Параллельная компьютерная алгебра: новая схема управления распараллеливанием матричных рекурсивных алгоритмов*

Г.И.Малашонок, А.А.Сидько

malaschonok@gmail.com, allochka.sidko@gmail.com

Аннотация. Посвящается исследованиям в области параллельной компьютерной алгебры, в частности распараллеливанию матричных рекурсивных алгоритмов на кластере с распределенной памятью. Предлагается новая схема динамического управления для матричных рекурсивных алгоритмов. Причиной появления этой технологии управления параллельными вычислениями было открытие большого класса матричных рекурсивных алгоритмов в последние десятилетия. Все они, включая и алгоритмы В.Штрассена, являются результатом развития идей, заложенных в известных работах А.Карацубы о быстром умножении чисел и полиномов. Это уже третье поколение динамических схем распараллеливания, в котором учтены особенности, которые проявляются при вычислениях с разреженными матрицами на кластере с распределенной памятью. Подробно рассматриваются новые программные объекты, которые обеспечивают эффективную работу динамической схемы управления. Для иллюстрации приводятся два примера: алгоритм обращения треугольной матрицы и алгоритм умножения матриц.

Ключевые слова

параллельная компьютерная алгебра, распределенная память, матричные рекурсивные алгоритмы, динамическое распараллеливание, динамическое управление параллельными вычислениями

1 Введение

Параллельные алгоритмы в компьютерной алгебре стали предметом активных исследований с девяностых годов. Результаты исследований, полученные в ранних работах, были представлены в трудах конференций PASCO: Parallel Symbolic Computation (Linz, Austria, 1994 and Maui, U.S.A., 1997). Потом был 10-летний перерыв и новая серия конференций началась в 2007 году: PASCO'2007 (London, Canada) и PASCO'2010 (Grenoble, France). В 2010 году международая конференция по компьютерной алгебре была организована Тамбовским университетом [1]. Начиная с 2006 года специальная сессия по параллельной компьютерной алгебре присутствует на ежегодной конференции Applications of Computer Algebra: ACA'2006 (Varna, Bulgaria), ACA'2008 (Linz, Austria), ACA'2009 (Western Ontario, Canada), ACA'2010 (Vlora, Albania). Последняя из них недавно была в Иерусалиме – ACA'2017.

На этих конференциях обычно представлены следующие направления: параллельные полиномиальные вычисления, параллельные алгоритмы для символьной линейной алгебры, для матричных операций и решения линейных систем, параллельные методы решения систем дифференциальных уравнений, параллельные методы вычисления базисов Грёбнера, параллельные алгоритмы в комбинатонике и криптографии, параллельные алгоритмы в вычислительной алгебраической геометрии, сложность алгоритмов параллельной компьютерной алгебры и другие близкие направления.

Параллельные алгоритмы принято разделять на две большие группы: статические и динамические. В первые годы развивались преимущественно статические подходы к созданию параллельных программ. Одной из первых систем с динамическим параллелизмом была T-system разработанная в Институте программных систем РАН под руководством С.А. Абрамова. Она сегодня известна под названием OpenTS [2],

* Данная работа была частично поддержана грантом РФФИ 16-07-00420

[3]. Иной подход к созданию параллельных программ с динамическим управлением развивался в Тамбовском государственном университете. Этот подход изначально был ориентирован на распараллеливание матричных рекурсивных алгоритмов, которые были получены этим коллективом к тому времени.

Первой была предложена централизованная динамическая LLP-схема управления [4], в которой один из узлов кластера исполнял роль диспетчера всего вычислительного процесса.

Следующей была разработана DDP-схема децентрализованного управления [5]-[7]. В этой схеме на каждом процессорном узле создавался свой диспетчерный процесс. Однако, в этой схеме отсутствовал контроль глубины рекрсии и возможность переключения на новое задание пока не завершено текущее задание.

В настоящей работе предлагается новая схема динамического управления вычислительным процессом, в которой снимаются эти ограничения. Новая схема упраления носит название DAP-VAT-схемы (Drop-Amine-Pine-Vokzal-Aerodrome-Terminal). Она отличается тем, что последовательно разворачивает функции в глубину, сохраняя все состояния на любом уровне вложенности до тех пор, пока все вычисления в текущем вычислительном поддереве не будут завершены. Это дает возможность любому процессору свободно переключаться с одной подзадачи на другую, не дожидаясь завершения счета текущей подзадачи.

Как и в первых двух схемах выделяются три этапа вычислений. Первый этап – это разворачивание рекурсивных вычислений от корневой функции к листовым. Второй этап – это вычисления в листовых вершинах. И третий этап – это сворачивание вычислительного дерева, с возвращением результатов вычислений во внешние функции, которое заканчивается на корневой вершине.

2 Граф рекурсивного алгоритма

Граф рекурсивного вычислительного алгоритма в развернутом виде удобно представить себе в трехмерном пространстве внутри куба. Такой куб разрезан параллельными плоскостями, число которых равно количеству рекурсивных погружений, которые надо выполнить в ходе вычислительного процесса.

Все вершины с одинаковой глубиной рекурсии изображаются в одной плоскости. Все вершины первого уровня – на первой плоскости, следующего уровня рекурсии – на следующей параллельной плоскости и так далее. Движение данных в таком графе происходит от входной вершины первого уровня к другим вершинам первого уровня, а от них к вершинам второго уровня, и так далее до листовых вершин, а затем в обратном направлении. При движении результатов в обратном направлении, от дочерних вершин к родительским, вычисления в каждой вершине завершаются и память освобождается. Окончание вычислений происходит после получения результата в корневой вершине.

Пусть имеется вычислительный кластер, содержащий n процессоров, и требуется организовать вычислительный процесс в соответствии с некоторым рекурсивным алгоритмом. Каждый процессор может выполнить вычисления в любом компактном фрагменте графа алгоритма и, кроме того, может передать любую компактную часть другому процессору.

Главный принцип децентрализованного управления состоит в том, что разворачивание вычислительного процесса сопровождается передачей другим процессорам фрагментов вычислительного дерева вместе со списком подчиненных им свободных процессоров кластера. При этом каждый процессор может получить любую из вершин дерева алгоритма и управлять вычислительным процессом в этой вершине. В ходе вычислений свободные процессоры динамически перераспределяются, обеспечивая балансировку загрузки.

3 Примеры: умножение матриц и обращение треугольных матриц

Для иллюстрации мы приведем два простых рекурсивных алгоритма. Рассмотрим рекурсивный алгоритм обращения невырожденной нижней треугольной матрицы порядка2^N, $(N > 1)$. Пусть матрица разбита на равные квадратные блоки и требуется найти обратную к ней матрицу A^{-1}. Эти две матрицы связаны уравнением: $AA^{-1} = I$. Введем обозначение для блоков матриц:

$$A = \begin{pmatrix} a & 0 \\ c & d \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} x & 0 \\ z & k \end{pmatrix}$$

Из уравнения следует, что должны выполняться соотношения:

$$x = a^{-1}; k = d^{-1}; cx + dz = 0; \text{, Следовательно, } z = -kcx.$$

Операцию обращения треугольной матрица a будем обозначать $inv(a)$. Процедуры составления матрицы из блоков и разложения матрицы на блоки будем обозначать $(x, z, k) \to B$ и $A \to (a, c, d)$. Тогда один уроень рекурсивного алгоритма можно записать так:

$$A \to (a, c, d);\ x = inv(a);\ k = inv(d);\ w = c * x;\ w = k * w;\ z = -w;\ (x, z, k) \to B$$

Алгоритм рекурсивного блочного умножения матриц $AB = C$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} l & m \\ n & p \end{pmatrix} = \begin{pmatrix} w1 & w2 \\ w3 & w4 \end{pmatrix}$$

можно записать так:

$$A \to (a, b, c, d);\ B \to (l, m, n, p); u1 = a * l;\ v1 = b * n;\ w1 = u1 + v1;\ u2 = a * m;\ v2 = b * p;$$

$$w2 = u2 + v2;\ u3 = c * l;\ v3 = d * n;\ w3 = u3 + v3;\ u4 = c * m;\ v4 = d * p;\ w4 = u4 + v4;\ (w1, w2, w3, w4) \to C$$

На рисунке 1 показан граф алгоритма рекурсивного обращения треугольной матрицы (слева) и граф блочно-рекурсивного алгоритма матричного умножения (справа). Каждый граф имеет одну входную вершину и одну выходную вершину. На входную вершину приходит вектор входных данных inData, и в этой вершине вычисляется «входная функция» inF. В выходной вершине функция outF вычисляет вектор с результатом вычислений outData. В этих примерах входная функция выполняет разделение матриц на блоки, а выходная функция собирает из блоков результат вычислений. Все дуги в графах ориентированы по направлению передачи данных от входной вершины к выходной.

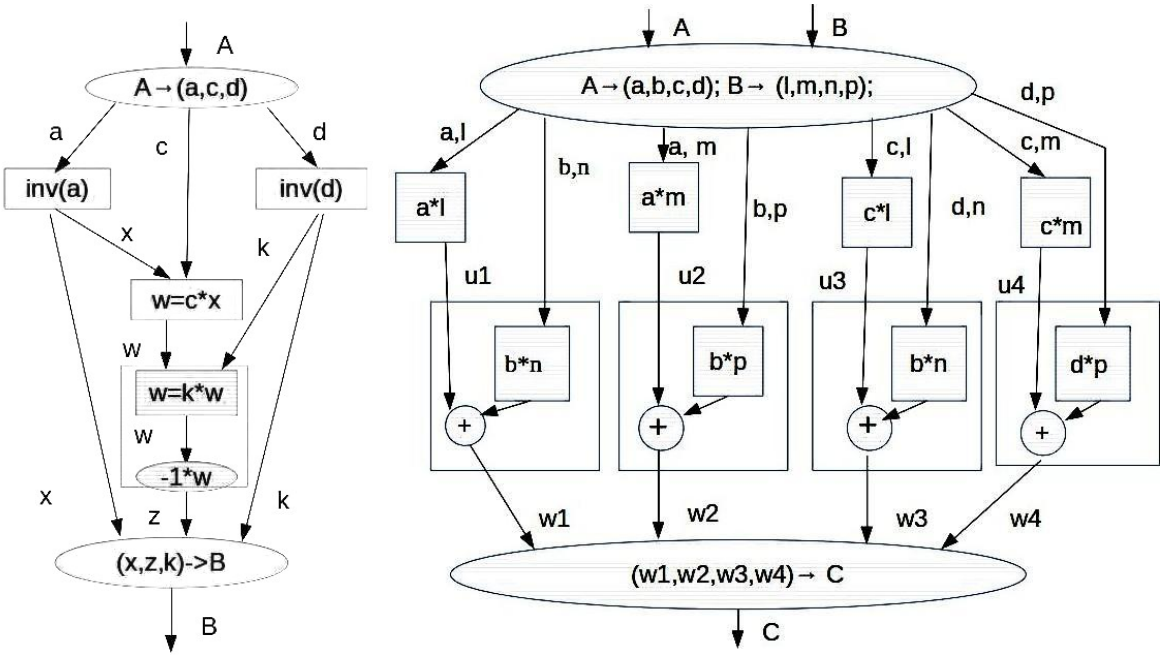

Рис. 1. Граф алгоритма рекурсивного обращения треугольной матрицы (слева) и граф блочно-рекурсивного алгоритма матричного умножения (справа).

4 Механизм управления вычислительным процессом

Рассмотрим составные части механизма управления вычислительным процессом.

4.1 Дроп

Разобьем вычислительный граф на отдельные компактные подграфы (дропы). Для этого выделим все вершины вычислительного графа, которые вычисляются рекурсивно (R-вершины). Такие вершины должны иметь высокую, как минимум нелинейную, зависимость сложности вычисления по отношению к объему данных. В каждый дроп-подграф входит одна R-вершина и, возможно, другие вершины вычислительного графа, в которые можно попасть из этой R-вершины, не проходя через остальные R-вершины. Мы рассматриваем только направленные пути, то есть пути по направлению передачи входных данных для функций. Разбиение вычислительного графа на дропы может быть неоднозначным,. При разбиении можно руководствоваться правилом не создавать лишние виды дропов, а иметь как можно меньшее число разных видов. Таким образом, дроп - это вычислительный подграф, который можно передать в другой процессор.

На рисунке 1 вершины, которые объединены в один дроп, обведены квадратным контуром. На рисунке 1 слева – это три вида дропов: inv, A*B и -A*B. На рисунке 1 справа – это два вида дропов: A*B и A*B+C.

Таким образом, мы определяем дропы, как наименьшие компоненты вычислительного графа, которые могут быть переданы в другие процессоры. Один из дропов соответствует всему вычислительному графу: для задачи обращения треугольных матриц – это inv, а для залачи умножения матриц – это A*B. Всего же необходимо иметь четыре разных вида дропов для полного покрытия этих двух вычислительных графов: inv, A*B, -A*B, A*B+C.

Если разворачивание вычисления рекурсивной функции осуществляется на одном и том же процессоре, то необходимо сохранять в памяти входные и выходные данные для всех уровней рекурсии.

4.2 Амин

Прежде чем дроп булет вычислен, необходимо развернуть соответствующий ему пограф. Такой подграф называется амин. Этот амин в свою очередь также состоит из дропов.

Например, амин A*B состоит из 4 дропов A*B, 4 дропов A*B+C, одной входной и одной выходной функции. амин inv состоит из 2 дропов inv, 1 дропа A*B, 1 дропа -A*B, одной входной и одной выходной функции.

4.3 Пайн

Все амины, которые формируются в одном процессоре сохраняются в общем списке, который носит название пайн. Этот список растет по мере формирования новых аминов. После завершения всех вычислений в амине и возврате результата, амин удаляется из памяти, но при этом все другие амины не меняют свои порядковые номера в данном пайне.

Это позволяет легко вести учет всех вычислительных задач при организации распределенных вычислений. Дропы могут вычисляться как в данном процессоре, так и в любом другом (дочернем) процессоре.Каждая вычислительная подзадача (дроп) имеет определенный адрес приписки (PAD), который определяется номером процессора (np), номером амина (na) в списке пайн этого процессора и номером дропа (nd) в этом амине.

$$PAD = (np, na, nd)$$

Это адрес по которому надо возвратить результат вычислений данного дропа. Вычисление дропа может осуществляться в любом процессоре. При этом будет построен соответствующий ему амин и размещен в некотором пайне. В адресном поле такого амина должен быть указан PAD для отправки результатов вычислений.

Коммуникацию с другими процессорами обслуживают «вокзал» «аєродром» и «терминал».

4.4 Вокзал

На вокзале располагаются все дроп-задания, которые ожидают своего направления на вычисления. Эти задания расположены на разных уровнях (levels). Эти уровни (вокзальные линии) соответствуют глубине рекурсии для дропов. Соответственно, с вокзала в первую очередь отправляются на выполнение дропы с самого низкого уровня. Это уровень называется текущим «уровнем вокала».

На вокзал прибывают новые дроп-задания всякий раз, когда при вычислении текущего амина обнаруживаются несколько дроп-заданий, которые могут быть вычисленны параллельно. Одно из них сразу направляется на вычисление, а остальные сохраняются на вокзале.

У каждого процессора имеется список подчиненных ему свободных процессоров, которые находятся в его распоряжении. Такой список может прийти от родительского процессора вместе с дроп-заданием. Кроме того, такой список может быть получен и от дочернего процессора, в специальном сообщении. Наконец, в него может попасть и сам текущий процессор, кога его вокзал окажется пуст и все дроп-задания выполнены.

Если еще нет дочерних процессоров, список свободных процессоров не пуст и вокзал не пуст, то число всех свободных процессоров делится на число дропов на самом нижнем уровне (уровне вокзала). В каждой такой группе процессоров первому процессору отправляется один дроп с нижнего уровня вокзала, а также ему направляются и все остальные процессоры его группы, как свободные процессоры.

4.5 Аэродром

Каждый процессор, который прислал дроп-задание называется родительским. Список всех родительских процессоров называется аэродромом.

В списке аэродрома, вместе с номером родительского процессора хранится и список pad-аресов заданий от этого родительского процессора. Этот список уменьшается, когда результаты дроп-заданий получены и отправляются родительскому процессору. Процессор исключается из списка аэродрома, когда будут вычислены все полученные от него дроп-задания.

4.6 Терминал

Терминал используется для связи с дочерними процессорами, которым были направлены дроп-заания. Все дочерние процессоры хранятся в терминале и располагаются там в соответствии со своим уровнем, аналогично вокзалу. Наименьший из уровней дочерних процессоров, которые имеются в терминале, называется «уровнем терминала».

В свою очередь «уровень процессора» — это наименьшее из двух чисел: уровень вокзала и уровень терминала. Как только меняется уровень любого процессора, этот процессор отправляет сообщение всем своим родительским процессорам, которые имеются на аэродроме.

5 Основные поля и функции

Приведем основные поля и функции объектов, описанных в предыдущем разделе.

5.1 Основные поля дроп-объекта

type – тип дропа (уникальный номер в списке всех типов дропов).

InData и outData – это векторы для входных и выходных данных.

NumberOfMainComponents – число главных компонент в InData, если значения этих компонент известны, то можно начинать вычисления.

myAmine– амин данного дропа.

daughterProc– процессор на котором вычисляется данный дроп.

state – состояние вычислительного процесса в текущий момент времени: (1) не готовы входные данные для начала вычислений, (2) готовы входные данные, можно начинать вычиления, (3) идут вычисления на данном процессоре, (4) идут вычисления на другом процессоре, (5) вычисления завершены и outData записан.

5.2 Основные функции дроп-объекта

Функция isItLeaf() оценивает объем вычислений для входных данных и возвращает true, если вычисления не следует распараллеливать.

Функция calcLeafDrop() вычисляет данный дроп в последовательном режиме.

5.3 Основные поля амин-объекта

PAD=(np, na, nd) – адрес по которому надо возвратить результат вычислений данного дропа.

type, inData, outData, NumberOfMainComponents – те же, что и у дропа,

outFunctionInData – вектор для входных данных выходной функции амина,

level - глубина рекурсии этого амина,

aminState – состояние вычислений: (0) вычисления не начинались, (1) идут вычисления, (2) – вычисления завершены.

arcs – топология графа амина: в i-той строке записаны связи для каждой компоненты выходного вектора i-того дропа (номер компоненты выходного вектора, номер дропа и номер компоненты входного вектора в этом дропе). Число строк равно числу дропов в амине плюс еще нулевая строка для выходного вектора из входной функции амина.

drop – массив всех дропов данного амина.

5.4 Основные функции амин-объекта

Функция inputFunction() принимает входные данные амина inData и формирует входные данные для дропов.

Функция outputFunction() принимает данные от дропов и формирует выходные данные амина на векторе outData.

5.5 Основные поля пайн-объекта

body – .список всех аминов, которые вычисляются в этом процессоре.

pineState – состояние вычислений: 0 – не идут вычисления, 1 - идут вычисления, 2 – вычисления завершены.

6 Организация вычислительных потоков

Мы используем два потока: вычислительный поток и диспетчерный поток. Эти потоки будут запущены на каждом процессоре кластера. Счетный поток отвечает за выполнение вычислений в соответствии с вычислительным алгоритмом. Диспетчерный поток отвечает за все функции управления вычислениями, включая прием всех приходящих сообщений.

6.1 Счетный поток CalcThread

Счетный поток ожидает приход первого дроп-задания на вокзал и запускает соответствующие вычисления.

6.1.1 Объекты счетного потока:

pine — список аминов на данном процессоре.

vokzal – массив из списков доступных дроп-задач.

aerodrome — список родительских процессоров.

terminal — массив из списков дочерних процессоров.

stopCalcThread – флаг остановки счетного потока.

isCalc – флаг, сообщающий, что идет процесс вычисления.

isNotEmptyVokzal – флаг, сообщающий, что список доступных дроп-задач не пуст.

currentDrop — содержит данные о текущем дропе: PAD, type, inData.

IsReadyDataOutF – флаг готовности данных для выходной функции амина.

6.1.2 Функции счетного потока:

stopCalcThread() – устанавливает флаг для остановки счетного потока.

isCalc() – сообщает, идет ли процесс вычисления.

setAmine()– принимает дроп-задание, создает амин и устанавливает флаг isCalc.

writeResultsToAmine() – результаты вычисления дропа записываются в его амин в векторы входных данных других дропов. Эти дропы проверяются на готовноть к исполнению и, в случае готовности, заносятся на Vokzal. Устанавливается флаг isReadyDataOutF.

runCalcThread() — основной метод счетного потока. Он ожидает пока Vokzal пуст. После прихода дроп-задания на Vokzal выбирается с Vokzal дроп с наименьшим уровнем и выполняется функция дропа isItLeaf(). Если эта функция вернула true, то culcLeafDrop() вычисляет данный дроп в последовательном режиме. Иначе выполняется функция setAmine() и создается amine в списке pine. Запускается вычисление входной функции, затем все доступные дропы записываются на Vokzal.

Поле завершения culcLeafDrop() вполняется функция writeResultsToAmine(). Она записывает результат в следующие дропы амина. Все вновь доступные дропы записываются на Vokzal и если флаг isReadyDataOutF равен true, то происходит завершение вычисления данного амина, а результат отправляется по pad-адресу данного амина. Если это адрес в другом процессоре, то выполняется команда межпроцессорной пересылки. Если это адрес в своем пайне, то writeResultsToAmine() записывает результат в следующие дропы амина по pad-адресу и все доступные дропы записываются в на Vokzal.

Если Vkzal пуст, то устанавливается флаг stopCalcThread, он снимается счетным потоком после получения дроп-задания с Vokzal.

6.2 Диспетчерський поток DispThread

Диспетчерский поток отвечает за обслуживание приема и передачи сообщений другим процессорам.

Он реализован как цикл и ожидает пока не будет сообщения о завершении всей программы или любого другого сообщения. Диспетчерский поток имеет высший приоритет и он периодически принудительно засыпает на некоторое время, чтобы в работу вошел счетной поток. Затем просыпается, прерывает счетный поток и просматривает все порты, занимается рассылкой сообщений. Каждое сообщение, посланное или отправлено другим процессорам, имеет свое значение tag, согласно которому выполняется то или иное действие:

– 0: сообщение содержит дроп-задание.

– 1: сообщение содержит свободные процессоры.

– 2: сообщение содержит состояние дочернего процессора.

– 3: сообщение содержит результат выполненной дроп-задачи.

– 4: сообщение содержит команду на завершение (конец всех вычислений).

Работу диспетчерского потока можно условно разбить на 8 процессов:

– Ожидание сигнала завершения.

– Прием задачи.

– Прием свободных процессоров.

– Прием и запись состояния дочернего процессора.

– Прием результата вычисленного дропа и запись этих результатов в соответствующий амин.

– Прием неглавных компонент и их дозапись в нужное место.

– Отправка доступных задач свободным процессорам (если есть задачи и процессоры).

– Отправка свободных процессоров дочерним (если нет доступных дроп-задач, но есть свободные и дочерние процессоры).

– Отправка всего списка свободных процессоров родительскому процессору (если пуст Вокзал и Терминал не содержит дочерних процессоров с положительными уровнями).

7 Заключение

Мы дали описание универсальной динамической схемы распараллеливания рекурсивных алгоритмов на кластере с распределенной памятью «DAP-VAT», привели описание основных объектов, их полей и функций, а также объяснили работу двух-поточной системы, которая запускается на каждом ядре кластера. Мы привели описание шести новых объектов, которые обеспечивают такой механизм управления и дают название этой схеме: drop, amine, pine, vokzal, aerodrome, terminal. Такую схему можно применять для любых матричных рекурсивных алгоритмов, как с плотными так и с разреженными матрицами. Схема была реализована на языке программирования Java с использованием пакетов OpenMPI и MathPartner[8], а ее работа была проверили на приведенных выше алгоритмах умножения и обращения матриц. Мы планируем провести подробное экспериментальное исследование эффективности этой схемы на других рекурсивных матричных алгоритмах и опубликовать результаты экспериментов.

Список литературы

[1] http://parca2010.parallel-computer-algebra.org/articles.pdf

[2] Abramov S., Adamovitch A., Kovalenko M. T-system: programming environment providing automatic dynamic parallelizing on IP-network of Unix-computers.*Report on 4-th International Russian-Indian seminar and exibition.* Sept. 15-25. Moscow, 1997.

[3] Abramov S., Adamovich A., Nyukhin A., Moskovsky A., Roganov V., Shevchuk E., Shevchuk Yu., Vodomerov A. OpenTS: An Outline of Dynamic Parallelization Approach.*PaCT 2005. Parallel Computing Technologies*, LNCS **3606**, Springer, Heidenberg, 303-312, 2005.

[4] Малашонок Г.И., Валеев Ю.Д. Организация параллельных вычислений в рекурсивных символьно-численных алгоритмах.*Труды конференции ПаВТ'2008* (Санкт-Петербург). Челябинск: Изд-во ЮУрГУ, 153-165, 2008.

[5] Ильченко Е.А. Об одном алгоритме управления параллельными вычислениями с децентрализованным управлением. *Вестник Тамбовского университета. Сер. Естественные и техничекие науки,* **18**, Вып. 4, 1198-1206, 2013.

[6] Ильченко Е.А. Об эффективном методе распараллеливания блочных рекуривных алгоритмов. *Вестник Тамбовского университета. Сер. Естественные и техничекие науки*, **20**, Вып. 5, 1173-1186, 2015.

[7] Малашонок Г.И. Управление параллельным вычислительным процессом. *Вестник Тамбовского университета. Сер. Естественные и техничекие науки*, **14**, Вып. 1, 269-274, 2009.

[8] Malaschonok G.I., MathPartner Computer Algebra, *Programming and Computer Software*, **43**, No. 2, 112–118, 2017.