ПАРАЛЛЕЛЬНЫЙ ЛОГИЧЕСКИЙ ВЫВОД НА КЛАСТЕРНЫХ СИСТЕМАХ

В.П. Кутепов, М.М. Кумачев

Введение

Логика, логический вывод первого и высших порядков представляют собой самый высокий уровень формализации человеческих рассуждений, математики, различных предметных областей, с которыми человек сталкивается на практике при решении различных задач. Языки логического программирования (Prolog, LISP и другие) созданы для того, чтобы упростить работу как на стадии логического программирования, то есть описания логическими средствами решаемой задачи, так и для реализации на компьютере алгоритмов логического вывода. Известно, что сложность алгоритмов вывода в логике первого порядка растет экспоненциально в зависимости от сложности задачи. И время выполнения алгоритма может достигать таких величин, что пользователь, ожидающий результата, все время стоит перед неразрешимой проблемой: либо вывод отсутствует, либо он еще не получен. Тем не менее, создание эффективных алгоритмов логического вывода вместе с возможностью их реализации на современных компьютерных системах, насчитывающих сегодня десятки и сотни тысяч узлов (процессоров или компьютеров), заметно расширяет круг задач, успешно решаемых логическими средствами.

В статье описан алгоритм параллельного вывода в логике первого порядка и результаты исследования эффективности его реализации на кластере Московского Энергетического Института (Технического Университета), состоящего из 16 узлов, каждый из которых состоит из двух двухъядерных процессоров с быстродействием ядра $4x10^9$ опер./сек.

1. Алгоритм параллельного логического вывода и его реализации

Наиболее распространенным методом дедуктивного вывода на сегодняшний день является принцип резолюции, предложенный Дж. Робинсоном. Этот метод положен в основу предлагаемого алгоритма параллельного логического вывода.

Прежде чем преступить к описанию предлагаемого алгоритма, перечислим требования, которым он должен удовлетворять, и основные пути достижения этих требований.

1. В качестве аппаратной платформы рассматривается кластер, при этом каждый узел кластера может представлять собой многоядерную систему с общей памятью.
2. Предлагаемый алгоритм должен обладать свойством полноты. То есть любая пара дизъюнктов должна быть гарантированно прорезольвирована между собой за конечное время.
3. В реализации алгоритма должна быть заложена возможность регулирования загруженности вычислительных узлов кластера с целью более эффективного использования его ресурсов, а также минимизации количества обменов между вычислительными узлами.

Идея предлагаемого алгоритма параллельного вывода на кластере состоит в том, чтобы разделить исходное множество дизъюнктов между узлами кластера и на каждом узле реализовать алгоритм логического вывода ранее разработанный в [3] для многоядерных компьютеров. Взаимодействие узлов между собой должно быть организовано так, чтобы уменьшить число обменов между узлами. Алгоритм устроен так, что каждый из узлов получает свой набор дизъюнктов. Новые дизъюнкты, полученные в процессе резольвирования на узле, периодически пересылаются остальным узлам по их запросам. Этот запрос инициируется всякий раз, если на узле завершился процесс резольвирования находящейся на нем порции дизъюнктов или по истечение кванта времени.

Действие алгоритма параллельного вывода на узле кластера

Порции дизъюнктов на узле представляются в его памяти в виде очереди. Для работы с очередью используется дескриптор очереди, состоящий из следующих полей:

- длина очереди (индекс последнего добавленного элемента в очередь);
- индекс последнего отправленного на резольвирование дизъюнкта.

На каждом узле кластера порождаются K нитей, где K выбирается равным числу ядер на узле, которые могут находиться в двух состояниях: в активном, в этом случае нить выполняет резольвирвоание дизъюнктов, и в состоянии ожидания, когда нити будет передан новый дизъюнкт. Нити в состоянии ожидания образуют еще одну очередь — очередь ждущих нитей. Нити поочередно переводятся в активное состояние, им передается дизъюнкт из очереди дизъюнктов и они осуществляют резольвирование этого дизъюнкта со всеми дизъюнктами с меньшими порядковыми номерами в очереди. Полученные в результате резольвирования новые дизъюнкты записываются в конец очереди, а нить помещается в очередь ждущих нитей. Для синхронизации доступа к очереди дизъюнктов на узле используется семафор [3].

Для формализованного описания алгоритма вводятся следующие обозначения:

$S = \langle c_1, c_2, \ldots, c_N \rangle$, $N \geq 1$ – очередь исходного множества дизъюнктов;

LastAdded – индекс последнего добавленного в очередь S дизъюнкта;
LastSentForResolving – индекс последнего отправленного на резольвирование дизъюнкта;
K – количество порождаемых нитей, разумно выбирать K, равным количеству ядер на узле;
Waiting Threads – очередь ждущих нитей;
Sem – двоичный семафор, служащий для синхронизации доступа к очереди S.

Далее при описании алгоритма подразумевается, что если явным образом не указан переход к какому-либо шагу, то осуществляется переход к шагу со следующим порядковым номером.

Начало.

Шаг 1. В очередь S поместить все элементы исходного множества дизъюнктов:
LastSentForResolving = 1.
LastAdded = Length (S).

Шаг 2. Породить K нитей. Нити поместить в очередь WaitingThreads.

Шаг 3. Из очереди WaitingThreads выбрать очередную нить (выбор происходит по алгоритму FIFO), перевести ее в активное состояние.

Шаг 4. Нить захватывает семафор Sem:

Если LastSentResolving < LastAdded, то объявить локальную для нити переменную Number = LastSentForResolving + 1, за нитью «закрепить» дизъюнкт с индексом Number, перейти к шагу 5;

иначе освободить семафор Sem, перевести нить в статус ожидания и поместить ее в конец очереди WaitingThreads, перейти к шагу 3.

Шаг 5. Новое значение LastSentForResolving = Number. Освободить семафор Sem.

Шаги 3–5 будут выполняться до тех пор, пока очередь WaitingThreads не пуста и пока LastSentResolving < LastAdded. Начиная с шага 6 и далее, описывается работа алгоритма для одной нити. На остальных будут выполняться аналогичные операции.

Шаг 6. Объявить локальную для нити переменную j = 1 (индекс следующего дизъюнкта для резольвирования с дизъюнктом с индексом Number).

Шаг 7. *Если* j < Number, то резольвировать дизъюнкты c_j и c_{Number} из S. Новые дизъюнкты, полученные в результате резольвирования, помещаются в множество S_{resolv}, локальное для нити, перейти к шагу 8;

иначе нить перевести в статус ожидания и поместить в конец очереди WaitingThreads, перейти к шагу 3.

Шаг 8. Если в результате выполнения первой части шага 7 был выведен пустой дизъюнкт, тогда послать сигнал останова всем нитям, перейти в конец;

иначе перейти к шагу 9.

Шаг 9. Удалить из очереди S_{resolv} дизъюнкты, которые уже есть в очереди S:

Если множество S_{resolv} не пусто переход к шагу 10;

иначе перейти к шагу 11.

Шаг 10. Захват семафора Sem:
Добавить в конец очереди S дизъюнкты из S_{resolv}.
LastAdded = LastAdded + length(S_{resolv}).
Освободить семафор Sem, перейти к шагу 11.

Шаг 11. j = j + 1. перейти к шагу 7.

Конец.

Действие алгоритма параллельного вывода на межузловом уровне

Для формализованного описания алгоритма вводятся следующие обозначения:

Anode – алгоритм, параллельного вывода на одном узле кластера;

n – количество узлов кластера, используемых для вывода. Узлы нумерют от 0 до (n-1);

$S = <c_1, c_2, ..., c_N>$ – очередь дизъюнктов, первоначально в нее занесены все элементы исходного множества дизъюнктов;

$S_i = <c_1, c_2, ..., c_M>, i = 0 .. (n – 1)$ – очередь дизъюнктов на узле;

$innerTag = <tag_1, tag_2, ..., tag_M>$ – очередь тэгов по числу элементов в очереди дизъюнктов на одном узле. Каждое значение очереди – номер узла, от которого получен этот дизъюнкт;

$outerFlag = <outerFlag_1, outerFlag_2, ..., outerFlag_M>$ – очередь, каждый элемент, которой показывает, каким узлам кластера уже был передан дизъюнкт с соответствующим индексом;

$outerFlag_i = < flag_1, flag_2, ..., flag_n >$ – элемент очереди outerFlag, представляет собой очередь флажков. Если j-й флажок равен 0, значит i-ый дизъюнкт еще не был отправлен на j-й узел, в противном случае – уже был отправлен.

Фактически дизъюнкт представляет собой структуру, которая содержит значение самого дизъюнкта, а также соответствующие ему innerTag и outerFlag;

t – квант времени, по истечение которого узел посылает запрос остальным узлам на новую порцию дизъюнктов. Настраивается опционально;

groupCount – размер пересылаемой порции дизъюнктов. Настраивается опционально. Если *groupCount* устанавливается равным нулю, то пересылаться будут все дизъюнкты, которые еще не были отправлены.

Начало.

Шаг 1. На вычислительном узле с номером 0, множество дизъюнктов S разделить на n равных частей и каждому из вычислительных узлов, включая и сам нулевой узел, разослать свою порцию дизъюнктов. Эту порция поместить в очередь S_i на узлах;

Начиная с шаге 2 и далее, идет описание работы алгоритма для узла с номером i. На остальных узлах будет выполняться точно такой же алгоритм.

Шаг 2. Все начальное множество дизъюнктов, полученное от нулевого узла, маркировать тэгом i, т.е. все элементам очереди innerTag присвоить значение i;

Шаг 3. Запустить таймер для отсчета кванта времени. Запустить алгоритм Anode на узле. Все новые дизъюнкты, получаемые в ходе выполнения Anode, добавить к множеству дизъюнктов S_i, в очередь тэгов innerTag добавить соответствующие тэги со значением i, а в outerFlag добавить соответствующие этим дизъюнктам очереди с нулевыми значениями флагов. При этом, дизъюнкты со значением innerTag отличным от i между собой не резольвировать. Это уже сделано на тех узлах, откуда они получены;

Шаг 4. *Если* на одном из узлов алгоритм Anode нашел пустой дизъюнкт, то остановить вычисления на всех узлах, перейти в конец;

иначе перейти к шагу 5;

Шаг 5. *Если* от j-го узла пришел запрос на очередную порцию дизъюнктов, то из S_i выбрать дизъюнкты, для которых соответствующее значение innerTag равно i и в outerFlag в соответствующей очереди j-ое значение равно 0, перейти к шагу 6;

иначе, перейти к шагу 7;

Шаг 6. *Если* число дизъюнктов найденных на шаге 5 больше или равно, чем groupCount, то отправить первые найденные groupCount дизъюнктов на j-й узел и в outerFlag в соответствующей очереди значение j-го флага устанавить в 1;

иначе, отправить все найденные дизъюнкты на j-й узел и аналогичным образом изменить значения в outerFlag.

Перейти к шагу 7;

Шаг 7. *Если Anode* закончил свою работу и в S_i не осталось непрорезольвированных между собой пар дизъюнктов, то останов таймера, перейти к шагу 10;

иначе, перейти к шагу 8;

Шаг 8. *Если* квант времени t исчерпан, то останов таймера, перейти к шагу 9;

иначе перейти к шагу 4;

Шаг 9. Всем узлам, кроме самого себя (узла с номером i), послать запрос на новую порцию дизъюнктов. Начать неблокирующее ожидание новой порции дизъюнктов, при этом Anode продолжает свою работу на узле. Перейти к шагу 11;

Шаг 10. Всем узлам, кроме самого себя (узла с номером i), послать запрос на новую порцию дизъюнктов. Блокирующее ожидание новой порции дизъюнктов;

Шаг 11. Дизъюнкты, полученные от узла j, добавить в S_i. Дизъюнкты, которые уже есть в S_i не добавляются, но соответствующее j-ому узлу значение в outerFlag устанавить в 1, чтобы такие дизъюнкт не были отправлены j-ому узлу, когда от него придет запрос на новую порцию дизъюнктов;

Шаг 12. *Если* Anode остановлен, то перейти к шагу 3;

иначе, переход к шагу 13;

Шаг 13. Обнулить таймер и запустить заново, перейти к шагу 4;

Конец.

Блок-схема алгоритма параллельного вывода на кластере (для узла с номером *i*)

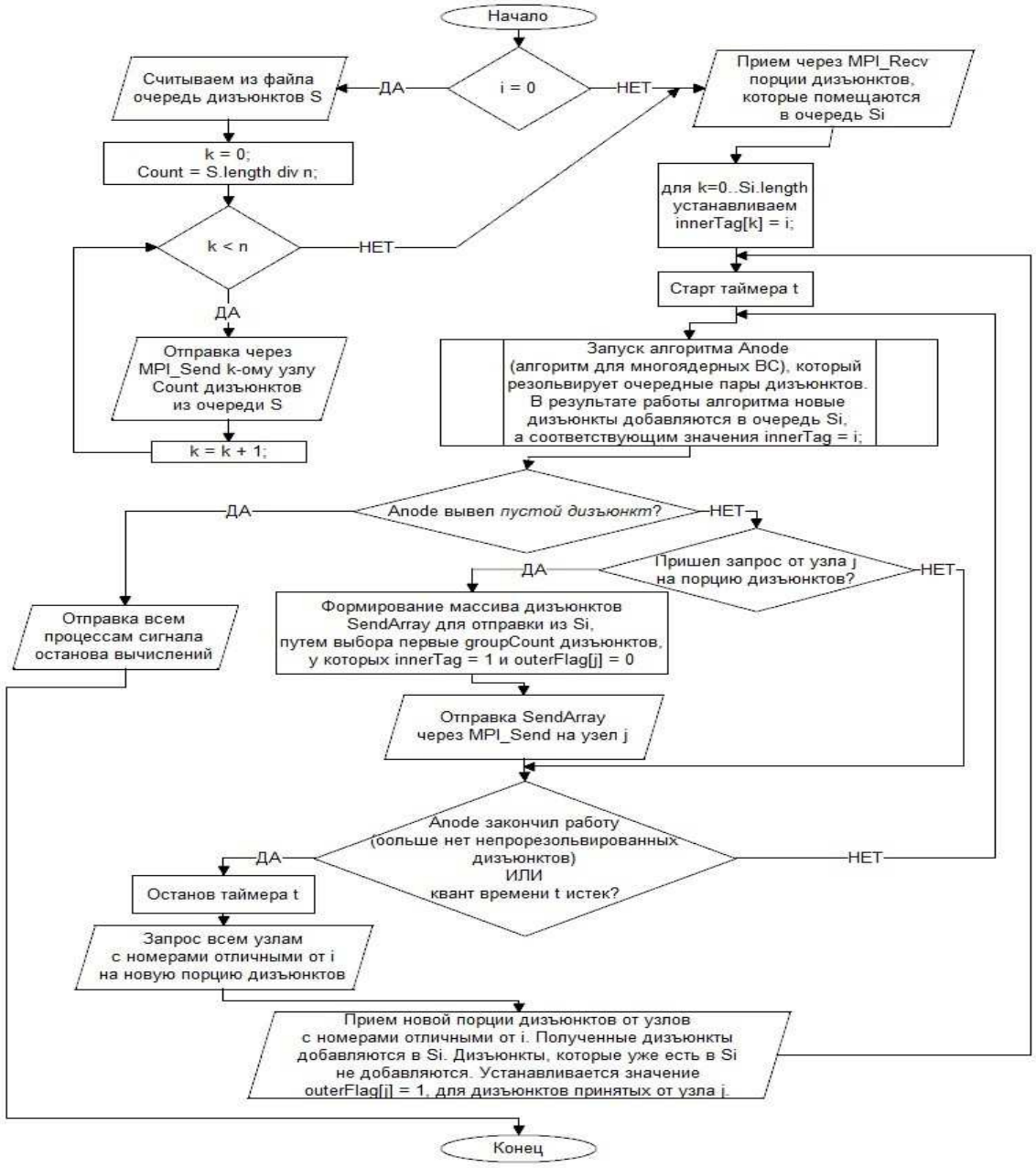

Рис. 1. Блок-схема алгоритма параллельного вывода на кластере

Общая блок-схема алгоритма параллельного вывода на кластере представлена на рис. 1.

2. Особенности реализации алгоритма и результаты экспериментальных исследований на кластере МЭИ (ТУ)

Эффективность разработанного алгоритма параллельного вывода исследована на кластере, описанном во введении. Для реализации алгоритма используется MPI для организации взаимодействий на межузловом уровне, и средства нитевого программирования (multithreading) для описания работы алгоритма на узлах. Использованы широковещательная рассылка данных, блокирующий и неблокирующий прием данных. В качестве языка программирования выбран язык C++.

Для экспериментальное исследования эффективности алгоритма использовались известные задачи логического вывода большой сложности: Steamroller и задача о Ханойских башнях. В качестве критериев эффективности рассматриваются время выполнения алгоритма и ускорение ускорение алгоритма.

На рис. 2–5 показаны результаты экспериментального исследования на кластере МЭИ (ТУ), полученные при доказательстве невыполнимости множества дизъюнктов для этих задач. Так как алгоритм параллельного вывода имеет опциональные параметры, такие как *groupCount* – число дизъюнктов

пересылаемых по запросу и t – квант времени, по исчерпании которого происходит запрос на новые порции дизъюнктов, то при проведении экспериментов варьировалось не только число узлов, но и эти параметры, в целях достижения максимальной эффективности работы алгоритма.

На рис. 2 и 3 представлены графики времени выполнения и ускорения для задачи Steamroller. Алгоритм выполнялся со следующими опциональными параметрами: groupCount равен 0, т.е. пересылаются все новые дизъюнкты, квант времени t равен 1 секунде. Число нитей, запущенных на узле кластера = 4.

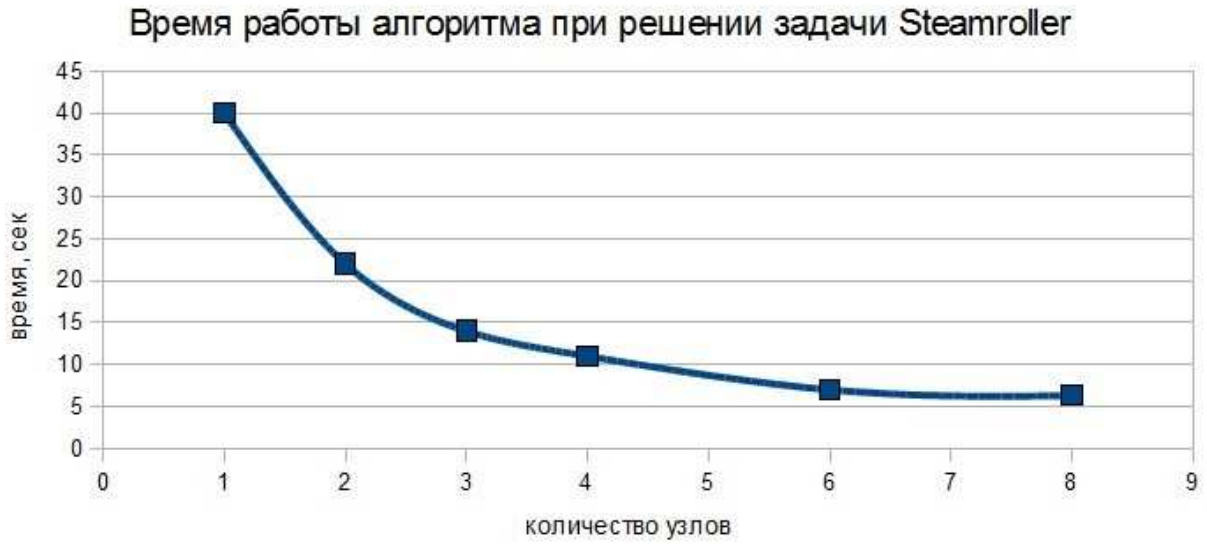

Рис. 2. Зависимость времени работы алгоритма от числа узлов для задачи Steamroller

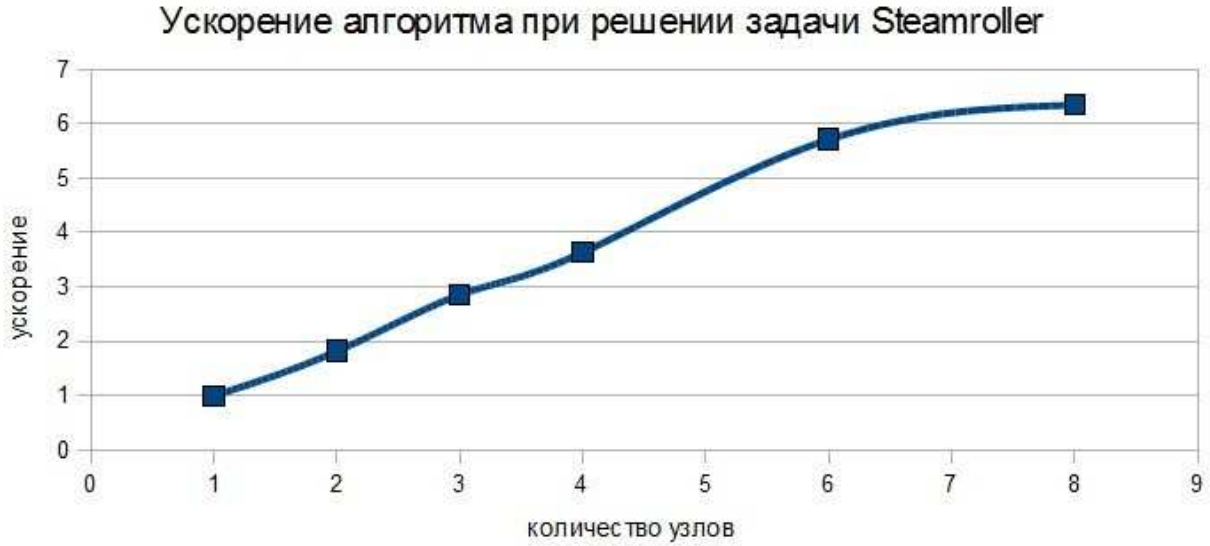

Рис. 3. Зависимость ускорения алгоритма от числа узлов для задачи Steamroller

На рис. 4 и 5 представлены графики времени выполнения и ускорения для задачи о Ханойских башнях с 5 кольцами. Алгоритм выполнялся со следующими опциональными параметрами: groupCount равен 0, т.е.

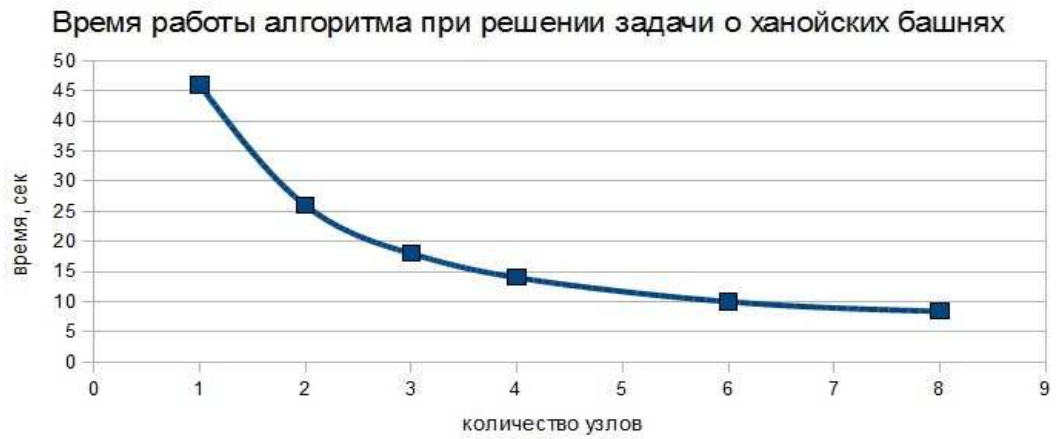

Рис. 4. Зависимость времени работы алгоритма от числа узлов для задачи о ханойских башнях с 5 кольцами

пересылаются все новые дизъюнкты, квант времени t равен 1 секунде. Число нитей, запущенных на узле кластера = 4.

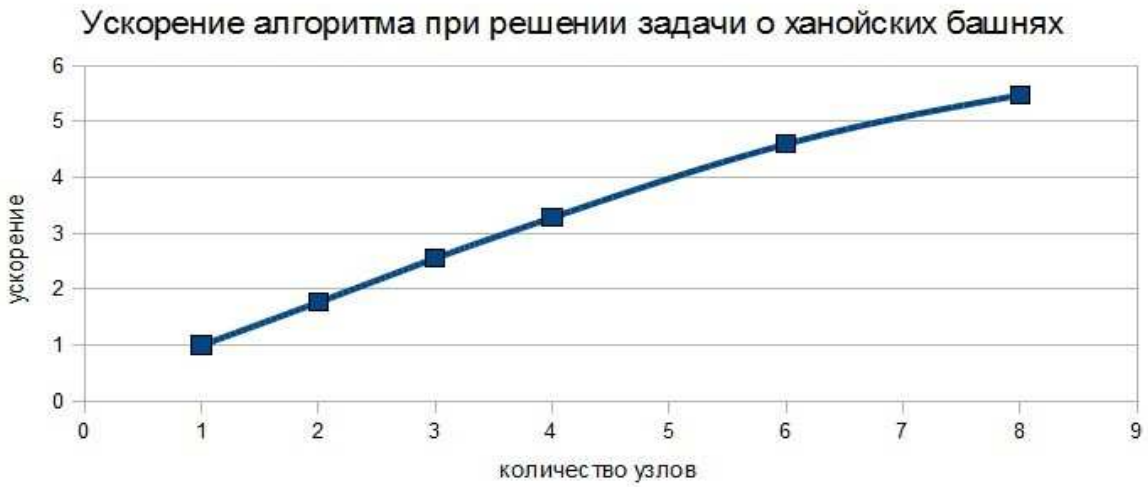

Рис. 5. Зависимость ускорения алгоритма от числа узлов для задачи о ханойских башнях с 5 кольцами

Результаты экспериментов показывают, что алгоритм повышает эффективность дедуктивного вывода, и позволяет использовать для решения задачи логического вывода, как мощности каждого узла в отдельности, благодаря технологии multithreading'a, используемой на узле, так и мощности кластера, за счет организации межузловых взаимодействий. Однако, для каждого класса задач необходимо правильно выбирать квант времени между запросами на пересылку данных. В случае, когда квант слишком мал, то количество пересылок возрастает и увеличивается нагрузка на коммуникации, а если квант слишком велик, то возникают ситуации, когда процессоры начинают простаивать.

Полученные результаты дают почву для дальнейшего усовершенствования алгоритма путем автоматизации выбора кванта времени в ходе работы алгоритма, и динамического изменения кванта времени в зависимости от загруженности узлов.

Заключение

Результаты экспериментов позволяют сделать следующие выводы.

1. Производная ускорения и времени вывода уменьшается с увеличением числа узлов. Для рассматриваемых задач уже на восьми узлах достигается «почти» максимальное значение этих критериев. Причиной является заметное увеличение интенсивности и числа межузловых обменов. Пропускная способность каналов у кластера МЭИ (ТУ) – 1010 бит/сек., латентность в пределах 1–2 мк/сек., и при интенсивных межузловых обменах каналы становятся сдерживающим эффективность фактором.
2. Выбор кванта времени, определяющего частоту межузловых обменов, является достаточно «тонкой» задачей. Уменьшая квант, мы уменьшаем простой процессоров на узлах, но при этом увеличивается частота обменов. Увеличивая квант, мы уменьшаем частоту обменов, но при этом увеличиваем вероятность простоя процессоров.
3. Требуются дополнительные исследования, и, возможно, усовершенствование алгоритма параллельного логического вывода. Необходимо исследовать влияние семафора, используемого на узле для синхронизации работы нитей, на критерии эффективности. В частности, в статье [4] предложен более совершенный алгоритм вывода на многоядерных системах, в котором не используется семафор.

ЛИТЕРАТУРА:

1. Ч. Чень, Р. Ли, «Математическая логика и автоматическое доказательство теорем», пер. на русский Издательство «Наука», 1983
2. В.Н. Вагин, Е.Ю. Головина, А.А. Загорянская, М.В. Фомина «Достоверный и правдоподобный вывод в интеллектуальных системах», М.: ФИЗМАТЛИТ, 2004
3. В.П. Кутепов, К.Ю. Хотимчук «Параллельный логический вывод на многоядерном компьютере», статья, 2010.
4. Д.С. Зарецкий «Параллельная реализация принципа резолюции с использованием управляющих эвристик», статья, 2010