinatorics. Springer, 2000. pp. 506-517.