Иванов Дмитрий Владимирович, Самарский государственный университет путей сообщения, г. Самара, Российская Федерация, кандидат физико-математических наук, доцент кафедры мехатроники в автоматизированных производствах, e-mail: dvi85@list.ru

Ivanov Dmitrii Vladimirovich, Samara State University of Transport Communications, Samara, the Russian Federation, Candidate of Physics and Mathematics, Associate Professor of the Mechatronics in Automated Production Department, e-mail: dvi85@list.ru

Ширинов Ильдар Раджабович, Самарский государственный университет путей сообщения, г. Самара, Российская Федерация, аспирант кафедры мехатроники в автоматизированных производствах, e-mail: shirinov89@mail.ru

Shirinov Ildar Radzhabovich, Samara State University of Transport Communications, Samara, the Russian Federation, Post-graduate Student of the Mechatronics in Automated Production Department, e-mail: shirinov89@mail.ru

УДК 519.6

ОБ ЭФФЕКТИВНОМ МЕТОДЕ РАСПАРАЛЛЕЛИВАНИЯ БЛОЧНЫХ РЕКУРСИВНЫХ АЛГОРИТМОВ

© **Е.А. Ильченко**

Ключевые слова: параллельный алгоритм; децентрализованное управление; расширенная присоединенная матрица; Adjoint; система Mathpar; SPMD вычислительная парадигма; MPI; многопоточность; hybrid parallel programming model; C++; GMP.

Дается описание общей схемы распараллеливания блочных рекурсивных алгоритмов, которая рассматривается на примере вычисления присоединенной матрицы, приводятся результаты экспериментов на вычислительном кластере «МВС-100К».

Рассмотрим пример блочного рекурсивного алгоритма. Даны две матрицы A, B, требуется найти их произведение. Разобьем матрицы A и B размером $n = 2^k$ на четыре равных блока:

$$\begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix} \times \begin{pmatrix} B_0 & B_1 \\ B_2 & B_3 \end{pmatrix} = \begin{pmatrix} C_0 & C_1 \\ C_2 & C_3 \end{pmatrix}.$$

Введем префиксное обозначение для функции матричного умножения $\mathfrak{m}(A, B) = C$, тогда получим:

$$\mathfrak{m}(A, B) = \begin{pmatrix} \mathfrak{m}(A_0, B_0) + \mathfrak{m}(A_1, B_2) & \mathfrak{m}(A_0, B_1) + \mathfrak{m}(A_1, B_3) \\ \mathfrak{m}(A_2, B_0) + \mathfrak{m}(A_3, B_2) & \mathfrak{m}(A_2, B_1) + \mathfrak{m}(A_3, B_3) \end{pmatrix} \quad (1)$$

Таким же образом можно поступить с каждым из восьми умножений в правой части равенства (1), рисунок 2 показывает параллельную реализацию этого алгоритма.

Рассмотрим блочный алгоритм обращения матрицы, имеющий сложность матричного умножения [1,2]. Если $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ является обратимой матрицей и A ее обратимый блок, то верно равенство:

$$M^{-1} = \begin{pmatrix} \boldsymbol{I} & -A^{-1}C \\ 0 & \boldsymbol{I} \end{pmatrix} \times \begin{pmatrix} \boldsymbol{I} & 0 \\ 0 & (D - BA^{-1}C)^{-1} \end{pmatrix} \times \begin{pmatrix} \boldsymbol{I} & 0 \\ -B & \boldsymbol{I} \end{pmatrix} \times \begin{pmatrix} A^{-1} & 0 \\ 0 & \boldsymbol{I} \end{pmatrix},$$

где $\boldsymbol{I}$ - единичная матрица. Раскрывая скобки, получим:

$$M^{-1} = \begin{pmatrix} A^{-1} + A^{-1}C\Big(D - BA^{-1}C\Big)^{-1}BA^{-1} & -A^{-1}C\Big(D - BA^{-1}C\Big)^{-1} \\ -\Big(D - BA^{-1}C\Big)^{-1}BA^{-1} & \Big(D - BA^{-1}C\Big)^{-1} \end{pmatrix}.$$

Обозначим обращение матрицы как функцию $\mathfrak{g}(M) = E$, тогда полученная формула примет вид:

$$\mathfrak{g}(M) = \begin{pmatrix} B_0 & B_1 \\ B_2 & B_3 \end{pmatrix},$$

где

$$B_0 = \mathfrak{g}(A) + \mathfrak{m}\Big(\mathfrak{m}\Big(\mathfrak{m}\big(\mathfrak{g}(A), \mathfrak{m}(C, B_3)\big), B\Big), \mathfrak{g}\big(A\big)\Big), B_1 = -\mathfrak{m}\big(\mathfrak{g}(A), \mathfrak{m}(C, B_3)\big),$$

$$B_2 = -\mathfrak{m}\Big(B_3, \mathfrak{m}\big(B, \mathfrak{g}(A)\big)\Big), B_3 = \mathfrak{g}\Big(D - \mathfrak{m}\big(\mathfrak{m}\big(B, \mathfrak{g}(A)\big), C\big)\Big).$$

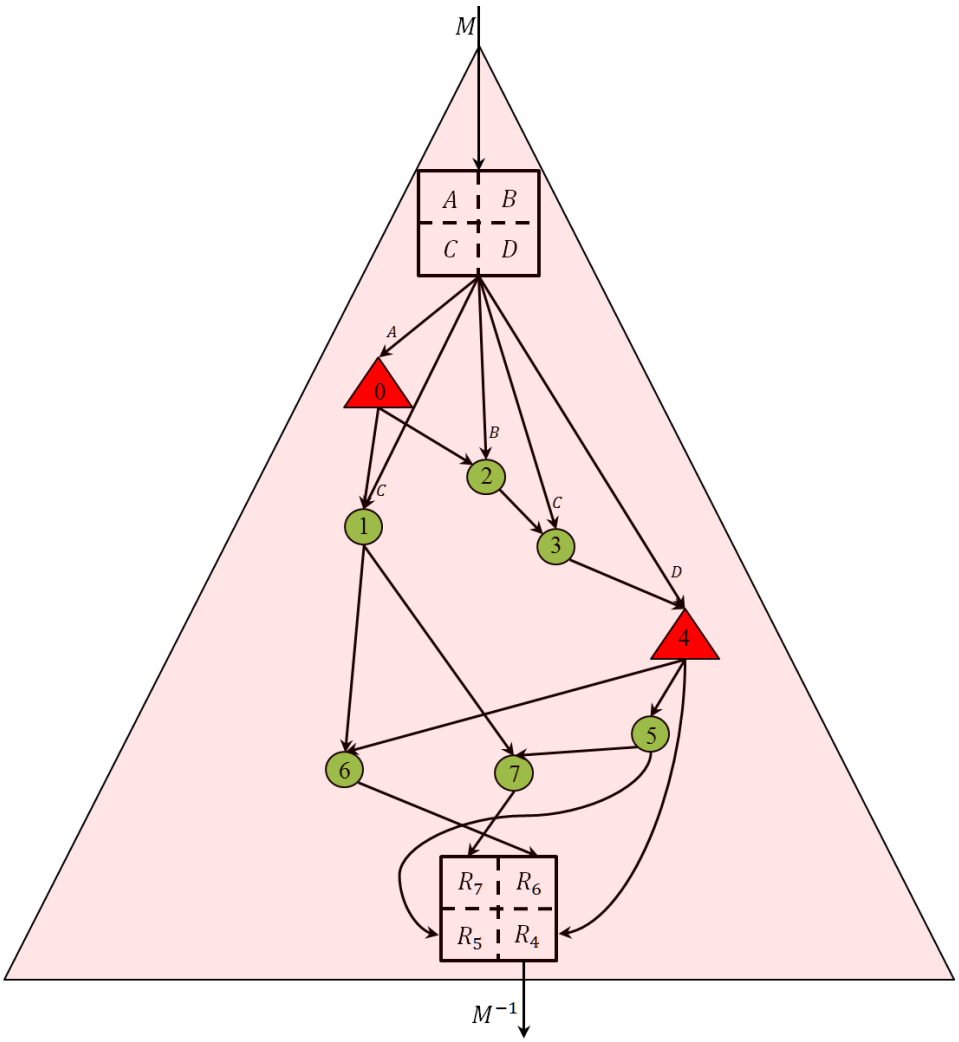

Рис. 1. Граф алгоритма для нахождения обратной матрицы

Номер типа T	Описание вершины	Входные данные I	Результат R	Форма вершины (на рисунке)	Цвет
0	$A \times B$, или $\mathfrak{m}(A, B)$	две матрицы $I = \{A, B\}$	матрица $R = \{C\}$	окружность	
1	A^{-1}, или $\mathfrak{g}(A)$	матрица $I = \{M\}$	матрица $R = \{M^{-1}\}$	треугольник	

Таблица 1. Описание типов вершин для алгоритма нахождения обратной матрицы

Как мы видим, в формуле для нахождения $\mathfrak{g}(M)$ в различной форме встречается суперпозиция функций $\mathfrak{m}$ и $\mathfrak{g}$. Рассмотрим один из случаев, например $\mathfrak{m}\big(B, \mathfrak{g}(A)\big)$, что означает матричное умножение $B \times A^{-1}$. Чтобы выполнить матричное умножение, сначала нужно вычислить A^{-1}. Эту и другие зависимости порядка вычислений одних функций от других можно представить в виде графа, где вершинами будут все функции, из которых состоит рекурсивный алгоритм, а связи будут задавать порядок вычислений вершин. К примеру, если между вершинами a и b есть дуга (a, b), то это означает, что вершина b должна быть посчитана строго после вершины a. Обозначим $-A^{-1} = V$, $V \times C = X$, $B \times V = Y$, $Y \times C = Q$, $(D+Q)^{-1} = Z$, $Z \times Y = W$, тогда формула для нахождения обратной матрицы будет выглядеть следующим образом:

$$M^{-1} = \begin{pmatrix} X \times W + A^{-1} & X \times Z \\ W & Z \end{pmatrix}. \tag{2}$$

Для алгоритма нахождения обратной матрицы мы имеем два типа вершин — функции $\mathfrak{m}$ и $\mathfrak{g}$.

Номер вершины	Номер типа	Входные данные	Постобработка	Описание вершины	$U(k)$	$W(k)$
0	1	$I_0 = A$	$R_0 = -R_0$	$V = -A^{-1}$	-	-
1	0	$I_1^A = R_0,\ I_1^B = C$	-	$X = V \times C$	0	-
2	0	$I_2^A = B,\ I_2^B = R_0$	-	$Y = B \times V$	0	-
3	0	$I_3^A = R_2,\ I_3^B = C$	-	$Q = Y \times C$	2	-
4	1	$I_4 = D + R_3$	-	$Z = (D + Q)^{-1}$	3	-
5	0	$I_5^A = R_4,\ I_5^B = R_2$	-	$W = Z \times Y$	4	-
6	0	$I_6^A = R_1,\ I_6^B = R_4$	-	$X \times Z$	1,4	-
7	0	$I_7^A = R_1,\ I_7^B = R_5$	$R_7 = R_7 - R_0$	$X \times W$	1,5	-

Таблица 2. Описание вершин для алгоритма нахождения обратной матрицы.

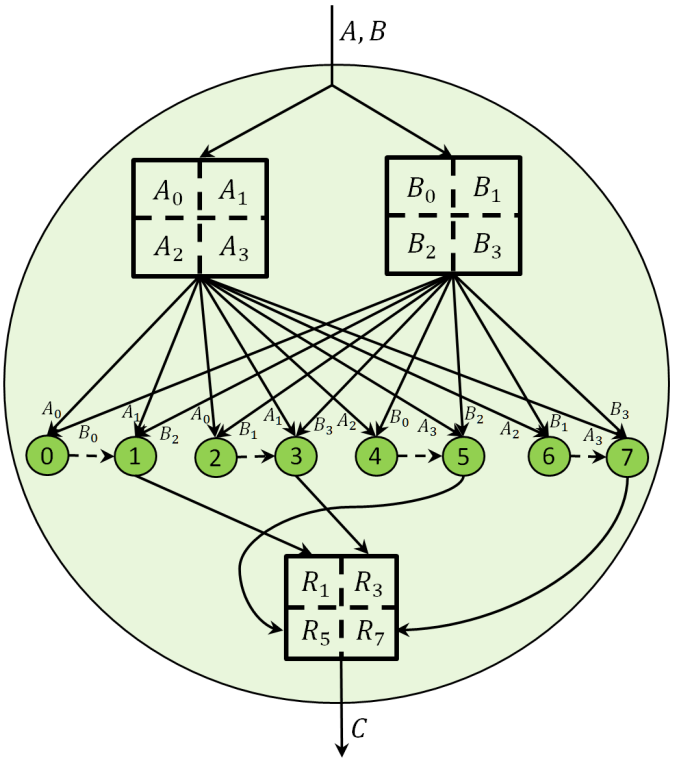

Рис. 2. Граф алгоритма для блочного матрчиного умножения

Номер вершины	Номер типа	Входные данные	Постобработка	Описание вершины	$U(k)$	$W(k)$
0	0	$I_0^A = A_0, I_0^B = B_0$	-	$A_0 \times B_0$	-	-
1	0	$I_0^A = A_1, I_0^B = B_2$	$R_1 = R_1 + R_0$	$A_1 \times B_2$	-	0
2	0	$I_0^A = A_0, I_0^B = B_1$	-	$A_0 \times B_1$	-	-
3	0	$I_0^A = A_1, I_0^B = B_3$	$R_3 = R_3 + R_2$	$A_1 \times B_3$	-	2
4	0	$I_0^A = A_2, I_0^B = B_0$	-	$A_2 \times B_0$	-	-
5	0	$I_0^A = A_3, I_0^B = B_2$	$R_5 = R_5 + R_4$	$A_3 \times B_2$	-	4
6	0	$I_0^A = A_2, I_0^B = B_1$	-	$A_2 \times B_1$	-	-
7	0	$I_0^A = A_3, I_0^B = B_3$	$R_7 = R_7 + R_6$	$A_3 \times B_3$	-	6

Таблица 3. Описание типов вершин для алгоритма блочного рекурсивного матричного умножения

Отсюда и далее вершину графа алгоритма со всеми присущими ей особенностями мы будем называть *задачей*. Если более формально, то *задача* — это множество входных данных A для алгоритма, множество выходных данных B, алгоритм F, отображающий $A \to B$. С задачей всегда будет связан некоторый граф G, определяющий подзадачи алгоритма и порядок их вычислений. Еще одно важное свойство любой задачи заключается в том, что

она *может* быть посчитана параллельно. Если же некоторую последовательность операций алгоритма нет смысла выполнять параллельно, мы не будем выделять эти действия в отдельную вершину или создавать для них отдельную задачу, а вынесем эти действия в предварительную или постобработку данных для некоторой вершины. Рассмотрим вершину с номером 0 для алгоритма нахождения обратной матрицы. Входные данные для задачи — матрица A, выходными будет $\mathfrak{g}(A) = A^{-1}$. Но поскольку нам нужна матрица $-A^{-1}$, а не A^{-1}, то после того, как эта вершина будет посчитана, необходимо выполнить некоторые преобразования матрицы A^{-1}. Действия, заключающиеся в некотором преобразовании выходных данных вершины, мы будем называть *постобработкой*. Действия, заключающиеся в инициализации входных данных вершины, мы будем называть *инициализацией вершины*. Инициализация происходит в тот момент, как только вершина становится *доступной* — т. е. тогда, когда все зависимые для нее вершины посчитаны. Постобработка происходит в тот момент, как только получены выходные данные для этой задачи и данные, необходимые для постобработки этой вершины. Входные данные для вершины с номером k будем обозначать как I_k, результат будем обозначать как R_k (под результатом понимаются данные, полученные по завершению постобработки вершины). Если входные данные вершины некоторого типа состоят более чем из одного элемента, верхним индексом будем обозначать какой-то конкретный элемент, например I_5^A. Если результат некоторой вершины состоит более чем из одного элемента, тогда верхним индексом будем обозначать конкретный элемент результата, например R_8^d. Номер типа для вершины с номером k будем обозначать как $T(k)$, множество зависимых вершин для нее будем обозначать как $U(k)$. Множество зависимых вершин, необходимых для постобработки, будем обозначать $W(k)$. Нужно заметить, что множества зависимых вершин для инициализации и постобработки в общем случае не будут совпадать, но в $W(k)$ мы не будем указывать вершины, которые уже присутствуют в $U(k)$. Тогда, принимая во внимание описание вершин из таблицы 2, уравнение (2) можно записать в виде:

$$M^{-1} = \begin{pmatrix} R_7 & R_6 \\ R_5 & R_4 \end{pmatrix}.$$

При определении инициализации и постобработки для некоторой вершины будем соблюдать следующее правило: если для выходных данных некоторой вершины k необходимо некоторое преобразование, причем оно одинаково для всех вершин, которые будут зависеть от k, тогда это преобразование $F(R_k)$ будем всегда выносить в постобработку этой вершины (k). Пример одной из наиболее оптимальных разбивок на вершины алгоритма блочного рекурсивного умножения приводится в таблице 3. Зависимые вершины для инициализации $\big(U(k)\big)$ на рисунках будем обозначать стрелками, необходимые вершины для постобработки $\big(W(k)\big)$ будем обозначать пунктирными стрелками. Граф блочного матричного умножения, соответствующий разбиению на вершины из таблицы 2, приводится на рис. 2.

Классической схемой для параллельной реализации подобных алгоритмов является схема с диспетчером (Manager-Workers method). Роль диспетчера выполняет один из узлов кластера, на остальных разворачивается дерево рекурсивного алгоритма (например, в случае блочного матричного умножения в определенный момент времени каждому узлу кластера будет сопоставлено вычисление некоторого матричного произведения $A_x \times B_y$). Изначально все процессоры (узлы кластера) являются свободными и находятся в соответствующем списке, который хранит диспетчер. К нему будет поступать 2 вида запросов:

– предоставить некоторому узлу M свободных процессоров, при этом он исключает их из списка свободных процессоров;

– добавить освободившееся процессоры в имеющийся список свободных.

Однако этот метод имеет существенные недостатоки. Рассмотрим ситуацию, когда у диспетчера закончились свободные процессоры. Пусть на некотором узле мы считаем за-

дачу, которая разбивается на 4 подзадачи, вычисляющиеся параллельно. От этого узла поступает запрос к диспетчеру, тот сообщает, что свободных процессоров нет, затем этот узел начнет считать полученные подзадачи по очереди. Проблема в том, что даже если у диспетчера появились свободные процессоры, запрос на них не придет от некоторого узла, пока тот считает какую-то задачу, и те подзадачи, для которых не оказалось свободных процессоров, вынуждены ждать соответствующего момента. Для решения этой проблемы мы могли бы организовать у диспетчера очередь подзадач — некоторый узел разбивает свою задачу, начинает считать одну из низ сам, а остальные отдает диспетчеру, который сам примет решение когда их начать считать. Но в этом случае диспетчер будет тормозить работу всей системы, если количество узлов достигнет некоторого N, диспетчер попросту не будет успевать обрабатывать все поступающие запросы.

Другой способ динамической реализации алгоритма - создание диспетчера на каждом узле кластера. Такое управление вычислительным процессом будет *децентрализованным*. Это можно сделать в многопоточном режиме — один поток будет диспетчером, остальные потоки будут использоваться для вычисления подзадач. Пусть на узле кластера имеется 8 ядер, тогда наиболее рационально будет использовать одно ядро для диспетчера, остальные семь ядер для счетных потоков. Таким образом, если на узле кластера N ядер, мы будем создавать N-1 счетных потоков для наиболее полного использования вычислительных ресурсов. Далее речь пойдет именно о таком способе организации параллельных вычислений. При этом мы продолжаем исследования, начатые в работах [4–12], посвященных динамическим алгоритмам управления параллельным вычислительным процессов. И непосредственно опираемся на работы [4] и [8], также посвященные данной тематике исследований.

Рассмотрим работу диспетчера более подробно. Диспетчерский поток имеет список свободных процессоров, список дочерних процессоров и список задач, отправленных на дочерние узлы. Диспетчерский поток реализован в виде бесконечного цикла и имеет 3 режима:

0 — режим ожидания задачи. В этом режиме ожидается задача от любого другого процессора. Может прийти одно из двух сообщений: либо задача, которую необходимо решить и результат вернуть отправителю, либо сигнал к завершению. Сигнал к завершению отсылается корневым узлом, т. е. тем, на котором была запущена стартовая задача. Если получен сигнал к завершению, происходит выход из цикла. Иначе выполняются действия по приему данных задачи, затем она полностью передается счетному потоку и происходит переход в режим 1;

1 — основной режим. В этом режиме счетный поток решает имеющуюся задачу и производит обмен свободными процессорами. В этом режиме происходят следующие действия:

- получение свободных процессоров от родительского узла;
- опрос дочерних узлов о завершенности работы;
- прием данных от дочерних узлов, завершивших работу;
- создание новых дочерних узлов;
- отправка дочерним узлам имеющихся свободных процессоров;
- проверка того, что вся задача, полученная от родительского узла решена.

Опишем эти действия более подробно. На *шаге 1* проверяется, пришло ли сообщение от родительского узла со свободными процессорами. Если это произошло, то полученные номера процессоров добавляются в список свободных. На *шаге 2* опрашиваются все дочерние узлы. Если от какого-то узла пришел запрос на завершение, происходит отправка этому

дочернему узлу номера последнего отосланного свободного процессора (lastSendet). Дочерний узел не вернет результат до тех пор, пока в его списке свободных узлов не окажется процессор с номером lastSendet. Это нужно для предотвращения потери номеров свободных процессоров.

На *шаге 3* для каждого дочернего узла делается проверка получения от него сообщения со свободными процессорами. Если сообщение пришло, то полученные номера заносятся в список свободных. Поскольку отправка свободных вершин от дочернего узла к родительскому осуществляется только после того, как дочерний узел завершил решение задачи, сразу происходит прием результата. Задача, отправленная этому дочернему узлу, помечается как выполненная. Номер узла переносится из списка дочерних процессоров в список свободных процессоров.

На *четвертом шаге* создаются новые дочерние узлы. Как один из вариантов управления параллельным вычислительным процессом, диспетчер может иметь настраиваемый параметр, который определяет максимальное количество дочерних узлов (maxD). Он может, например, экспериментально подбираться для каждого алгоритма. Но это число не может быть меньше двух. Слишком большое значение этого параметра также может приводит к ухудшению работы системы. Если число имеющихся дочерних узлов меньше maxD, будут созданы новые. Обозначим общее количество имеющихся свободных процессоров через Nfree, число дочерних узлов — через Dnodes. Допустим, имеется возможность создать еще L дочерних узлов. L определяется как минимум из 3 величин: maxD-Dnodes, Nfree и количества задач, имеющихся в очереди. Затем произвольным образом из списка свободных процессоров берется L номеров, отсылается им ровно по одной задаче и эти процессоры помещаются в список дочерних узлов.

На *пятом шаге* все имеющиеся свободные процессоры делятся на Dnodes групп. Затем каждая группа отсылается соответствующему дочернему узлу. Это означает, что мы всегда будем отдавать номера свободных процессоров поровну для всех дочерних узлов (ну или если количество свободных процессоров не кратно числу дочерних, какие-то из дочерних получат на единицу меньше). Номера свободных процессоров отсылаются только тогда, когда мы не можем создать из них новые дочерние узлы (т. е. когда число дочерних узлов достигло заранее заданного максимума MaxD).

Также имеющиеся подзадачи могут быть отданы локальным счетным потокам. Это делается только в том случае, если количество дочерних узлов достигло MaxD, либо отсутствуют номера свободных процессоров, чтобы мы могли создать новые дочерние узлы. Таким образом, если стоит вопрос, кому отдавать подзадачу на счет — локальным потокам или дочерним узлам, приоритет будет на дочерних узлах.

Далее проверяется тот факт, что вся имеющаяся задача решена. Если задача решена на процессоре с номером 0, то всем узлам отсылается сигнал к завершению работы. Если же задача решена на другом узле, родительскому узлу отправляется запрос на завершение и происходит переход в режим 2;

2 — режим завершения работы узла. В этом режиме происходят следующие действия. Если пришло сообщение со свободными узлами от родительского узла, добавим их в список свободных узлов. Если был получен номер lastSendet, делается проверка того факта, что в списке свободных процессоров он присутствует. Если он действительно присутствует, то родительскому узлу отсылаются все имеющиеся свободные узлы, следом отправляется полученный результат вычислений. Если число lastSendet не было получено, проверяется, пришло ли сообщение с ним.

Для тестирования производительности этого метода был выбран алгоритм для нахождения расширенной присоединенной матрицы [3].

Пусть дана матрица A и некоторое число d_0 , тогда матрицы A, S, E и число d назовем

расширенным присоединенным отображением для матрицы M:

$$A_{ext}(M, d_0) = (A, S, E, d).$$

M имеет размер $n \times n$, где $n = 2^k$. Если $M = 0 : A_{ext}(0, d_0) = (d_0\boldsymbol{I}, 0, 0, d_0)$, где $\boldsymbol{I}$ — единичная матрица. Если $k = 0$ и $M = a \neq 0$: $A_{ext}(a, d_0) = (d_0, a, 1, a)$. Если $k > 0$ и $M \neq 0$ разобьем матрицу M на четыре равных блока $M = (M_{ij}), i, j \in 1, 2$:

$$M = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}.$$

Введем обозначения:

$$I_{ij} = E_{ij}E_{ij}^T, \bar{I}_{ij} = \boldsymbol{I} - I_{ij}, Y_{ij} = E_{ij}^T S_{ij} - d_{ij}\boldsymbol{I}, \quad i, j \in 1, 2.$$

Пусть

$$A_{ext}(M_{11}, d_0) = (A_{11}, S_{11}, E_{11}, d_{11}).$$

Обозначим

$$M_{12}^1 = \frac{A_{11}M_{12}}{d_0}, M_{21}^1 = -\frac{M_{21}Y_{11}}{d_0}, M_{22}^1 = \frac{M_{22}d_{11} - M_{21}E_{11}^T M_{12}^1}{d_0}.$$

Пусть

$$A_{ext}(\bar{I}_{11}M_{12}^1, d_{11}) = (A_{12}, S_{12}, E_{12}, d_{12}),$$
$$A_{ext}(M_{21}^1, d_{11}) = (A_{21}, S_{21}, E_{21}, d_{21}).$$

Обозначим

$$M_{22}^2 = -\frac{A_{21}M_{22}^1 Y_{12}}{(d_{11})^2}, \ d_s = \frac{d_{21}d_{12}}{d_{11}}.$$

Пусть

$$A_{ext}(\bar{I}_{21}M_{22}^2, d_s) = (A_{22}, S_{22}, E_{22}, d_{22}).$$

Обозначим

$$M_{11}^2 = -\frac{S_{11}Y_{21}}{d_{11}}, \ M_{12}^2 = \frac{\left(\frac{\frac{S_{11}E_{21}^T A_{21}}{d_{11}}M_{22}^1 - I_{11}M_{12}^1 d_{21}}{d_{11}}\right)Y_{12} + S_{12}d_{21}}{d_{11}},$$

$$M_{12}^3 = -\frac{M_{12}^2 Y_{22}}{d_s}, \ M_{22}^3 = S_{22} - \frac{I_{21}M_{22}^2 Y_{22}}{d_s}, \ A^1 = A_{12}A_{11}, \ A^2 = A_{22}A_{21},$$

$$L = \left(\frac{A^1 - \frac{I_{11}M_{12}^1 E_{12}^T A^1}{d_{11}}}{d_{11}}\right)d_{22}, \ P = \frac{A^2 - \frac{I_{21}M_{22}^2 E_{22}^T A^2}{d_s}}{d_{21}},$$

$$F = -\frac{\left(\frac{S_{11}E_{21}^T A_{21}}{d_{11}}\right)d_{22} + \frac{M_{12}^2 E_{22}^T A^2}{d_s}}{d_{21}}, \ G = -\frac{\left(\frac{M_{21}E_{11}^T A_{11}}{d_0}\right)d_{12} + \frac{M_{22}^1 E_{12}^T A^1}{d_{11}}}{d_{11}},$$

$$A = \begin{pmatrix} \frac{L+FG}{d_{12}} & F \\ \frac{PG}{d_{12}} & P \end{pmatrix}, \ S = \begin{pmatrix} \frac{M_{11}^2 d_{22}}{d_{21}} & M_{12}^3 \\ \frac{S_{21}d_{22}}{d_{21}} & M_{22}^3 \end{pmatrix}, \ E = \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{pmatrix},$$

$$A_{ext}(M, d_0) = (A, S, E, d_{22}).$$

Используемые в алгоритме матрицы E_{ij} имеют особое строение — в каждой строке может стоять только один ненулевой элемент, причем он всегда будет равен единице. Поэтому вместо явного хранения матрицы E_{ij} можно иметь 2 одномерных массива, в первом будем хранить номера строк, где находятся единичные элементы (E_i), во втором номера столбцов (E_j). Рассмотрим матричные умножения вида $A = E_{ij} \times M$: строки матрицы M с номерами $E_j[k]$ должны быть скопированы в матрицу A, и будут в ней иметь номера $E_i[k]$. В случае умножения вида $A = E_{ij}^T \times M$ нужно просто обменять массивы E_i и E_j ролями. Из алгоритма видно, что умножения на матрицу E_{ij} всегда можно выполнять только слева, поэтому случай $A = M \times E_{ij}$ рассматривать не будем. Умножения вида $A = I_{ij} \times M$ выполняются так: строки матрицы M с номерами $E_i[k]$ должны быть скопированы на строки с теми же номерами в матрицу A (нужно заметить, что матрица E_{ij} может иметь полностью нулевые строки, поэтому матрица A не будет точной копией матрицы M. Ввиду особого строения матриц E_{ij} и S_{ij}, нахождение $Y_{ij} = E_{ij}^T S_{ij} - d_{ij}\boldsymbol{I}$ можно также выполнять более быстрым образом. Матрицу $E_{ij}^T S_{ij}$ найдем способом, описанным выше. Далее необходимо построчно скопировать матрицу $E_{ij}^T S_{ij}$ в Y_{ij}, соблюдая следующие правила: если текущая строка матрицы $E_{ij}^T S_{ij}$ имеет одни нули, тогда в Y_{ij} эта строка также будет содержать нули кроме столбца с номером, равным номеру текущей строки — на этой позиции будет элемент $-d$. Если текущая строка имеет какие-то ненулевые элементы, то они должны быть просто скопированы в Y_{ij}, при этом необходимо отбросить элемент с номером столбца, равным номеру строки — таким образом, в Y_{ij} он будет равен нулю. Умножения вида $A = \bar{I}_{ij} \times M$ выполняются так: пусть E_i^{inv} будет дополнением для множества E_i (например, если текущий размер блоков матриц равняется 4, и E_i содержит элементы [2,0], тогда дополнением будет массив [1,3]). Далее необходимо выполнить рассмотренное выше умножение $A = I_{ij} \times M$, взяв в качестве I_{ij} массив E_i^{inv}.

Для реализации алгоритма могут быть использованы подзадачи следующих типов:

Номер типа T	Описание вершины	Входные данные I	Результат R	Форма вершины (на рисунке)	Цвет
0	$A \times B$	две матрицы $I = \{A, B\}$	матрица $R = \{C\}$	окружность	
1	$\frac{A \times B}{d}$	две матрицы и делитель $I = \{A, B, d\}$	матрица $R = \{C\}$	треугольник	
2	$A_{ext}(M, d)$	матрица и входной параметр $I = \{M, d\}$	три матрицы и выходной параметр $R = \{A, S, E, d\}$	квадрат	

Таблица 4. Описание типов вершин для алгоритма присоединенной обратной матрицы

Умножение $A \times B$ будем выполнять в блочном виде, разделив матрицы A и B на четыре части, как показано в таблице 3 и изображено на рисунке 2. Умножение $\frac{A \times B}{d}$ выполняется аналогично умножению $A \times B$, за тем исключением, что к результату применяется постобработка вида C/d. В таблице 5 приводится описание используемых при реализации вершин в соответствии с алгоритмом для нахождения расширенной присоединенной матрицы (поскольку вершина с номером типа 2 и есть сам алгоритм для вычисления расширенной присоединенной матрицы, это описание также относится и к ней). Матричные умножения, одним из сомножителей которых являются матрицы $I_{ij}, \bar{I}_{ij}, E_{ij}$ или E_{ij}^T не имеет смысла вычислять параллельно, поэтому такие умножения не будут выноситься в отдельную вершину. Матрица Y_{ij} также должна быть вычислена последовательно.

№	$T(k)$	Входные данные	Постобработка	Описание вершины	$U(k)$	$W(k)$
0	4	$I_0^M = M_{11}, I_0^d = d_0$	-	$A_{ext}^{11}(M_{11}, d_0)$	-	-
1	2	$I_1^A = R_0^A, I_1^B = M_{12}, I_1^d = d_0$	-	M_{12}^1	0	-
2	2	$I_2^A = M_{21}, I_2^B = Y_{11}, I_2^d = d_0$	$R_2 = -R_2$	M_{21}^1	0	-
3	0	$I_3^A = M_{21}, I_3^B = R_0^{ET} R_1$	$R_3 = \frac{M_{22} R_0^d - R_3}{d_0}$	M_{22}^1	1	-
4	4	$I_4^M = \bar{I}_{11} R_1, I_4^d = R_0^d$	-	$A_{ext}^{12}(\bar{I}_{11} M_{12}^1, d_{11})$	1	-
5	4	$I_5^M = R_2, I_5^d = R_0^d$	-	$A_{ext}^{21}(M_{21}^1, d_{11})$	2	-
6	0	$I_6^A = R_5^A, I_6^B = R_3$	-	$A_{21} M_{22}^1$	3, 5	-
7	2	$I_7^A = R_6, I_7^B = Y_{12}, I_7^d = (d_{11})^2$	$R_7 = -R_7$	M_{22}^2	4, 6	-
8	4	$I_8^M = \bar{I}_{21} R_7, I_8^d = d_s$	-	$A_{ext}^{22}(\bar{I}_{21} M_{22}^2, d_s)$	7	-
9	2	$I_9^A = R_0^S, I_9^B = Y_{21}, I_9^d = R_0^d$	$R_9 = -\frac{R_9 R_8^d}{R_5^d}$	$\frac{M_{11}^2 d_{22}}{d_{21}}$	5	8

Таблица 5. Описание вершин для алгоритма нахождения присоединенной матрицы

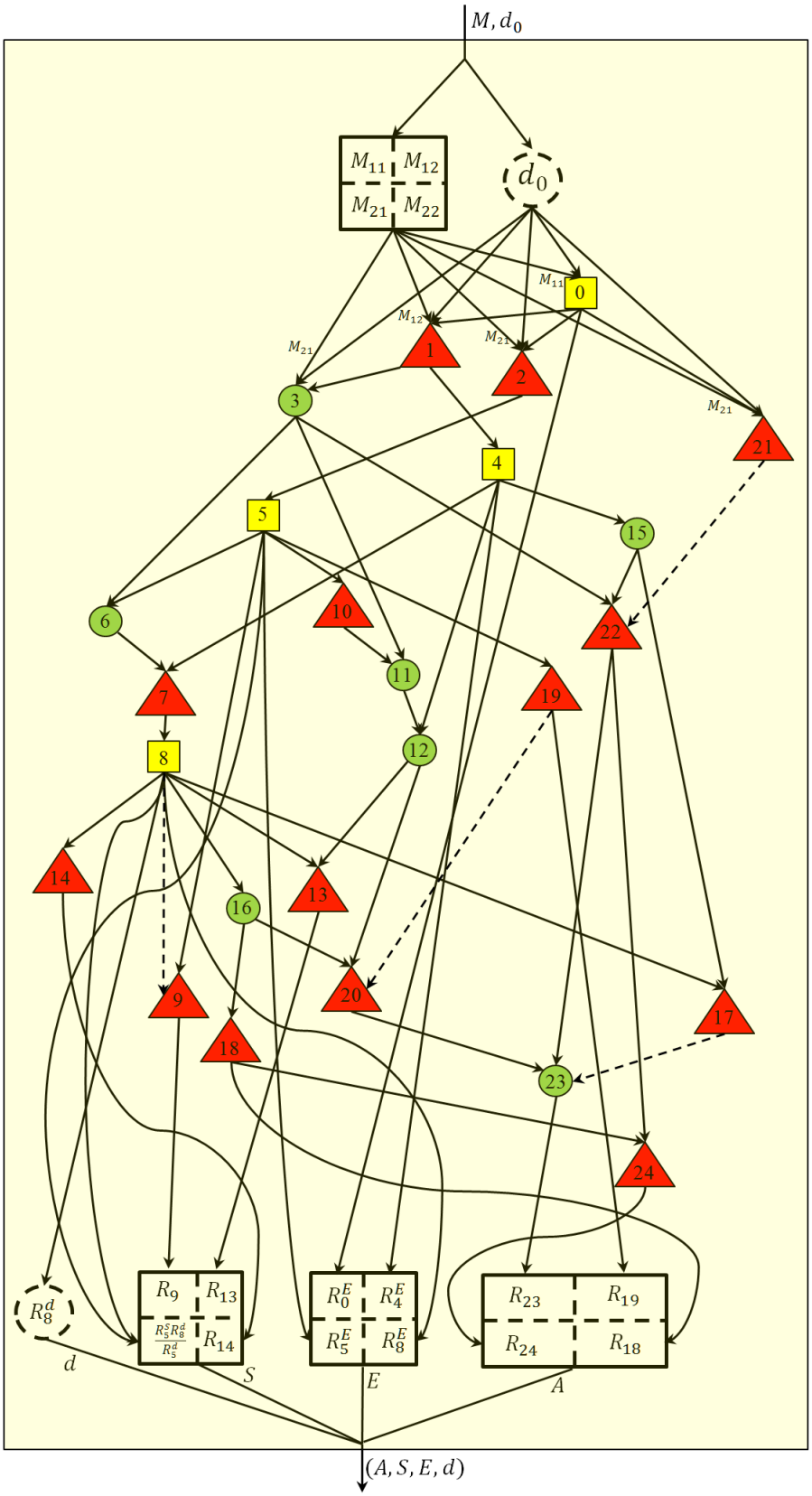

Рис. 3. Граф алгоритма вычисления расширенной присоединённой матрицы

№	$T(k)$	Входные данные	Постобработка	Описание вершины	$U(k)$	$W(k)$
10	2	$I_{10}^A = R_0^S, I_{10}^B = R_5^{E^T} R_5^A$ $I_{10}^d = R_0^d$	-	$\frac{(S_{11})\times(E_{21}^T A_{21})}{d_{11}}$	5	-
11	0	$I_{11}^A = R_{10}, I_{11}^B = R_3$	$R_{11} = \frac{R_{11} - I_{11} R_1 R_5^d}{R_0^d}$	$\frac{\frac{S_{11} E_{21}^T A_{21}}{d_{11}} M_{22}^1 - I_{11} M_{12}^1 d_{21}}{d_{11}}$	3, 10	-
12	0	$I_{12}^A = R_{11}$ $I_{12}^B = Y_{12}$	$R_{12} = \frac{R_{12} + R_4^S R_5^d}{R_0^d}$	M_{12}^2	4, 11	-
13	2	$I_{13}^A = R_{12}, I_{13}^B = Y_{22}$ $I_{13}^d = d_s$	$R_{13} = -R_{13}$	M_{12}^3	8, 12	-
14	2	$I_{14}^A = I_{21} R_7, I_{14}^B = Y_{22}$ $I_{14}^d = d_s$	$R_{14} = R_8 - R_{14}$	M_{22}^3	8	-
15	0	$I_{15}^A = R_4^A, I_{15}^B = R_0^A$	-	A^1	4	-
16	0	$I_{16}^A = R_8^A, I_{16}^B = R_5^A$	-	A^2	8	-
17	2	$I_{17}^A = I_{11} R_1, I_{17}^B = R_4^{E^T} R_{15},$ $I_{17}^d = R_0^d$	$R_{17} = \left(\frac{R_{15} - R_{17}}{R_0^d}\right) R_8^d$	L	8, 15	-
18	2	$I_{18}^A = I_{21} R_7, I_{18}^B = R_8^{E^T} R_{16},$ $I_{18}^d = d_s$	$R_{18} = \frac{R_{16} - R_{18}}{R_5^d}$	P	16	-
19	2	$I_{19}^A = R_0^S, I_{19}^B = R_5^{E^T} R_5^A$ $I_{19}^d = R_0^d$	-	$\frac{(S_{11})\times(E_{21}^T A_{21})}{d_{11}}$	5	-
20	2	$I_{20}^A = R_{12}, I_{20}^B = R_8^{E^T} R_{16}$ $I_{20}^d = d_s$	$R_{20} = -\frac{R_{19} R_8^d + R_{20}}{R_5^d}$	F	12, 16	19
21	2	$I_{21}^A = M_{21}, I_{21}^B = R_0^{E^T} R_0^A$ $I_{21}^d = d_0$	-	$\frac{(M_{21})\times(E_{11}^T A_{11})}{d_0}$	0	-
22	2	$I_{22}^A = R_3, I_{22}^B = R_4^{E^T} R_{15}$ $I_{22}^d = R_0^d$	$R_{22} = -\frac{R_{21} R_4^d + R_{22}}{R_0^d}$	G	3, 15	21
23	0	$I_{23}^A = R_{20}, I_{23}^B = R_{22}$	$R_{23} = \frac{R_{17} + R_{23}}{R_4^d}$	$\frac{L+FG}{d_{12}}$	20,22	17
24	2	$I_{24}^A = R_{18}, I_{24}^d = R_4^d,$ $I_{24}^B = R_{21}$	-	$\frac{PG}{d_{12}}$	18,22	-

Таблица 5 (продолжение). Описание вершин для алгоритма нахождения присоединенной матрицы

Для исследований алгоритма была написана программа на C++. Для реализации многопоточности была выбрана библиотека Pthreads, для реализации арифметики многократной точности была взята библиотека GMP. Эксперименты проводились на кластере «МВС-100К». В качестве исходной матрицы генерировалась случайная матрица размером 512x512, не имеющая нулевых элементов. Сами элементы матрицы генерировались по модулю 8 (3 бита). На рисунках 2, 3, 4 приводятся зависимости времени вычислений от количества

ипользуемых узлов кластера при заданном количестве счетных потоков.

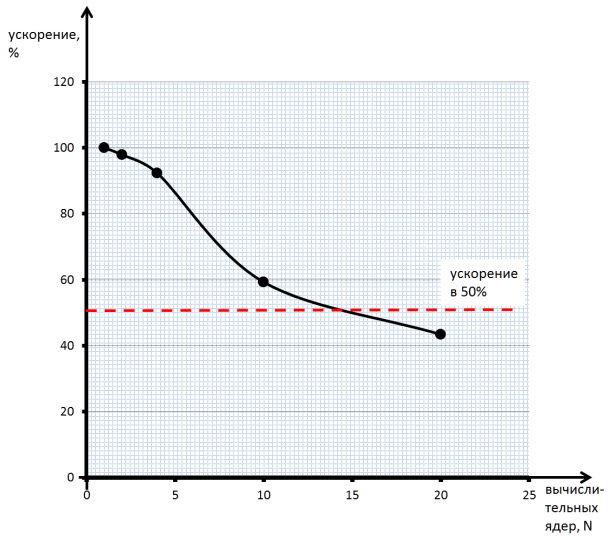

Рис. 4. График зависимости времени вычисления присоединенной матрицы от используемого количества узлов, количество счетных потоков равно одному

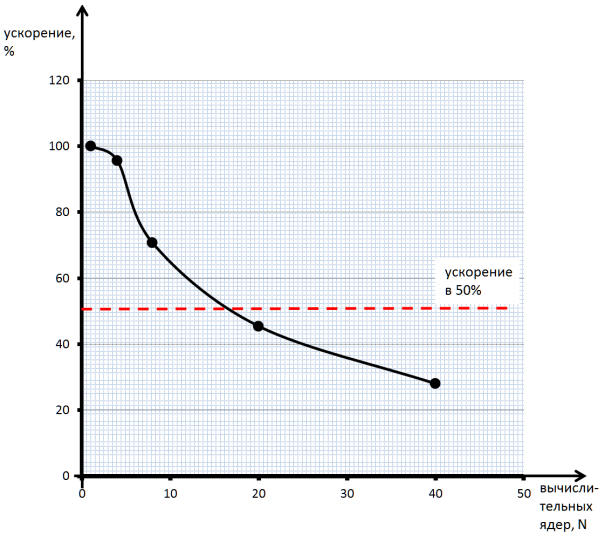

Рис. 5. График зависимости времени вычисления присоединенной матрицы от используемого количества узлов, количество счетных потоков равно двум

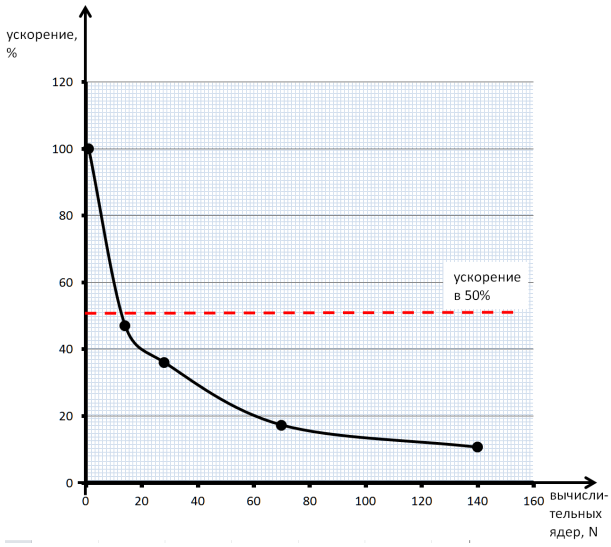

Рис. 6. График зависимости времени вычисления присоединенной матрицы от используемого количества узлов, количество счетных потоков равно семи

Заключение

Описанный подход является одним из возможных вариантов организации параллельных вычислений с децентрализованным управлением. Данная работа развивает идеи, изложенные в [3, 4]. Для исследования эффективности данного метода была написана программа на C++, в качестве реализуемой задачи был выбран алгоритм нахождения присоединенной матрицы, поскольку он требует больших объемов вычислений и характеризуется довольно сложной постобработкой вершин. Была проведена серия экспериментов для алгоритмов компьютерной алгебры на кластере МВС-100К Межведомственного суперкомпьютерного центра РАН. Дальнейшее развитие изложенной концепции заключается в более рациональ-

ном распределении заданий диспетчером для получения лучшего масштабирования программы.

ЛИТЕРАТУРА

1. *Strassen V.* Gaussian Elimination is not optimal. Numerische Mathematik. 1969. V. 13. P. 354-356.

2. *Малашонок Г.И.* Матричные методы вычислений в коммутативных кольцах. Тамбов: Изд-во Тамбовского университета, 2002.

3. *Malaschonok G.I.* О вычислении ядра оператора действующего в модуле // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2008. Т. 13. Вып. 1. С. 129-131.

4. *Malaschonok G., Ilchenko E.* Decentralized control of parallel computing // International conference Polynomial Computer Algebra. St. Petersburg, PDMI RAS, 2012. P. 57-58.

5. *Бетин А.А.* Эксперименты с параллельным алгоритмом вычисления присоединённой матрицы и параллельным умножением файловых матриц // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2010. Т. 15. Вып. 1. С. 341-345.

6. *Бетин А.А.* Эксперименты с параллельным алгоритмом вычисления присоединённой матрицы // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2010. Т. 15. Вып. 6. С. 1748-1754.

7. *Малашонок Г.И.* Компьютерная математика для вычислительной сети // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2010. Т. 15. Вып. 1. С. 322-327.

8. *Малашонок Г.И.* Управление параллельным вычислительным процессом // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2009. Т. 14. Вып. 1. С. 269-274.

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

10. *Г.И. Малашонок, Ю.Д. Валеев.* Рекурсивное распараллеливание символьно-численных алгоритмов // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2006. Т. 11. Вып. 4. С. 536-549.

11. *Малашонок Г.И., Валеев Ю.Д.* Динамическое распределение заданий в вычислительном кластере по графу алгоритма // 11 Державинские чтения. Тезисы докладов. Тамбов: Изд-во ТГУ им. Г.Р. Державина, 2006. С. 38-47.

12. *Г.И. Малашонок, Ю.Д. Валеев.* О некоторых подходах к построению параллельных программ // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2005. Т. 10. Вып. 1. С. 154-156.

13. *Malaschonok G.I.* Effective Matrix Methods in Commutative Domains // Formal Power Series and Algebraic Combinatorics. Berlin: Springer, 2000. P. 506-517.

БЛАГОДАРНОСТИ: Автор выражает благодарность своему руководителю Г.И. Малашоноку за постановку задачи. Работа выполнялась при частичной поддержке гранта РФФИ № 12-07-00755-а.

Поступила в редакцию 1 июня 2015 г.

Ilchenko E.A. ABOUT EFFECTIVE METHODS OF PARALLELIZING BLOCK RECURSIVE ALGORITHMS

We describe an algorithm for the decentralized control of parallel computing process which is based on the SPMD computational paradigm and present the results of experiments on a cluster «MVS-100K».

Key words: parallel algorithm; decentralized control; Adjugate matrix, the system Mathpar, SPMD computational paradigm, adjoint, MPI, multithreading, hybrid parallel programming model, C++, GMP.

Ильченко Евгений Александрович, Тамбовский государственный университет им. Г.Р. Державина, г. Тамбов, Российская Федерация, аспирант кафедры математического анализа, e-mail: ilchenkoea@gmail.com.

Ilchenko Evgeni Aleksnadrovich, Tambov State University named after G.R. Derzhavin, Tambov, the Russian Federation, Post-graduate Student of the Mathematical Analysis Department, e-mail: ilchenkoea@gmail.com