В. П. Кутепов, М.М. Кумачев

Параллельный логический вывод на компьютерных системах[1]

Аннотация. В статье исследуются различные алгоритмы параллельного логического вывода, основанные на принципе резолюции. Проведены экспериментальные исследования алгоритмов на компьютерных системах с общей памятью и кластере. Описанные результаты показывают влияние архитектуры и характеристик компьютерных систем, степени распараллеливания и эвристик на эффективность параллельного вывода.

Ключевые слова: логика, принцип резолюции, алгоритмы, параллельные вычисления, параллельный логический вывод.

Введение

В настоящее время все активнее развиваются интеллектуальные системы, которые опираются на логический вывод первого и высших порядков, поскольку логика – самый высокий уровень формализации человеческих рассуждений, математики, различных предметных областей, с которыми человек сталкивается на практике при решении различных задач. Для упрощения описания предметной области и решаемой задачи логическими средствами, а также для реализации на компьютере алгоритмов логического вывода были разработаны языки логического программирования (Planner, Prolog и другие). Вместе с тем, известно, что проблема вывода в логике первого порядка является полуразрешимой, а сложность алгоритмов растет экспоненциально в зависимости от сложности задачи, и длительное время выполнения алгоритма может поставить пользователя перед неразрешимой дилеммой: либо решить, что вывод отсутствует, либо ожидать неопределенное время, пока он не будет получен. Поэтому разработка эффективных параллельных алгоритмов логического вывода с возможностью их реализации на современных компьютерных системах является актуальной задачей и расширяет круг задач, которые могут быть решены с помощью логических средств.

Реализация алгоритмов параллельного логического вывода на многопроцессорных компьютерных системах (КС), в частности на узлах кластера, в принципе может являться более эффективной, чем кластерная реализация, так как исключается необходимость обмена данными между узлами. Однако кластерные КС предполагают большие возможности масштабирования, а, следовательно, и большую степень распараллеливания.

В статье рассматриваются алгоритмы параллельного логического вывода в логике первого порядка и приводятся результаты исследования их эффективности на компьютерных системах, в частности, 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/с и пропускной способностью каналов передачи данных между узлами равной 10 Гбит/с; объем оперативной памяти узла – 4 Гб.

[1] Работа выполнена при поддержке РФФИ, проект 11-01-00232а

1. Алгоритмы параллельного логического вывода. Критерии и параметры их эффективности

Исследуемые в статье алгоритмы параллельного логического вывода основаны на принципе резолюции, предложенным Дж. Робинсоном и широко применяемым на практике.

Общая идея резолютивного вывода заключается в представлении отрицания доказываемой формулы логики первого порядка в конъюнктивной нормальной форме (КНФ) [1] или в практическом плане отрицания множества дизъюнктов S и последующем попарном резольвировании (применении принципа резолюции) дизъюнктов из объединения S с получаемыми в процессе резольвирования новыми дизъюнктами или резольвентами. Этот процесс продолжается до тех пор, пока не будет выведен «пустой» дизъюнкт (значение ложь), или процесс резольвирования не имеет продолжения. Первый случай является доказательством истинности исходной формулы, второй указывает на недостаточность аргументов для доказательства отрицания. В [2] вместо доказательства ложности отрицания исходной формулы методом резолюции используется прямое доказательство истинности исходной формулы, приведенной к конъюнктивной нормальной форме. Это возвращает процесс вывода в его естественное и привычное для человека русло выполнения доказательства и, кроме этого, если вывод не может быть продолжен, то оставшиеся конъюнкты подсказывают какие должны быть добавлены вспомогательные посылки, чтобы можно было доказать истинность исходной формулы логики первого порядка. Это является важным обстоятельством в формализации процедур активно исследуемого в настоящее время принципа абдуктивного вывода [3].

Стратегии распараллеливания алгоритмов логического вывода, в частности, основанного на принципе резолюции, достаточно хорошо известны (например, [4, 5]). Их легко можно упорядочить по глубине или степени распараллеливания [6], начиная от распараллеливания процессов унификации, а затем увеличивая зернистость распараллеливания на уровне одновременного резольвирования всех литер одного дизъюнкта с литерами другого дизъюнкта (так называемый OR-параллелизм) или с литерами всех других дизъюнктов (AND-параллелизм) [4, 5].

Однако эффективность реализации алгоритмов параллельного логического вывода на современных многопроцессорных (многоядерных) системах с общей памятью и на кластерах существенно зависит от выбора оптимальной степени распараллеливания, поскольку мелкозернистое распараллеливание приведет к большим затратам времени на управление процессами вывода и обмен данными компонентами компьютерной системы.

Поэтому в рассматриваемых в статье алгоритмах параллельного вывода используются стратегии крупнозернистого распараллеливания, когда каждый процесс осуществляет вывод над выделенным ему подмножеством множества всех резольвируемых на определенном шаге дизъюнктов.

Прежде чем перейти к описанию алгоритмов, необходимо сформулировать основные требования и критерии их эффективности. В качестве критериев эффективности параллельного алгоритма обычно используется время его выполнения на компьютерной системе $T(N, \bar{x})$, коэффициент ускорения $C(N, \bar{x}) = T(1, \bar{x}) / T(N, \bar{x})$ и использование ресурсов $E(N, \bar{x}) = T(1, \bar{x}) / (C(N, \bar{x}) \times N)$. Здесь N – количество процессорных элементов в КС, $\bar{x}$ – другие параметры, существенно влияющие на эффективность [6, 7].

В качестве этих параметров обычно выступают:

– сам алгоритм и его реализация (степень или глубина распараллеливания, количество выполняемых параллельных процессов, обеспечение совместного доступа к разделяемым данным, использование семафоров и другие);

– интенсивность обменов между вычислителями (этот параметр представляет особый интерес при реализации алгоритма на многоузловых кластерных системах);

– интенсивность обменов данными между кэшируемой и оперативной памятью, оперативной памятью и жестким диском (так называемый «свопинг»).

Исследуемые в статье алгоритмы параллельного логического вывода в общем случае представляются в виде конечного множества одновременно выполняемых процессов, для представления которых используются средства MULTITHREADING при их реализации на

многопроцессорных (многоядерных) КС и средства MPI при реализации на кластерах. Реализация на кластерах требует более тонкого подхода к выбору степени распараллеливания и разделения множества резольвируемых дизъюнктов на подмножества с тем, чтобы уменьшить негативное влияние обмена данными между процессами (узлами кластера) на время выполнения алгоритмов.

Помимо требования полноты, т.е. обеспечения резольвирования любой пары дизъюнктов из множества S за конечное время, достижение высокой эффективности алгоритмов параллельного логического вывода предполагает, что они должны удовлетворять следующим условиям:

– исключение повторного резольвирования одинаковых пар дизъюнктов, что ведет к уменьшению T(N, $\bar{x}$) и используемой оперативной памяти;

– оптимальное разделение множества одновременно резольвируемых дизъюнктов на подмножества и уменьшение роли централизации в управлении взаимодействием процессов с целью уменьшение интенсивности обменов данными между компонентами КС при выводе.

В общем случае реализующий принцип резолюции алгоритм можно представить состоящим из двух основных действий: выбора резольвируемого дизъюнкта D_α и выбора дизъюнкта D_β, с которым он должен быть прорезольвирован.

Ясно, что выбор дизъюнктов D_α и D_β и их резольвирование могут выполняться как параллельные процессы. В более общем виде действия каждого такого процесса можно представить состоящим из двух циклов: выбора из множества дизъюнктов S подмножеств $S_1 \subseteq S$ и $S_2 \subseteq S$ дизъюнктов, где S_1 содержит дизъюнкты, которые должны быть прорезольвированы с дизъюнктами из S_2, и последующего выполнения собственно операции резольвирования. Тем, каким образом реализуется выбор подмножеств S_1 и S_2, резольвируемых параллельными процессами дизъюнктов, а также сам процесс резольвирования, определяется специфика конкретного алгоритма параллельного вывода в логике первого порядка. По этим принципам строятся алгоритмы, исследуемые в статье.

Первый алгоритм, который можно назвать алгоритмом с общей очередью, подразумевает, что все процессы имеют общий доступ к множеству дизъюнктов S, представленных в виде очереди, при этом дизъюнкты в очереди пронумерованы. Далее, за каждым процессом по очереди закрепляется дизъюнкт из этого множества («активный» дизъюнкт), и процесс производит резольвирования закрепленного дизъюнкта со всеми дизъюнктами из S с меньшим порядковым номером в очереди, а полученные резольвенты помещаются в конец очереди дизъюнктов. Когда процесс закончил резольвирование закрепленного за ним активного дизъюнкта со всеми дизъюнктами из очереди S с меньшим порядковым номером, за ним закрепляется новый дизъюнкт, который еще не был закреплен ни за одним из процессов, и снова начинается процесс резольвирования.

Для формализованного описания алгоритма с общей очередью 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, перейти 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.

Такой алгоритм позволяет легко избежать повторных резольвирований, однако требует обеспечения совместного доступа к общей очереди дизъюнктов и использование семафора, что при увеличении количества процессов приводит к уменьшению эффективности параллельного вывода [8, 9].

Чтобы избежать накладных расходов от использования общей очереди, предлагается альтернативная реализация первого алгоритма, которую можно назвать алгоритмом с управляющим процессом. Эта реализация подразумевает, что каждый процесс хранит свою копию множества всех дизъюнктов S, представленную в виде очереди, а также вводится управляющий процесс, который также хранит свою копию множества резольвируемых дизъюнктов S. Последовательно выбирая из своей очереди дизъюнктов активные дизъюнкты, управляющий процесс распределяет их между остальными параллельными процессами. Каждый процесс, «получив» активный дизъюнкт, начинает резольвировать его со всеми дизъюнктами из своей очереди с меньшим порядковым номером, чем у активного дизъюнкта (это позволяет избежать повторений). Полученные каждым процессом резольвенты отправляются в очередь управляющего процесса, который их нумерует и возвращает остальным процессам. Эти дизъюнкты дописываются в конец очередей каждого процесса. Таким образом, соблюдается полнота и достигается отсутствие повторений. Очевидным недостатком этого алгоритма является большой расход памяти, так как каждый параллельный процесс хранит полную копию всего множества дизъюнктов.

Еще один алгоритм, который можно назвать алгоритмом с разделяемым множеством дизъюнктов, позволяет сократить объем используемой памяти и при этом отказаться от семафора. Этот алгоритм также подразумевает наличие управляющего процесса, который содержит все множество дизъюнктов S в памяти в виде очереди. Копия этого множества дизъюнктов равномерно распределяется между остальными процессами, т.е. каждый процесс, кроме управляющего, содержит некоторое подмножество множества дизъюнктов S. Далее алгоритм работает по следующему циклу: управляющий процесс последовательно выбирает из своей очереди очередной дизъюнкт и отправляет его на резольвирование остальным процессам. При этом в отличие от предыдущего алгоритма, выбранный дизъюнкт не закрепляется за одним из процессов, а рассылается всем процессам сразу. «Получив» дизъюнкт от управляющего процесса, процессы резольвируют полученный дизъюнкт со всеми «своими» дизъюнктами, т.е. с тем подмножеством дизъюнктов, которые хранятся в очереди процесса. Все полученные процессами резольвенты отправляются управляющему процессу. После того, как управляющий процесс собрал все резольвенты от всех процессов на очередном шаге цикла, он равномерно разделяет их на подмножества и рассылает остальным процессам, для продолжения резольвирования. После этого цикл считается завершенным и новый цикл опять начинается

с выбора управляющем процессом дизъюнкта, который будет отправлен на резольвирование. Принцип полноты выполняется за счет того, что этот алгоритм также резольвирует дизъюнкты по принципу «каждый с каждым», так как управляющий процесс рассылает каждый дизъюнкт на резольвирование всем процессам сразу.

Алгоритм с разделяемым множеством дизъюнктов является централизованным, поскольку все параллельные процессы синхронизируются с управляющим процессом, что является существенным недостатком этого алгоритма

Важно отдельно сказать о разделении множества дизъюнктов на подмножества. Как было описано выше, алгоритм с разделяемым множеством дизъюнктов использует равномерное разделение дизъюнктов на подмножетсва, но, очевидно, это не всегда оптимальное решение, и вопрос эффективного разделения множества дизъюнктов на подмножества остается открытым.

Для повышения эффективности выполнения всех алгоритмов также используются управляющие эвристики [10]. Одной из таких эвристик является динамическое упорядочивание по «сложности» множества резольвируемых дизъюнктов S в ходе работы алгоритма. Идея этой эвристики возникла из очевидного факта: вывести пустой дизъюнкт можно только путем резольвирования двух однолитеральных дизъюнктов, образующих контрарную пару. Она реализована следующим образом: те дизъюнкты, которые хранятся в очереди управляющего процесса, динамически упорядочиваются [10]. Под упорядочиванием здесь понимается размещение дизъюнктов в памяти по возрастанию числа литер в дизъюнкте, что позволяет в первую очередь резольвировать между собой дизъюнкты меньшей длины. Поскольку, упорядочивание множества S каждый раз после добавления новой резольвенты повлекло бы за собой большие накладные расходы, то для выполнения этой процедуры используется хорошо известный алгоритм сортировки включением, что позволяет не просматривать каждый раз все множество дизъюнтов.

Другой эвристикой является метод границы. Идеей метода границы является введение динамически изменяемого ограничения на длину одного из дизъюнктов. Вводится граница, изначально равная 1. Далее резольвироваться будут только те пары дизъюнктов, в которых длина хотя бы одного из дизъюнктов не превышает значения границы. В случае, если результат (пустой дизъюнкт) не был получен, то значение границы увеличивается на единицу и резольвирование продолжается.

2. Экспериментальные исследования эффективности алгоритмов параллельного логического вывода

Как уже было отмечено выше, реализация предложенных алгоритмов логического вывода существенно зависит от КС. Для компьютерных систем с распределенной памятью (кластеров) реализация существенно усложняется, так как помимо разделения множества дизъюнктов на параллельно резольвируемые подмножества, которые будут храниться в памяти разных узлов КС, требуется постоянный обмен вновь полученными дизъюнктами между узлами с целью обеспечения полноты процедуры вывода.

В этом разделе приводятся и анализируются результаты эффективности алгоритмов параллельного вывода на компьютерных системах с общей памятью, в качестве которых в данном случае выступает узел кластера МЭИ (ТУ), содержащий 4 ядра, что позволяет одновременно выполнять 4 процесса.

Для исследования выбраны алгоритм с общей очередью (введем сокращение А1) и алгоритм с разделяемым множеством дизъюнктов (сокращение А2), как два принципиально разных подхода к решению поставленной задачи с точки зрения распределения дизъюнктов между узлами и модели вычислений. Эти алгоритмы реализованы с использованием средств нитиевого программирования MULTHITREADING. При этом количество нитей определяет количество одновременно выполняемых процессов резольвирования.

В качестве тестовых задач для исследования эффективности алгоритмов используются хорошо известные и ставшие почти стандартными задачи о Ханойских башнях и Steamroller, характеризующиеся лавинообразным ростом числа дизъюнктов в ходе вывода [1].

2.1. Исследование эффективности алгоритмов А1 и А2 на узле кластера

Результаты экспериментальных исследований алгоритмов представлены в Табл. 1 и Табл. 2. Поскольку используются средства ни-

тиевого программирования, то одна нить эквивалентна одному процессу в терминах исследуемых алгоритмов.

Время выполнения алгоритмов представлено в секундах.

Результаты экспериментов показывают, как с ростом числа нитей (процессоров) уменьшается время выполнения алгоритмов. Стоит отметить, что алгоритмы решают задачу о Ханойских башнях с четырьмя и пятью кольцами примерно за одно и тоже время, но на шести кольцах первый алгоритм начинает проигрывать.

Прежде, чем сделать выводы о том, почему были получены такие результаты, рассмотрим еще две характеристики алгоритмов: количество выведенных в ходе работы алгоритма уникальных дизъюнктов N, т.е. количество неповторяющихся дизъюнктов в резольвируемом множестве, а также количество произведенных резольвирований Resol. В Табл. 3 и Табл. 4 представлены эти данные.

На задаче о Ханойских башнях для четырех и пяти колец количество выведенных уникальных дизъюнктов и общее число резольвирований, проделанное алгоритмами, примерно равны. При этом из Табл. 1 и Табл. 2 видно расхождение во времени на четырех кольцах. Это может быть обусловлено тем, что на относительно небольших задачах доля влияния семафора и накладные расходы на работу с общей очередью оказывают значимое влияние на итоговое время выполнения A1.

Табл. 1. Сравнение алгоритмов логического вывода по времени выполнения

Число нитей	Задача о Ханойских башнях						Steamroller	
	4 кольца		5 колец		6 колец			
	A1	A2	A1	A2	A1	A2	A1	A2
1	00.594	00.390	12.531	13.775	1809.696	1166.275	03.202	00.950
2	00.443	00.238	07.231	08.773	1352.445	764.729	02.665	00.550
3	00.325	00.201	06.312	06.569	1171.339	596.001	02.342	00.445
4	00.310	00.192	05.642	06.114	1097.001	529.875	02.336	00.396

Табл. 2. Сравнение алгоритмов логического вывода по ускорению

Число нитей	Задача о Ханойских башнях						Steamroller	
	4 кольца		5 колец		6 колец			
	A1	A2	A1	A2	A1	A2	A1	A2
1	1	1	1	1	1	1	1	1
2	1.34	1.64	1.73	1.57	1.34	1.53	1.2	1.73
3	1.83	1.94	1.99	2.1	1.54	1.96	1.37	2.13
4	1.92	2.03	2.22	2.25	1.65	2.20	1.38	2.4

Табл. 3. Количество дизъюнктов N и резольвирований Resol для A1

Число нитей	Задача о Ханойских башнях						Steamroller	
	4 кольца		5 колец		6 колец			
	N	Resol	N	Resol	N	Resol	N	Resol
1	796	1502	6200	12056	80524	143215	6841	14112
2	810	1543	6034	11479	80520	143204	6932	14879
3	782	1487	6112	11589	80451	143111	6949	14999
4	818	1569	6146	11697	80438	142996	6950	15008

Табл. 4. Количество дизъюнктов N и резольвирований Resol для A2

Число нитей	Задача о Ханойских башнях						Steamroller	
	4 кольца		5 колец		6 колец			
	N	Resol	N	Resol	N	Resol	N	Resol
1	844	1521	6191	11048	53725	96289	2972	8269
2	780	1401	6222	11081	53782	96383	2797	7769
3	844	1514	6180	11001	53704	96219	2704	7501
4	843	1506	6168	10990	53455	95591	2706	7498

В то же время для задач Steamroller и о Ханойских башнях для шести колец иная картина: первый алгоритм работал дольше, но и объем работы, проделанной первым алгоритмом тоже больше. Это говорит о том, что разделение множества дизъюнктов между процессами в А2 оказалось более удачным подходом, чем работа с общей очередью в А1. И, более того, само разделение множества дизъюнктов на порции было выполнено эффективно, что позволило вывести пустой дизъюнкт быстрее, чем в первом алгоритме.

2.2. Исследование влияния управляющих эвристик на эффективность алгоритма А2

Рассмотрим влияние управляющих эвристик на алгоритм А2, так как для него внедрение эвристик упорядочивания и границы является естественными и реализуются без изменения структуры алгоритма, в то время как внедрение эвристик для алгоритма с общей очередью (А1) требует дополнительных модификаций в структуре алгоритма.

В Табл. 5 и Табл. 6 представлены результаты исследований алгоритма А2 с использованием управляющих эвристик.

Время выполнения программ представлено в секундах.

В Табл. 7 показано количество выведенных уникальных дизъюнктов и число произведенных резольвирований.

В этих экспериментах рассматривалось совместное влияние управляющих эвристик динамического упорядочивания и границы, рассмотренные в разделе 1.

Полученные результаты позволяют сделать следующие выводы:

1) время логического вывода с использованием эвристик уменьшается почти на порядок в зависимости от числа нитей (процессоров);

2) коэффициент ускорения, достигаемый при использовании управляющих эвристик, достаточно близок к значению коэффициента ускорения без их использования. Это объясняется временем, которое затрачивается на процедуру упорядочивания дизъюнктов при выводе с использованием эвристик;

3) использование управляющих эвристик может приводить и к увеличению времени вывода при увеличении степени распараллеливания (случай задачи Steamroller для трех и четырех нитей из Табл. 5).

Табл. 5. Время выполнения А2 с использованием управляющих эвристик

Число нитей	Задача о Ханойских башнях			Steamroller
	4 кольца	5 колец	6 колец	
	Время выполнения	Время выполнения	Время выполнения	Время выполнения
1	00.119	01.572	52.868	00.117
2	00.069	00.853	28.614	00.073
3	00.052	00.558	22.296	00.068
4	00.042	00.429	17.969	00.082

Таблица 6. Ускорение А2 с использованием управляющих эвристик

Число нитей	Задача о Ханойских башнях			Steamroller
	4 кольца	5 колец	6 колец	
	Время выполнения	Время выполнения	Время выполнения	Время выполнения
1	1	1	1	1
2	1.72	1.84	1.85	1.6
3	2.29	2.82	2.37	1.72
4	2.83	3.66	2.94	1.43

Табл. 7. Количество дизъюнктов N и резольвирований Resol для А2 с использованием управляющих эвристик

Число нитей	Задача о Ханойских башнях						Steamroller	
	4 кольца		5 колец		6 колец			
	N	Resol	N	Resol	N	Resol	N	Resol
1	475	625	2255	3039	13805	18415	1140	1175
2	470	614	2211	2926	13643	18016	1057	1129
3	482	635	2199	2914	13652	18080	1312	1443
4	467	603	2086	2746	13371	17664	1592	1691

3. Эффективность алгоритмов параллельного вывода

Реализация алгоритмов параллельного логического вывода на кластере открывает большие возможности распараллеливания в силу больших возможностей по увеличению числа процессоров (узлов). Однако при этом возникает необходимость использования обменов данными между узлами по каналам связи, что требует более тонкого выбора степени распараллеливания, от которой зависит количество параллельно выполняемых процессов и частота межузловых обменов.

Исследуемые алгоритмы параллельного вывода на кластере основываются на разделении множества дизъюнктов на подмножества с последующим их параллельным резольвированием на разных узлах кластера. Взаимодействие узлов между собой должно быть организовано так, чтобы уменьшить число обменов между узлами. Предлагаемый алгоритм является полностью децентрализованным, и множество резольвируемых дизъюнктов распределяется между узлами кластера так, чтобы «уравновесить» их работу. На каждом из узлов кластера, в свою очередь, выполняются алгоритмы параллельного вывода, которые использовались ранее для многопроцессорных компьютерных систем. Новые дизъюнкты, полученные в процессе резольвирования на узле, пересылаются другим узлам кластера в следующих случаях: или по запросу от одного из узлов (это может произойти, если узел уже произвел все возможные резольвирования пар дизъюнктов, которые на нем хранятся), или по истечении определенного кванта времени.

Такая организация модели вычислений обеспечивает большую асинхронность в работе узлов при выводе и уменьшает их простой.

Для реализации алгоритма параллельного логического вывода на кластере использовалась реализация стандарта MPI.

3.1. Действия алгоритма на узле кластера

Алгоритм параллельного логического вывода на кластере представляет собой множество параллельно выполняемых «MPI-процессов», каждый из которых распределен на один из узлов кластера. «MPI-процесс» выполняет резольвирование подмножества дизъюнктов, распределенных на узел, используя модифицированный алгоритм A1.

Модификация алгоритма A1 затронула представление дизъюнктов в памяти, а также был изменен принцип выбора дизъюнктов, отправляемых на резольвирование.

В памяти узла кластера дизъюнкты представляются следующим образом: каждый дизъюнкт содержит дескриптор, в котором хранится информация о том, на каком узле этот дизъюнкт был выведен и каким узлам он уже был передан.

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 поступления нового дизъюнкта. Нити в состоянии ожидания образуют очередь ждущих нитей. Они поочередно переводятся в активное состояние по мере получения от других узлов новых дизъюнктов, осуществляя их резольвирование только со всеми «собственными» дизъюнктами, т.е. дизъюнктами, которые были распределены на этот узел при разделении начального множества дизъюнктов на подмножества, или дизъюнктами, которые были получены в результате резольвирования на этом узле. Новые дизъюнкты, получаемые в результате резольвирвоания, записываются в конец очереди, а нить, завершившая резольвирование, помещается в очередь ждущих нитей.

3.2. Действия алгоритма на межузловом уровне

Для формализованного описания алгоритма введем следующие обозначения:

Anode – обозначение для модифицированного алгоритма A1, который выполняется на узле кластера;

n – количество узлов кластера, используемых для вывода. Узлы нумерются 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 i;

$innerTag = <tag_1, tag_2, ..., tag_M>$ – 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; 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;

t – квант времени, по истечении которого узел посылает запрос остальным узлам на новую порцию дизъюнктов; настраивается опционально;

groupCount – размер пересылаемой порции дизъюнктов; настраивается опционально; если *groupCount* устанавливается равной нулю, то пересылаться будут все дизъюнкты, которые еще не были отправлены.

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

Ниже приводится формальное описание алгоритма.

Начало.

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

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

Шаг 2. Все начальное множество дизъюнктов, полученное от нулевого узла, маркировать тэгом i, т.е. всем 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 $outerFlag_j$ 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 вывел пустой дизъюнкт, то остановить вывод 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 установить 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 остановить таймер, перейти к шагу 10; *иначе*, перейти к шагу 8.

Шаг 8. *Если* квант времени t исчерпан, то остановить таймер, 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. Перейти в 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 (эта ситуация возможна, так как в ходе вывода могут быть получены одинаковые резольвенты), но соответствующее j-ому узлу значение в $outerFlag_i$ устанавить в 1, чтобы такие дизъюнкты 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 описанного алгоритма параллельного вывода на кластере представлена на Рис. 1.

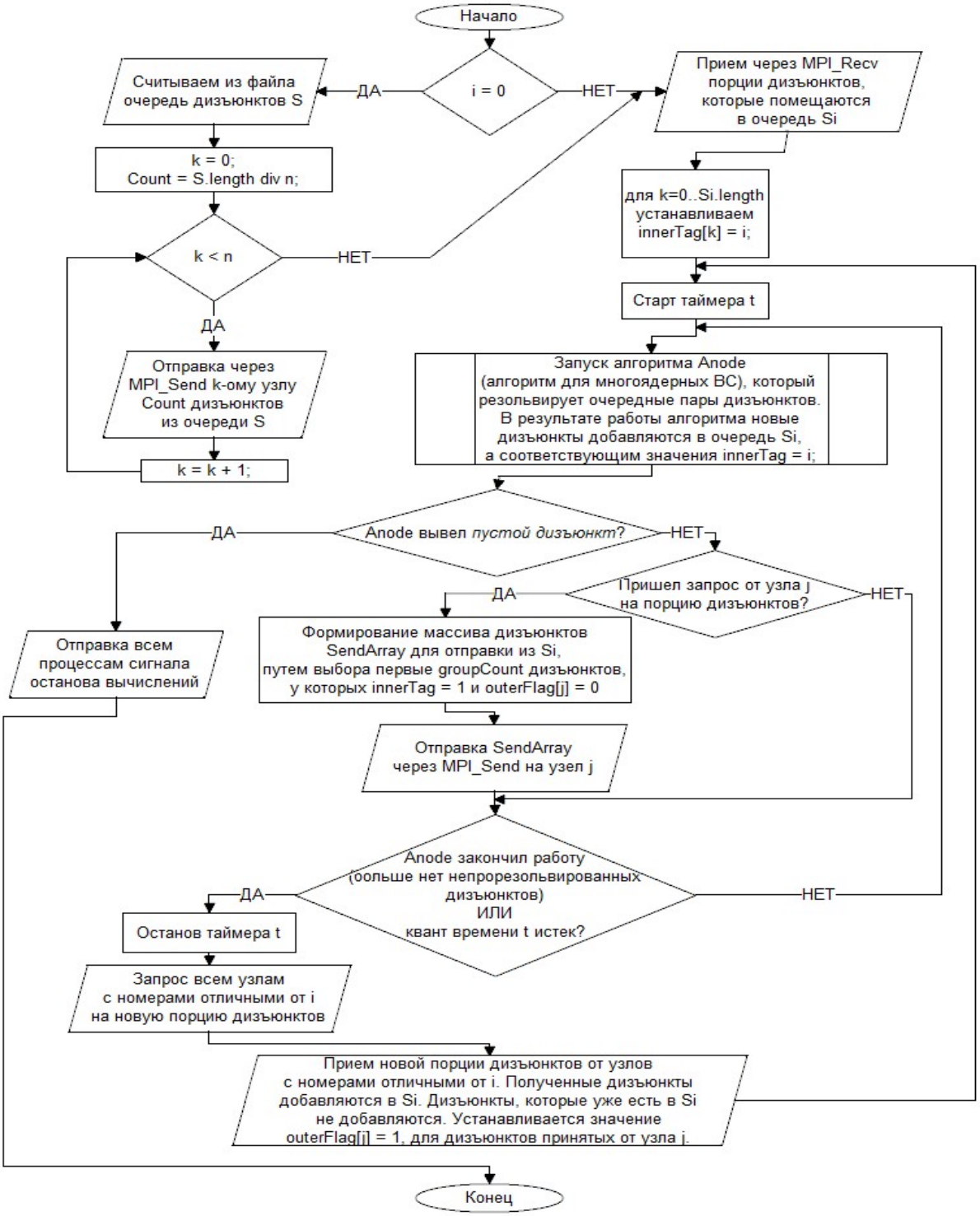

Рис. 1. Блок-схема алгоритма параллельного логического вывода на кластере (действия на узле с номером i)

4. Результаты исследований эффективности алгоритма параллельного логического вывода

Описанный в разделе 3 алгоритм параллельного вывода реализован с использованием стандарта MPI, так как MPI позволяет обеспечить эффективный обмен данными между узлами вычислительной системы, а также реализует такие важные для описанного алгоритма функции, как широковещательная рассылка данных, блокирующий и неблокирующий прием данных от других процессов и другие. Действия алгоритма на узле кластера описаны средствами MULTITHREADING. В качестве языка реализации был выбран язык C++, для которого существует реализация стандарта MPI.

Для выполнения на узле кластера был выбран алгоритм A1 в силу того, что он может работать асинхронно. Все дизъюнкты, полученные в ходе резольвирования на узле, сохраняются в общей очереди, причем добавление дизъюнктов в очередь происходит сразу после того, как произведено очередное резольвирование. Это важное свойство алгоритма A1 позволяет увеличить эффективность межузловых обменов дизъюнктами, так как при получении запроса на отправку очередной порции дизъюнктов узел имеет возможность отправить все «новые» дизъюнкты, даже если выполнение алгоритма A1 на узле еще не завершено. Алгоритма A2 такой возможности не предоставляет, потому что до тех пор, пока все нити не завершат очередной цикл резольвирований, управляющий процесс не получит «новые» дизъюнкты.

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

Экспериментальные исследования эффективности алгоритма проводились на кластере МЭИ, технические характеристики которого приведены во введении.

На Рис. 2–7 показаны результаты экспериментов, полученные при доказательстве невыполнимости множества дизъюнктов указанных задач.

Так как алгоритм параллельного вывода на кластерах 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. Экспериментально было получено, что наиболее эффективной является пересылка по кванту времени всех новых дизъюнктов, полученных на узле, другому узлу. Подбор кванта времени является достаточно сложной задачей, и в приведенных примерах использован фиксированный квант времени равный 0,1 сек задач

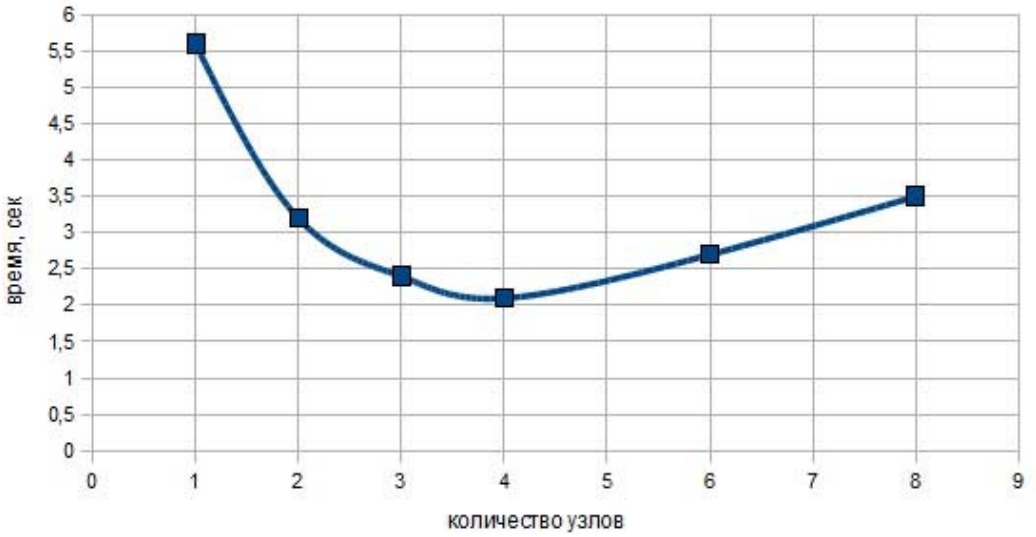

Рис. 2. Зависимость времени выполнения алгоритма от числа узлов кластера для задачи Ханойские башни, 5 колец (на каждом узле выполняется по 4 нити)

о Ханойских башнях (5 колец) и Steamroller и 0,5 сек для задачи о Ханойских башнях (6 колец), однако поиск оптимального кванта времени в зависимости от алгоритма и решаемой задачи это всегда предмет для исследований.

Полученные результаты подтверждают общие соображения о влиянии на эффективность параллельного логического вывода двух взаимосвязанных параметров: степени распараллеливания (количество используемых узлов и процессоров кластера) и накладных расходов, возникающих из-за необходимости управления процессами и обменом данных между узлами.

Эксперименты это наглядно иллюстрируют. Так с увеличением сложности задач (Рис. 4 и 5) происходит почти линейное уменьшение времени логического вывода с увеличением количества узлов. Однако, начиная с некоторого количества узлов, производная этого уменьшения стремится к нулю по указанным выше причинам: уменьшение «сложности» процессов на узлах с увеличением их количества и, как следствие, увеличение обменных взаимодействий между узлами и времени, затрачиваемого на управление.

Для менее сложных задач (Рис. 2, 3, 6, 7) при увеличении количества узлов больше некоторой оптимальной величины по этой причине происходит заметное ухудшение показателей критериев эффективности.

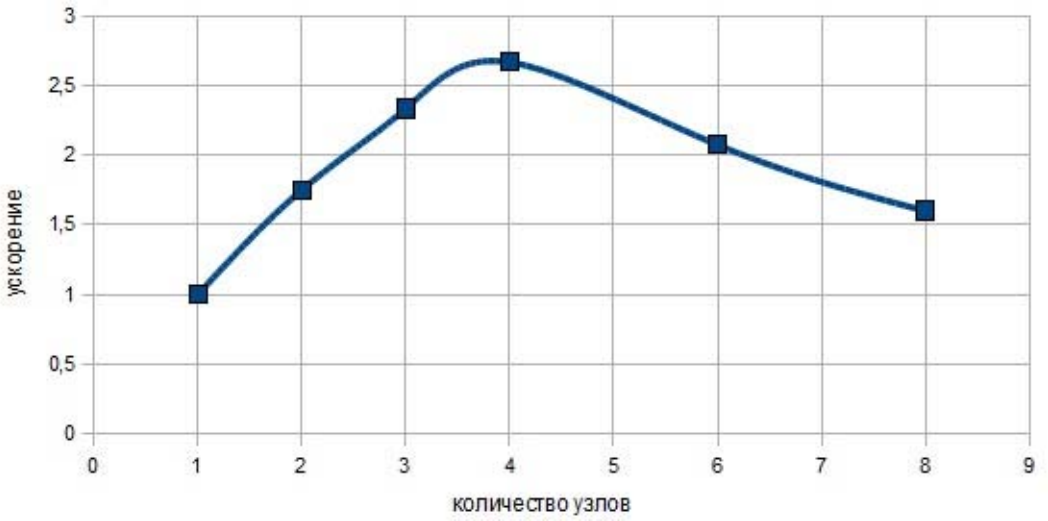

Рис. 3. Зависимость ускорения алгоритма от числа узлов кластера для задачи Ханойские башни, 5 колец (на каждом узле выполняется по 4 нити)

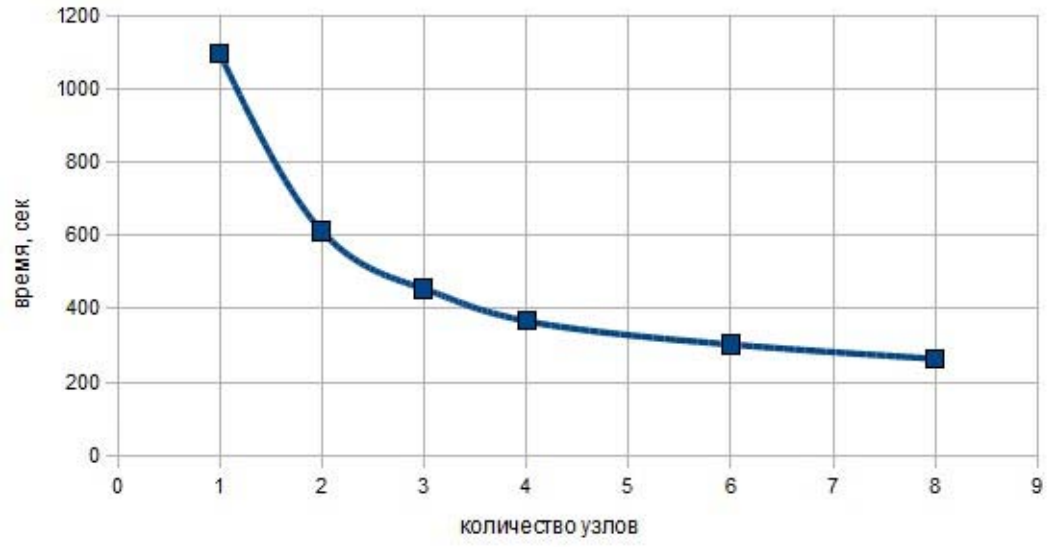

Рис. 4. Зависимость времени выполнения алгоритма от числа узлов кластера для задачи Ханойские башни, 6 колец (на каждом узле выполняется по 4 нити)

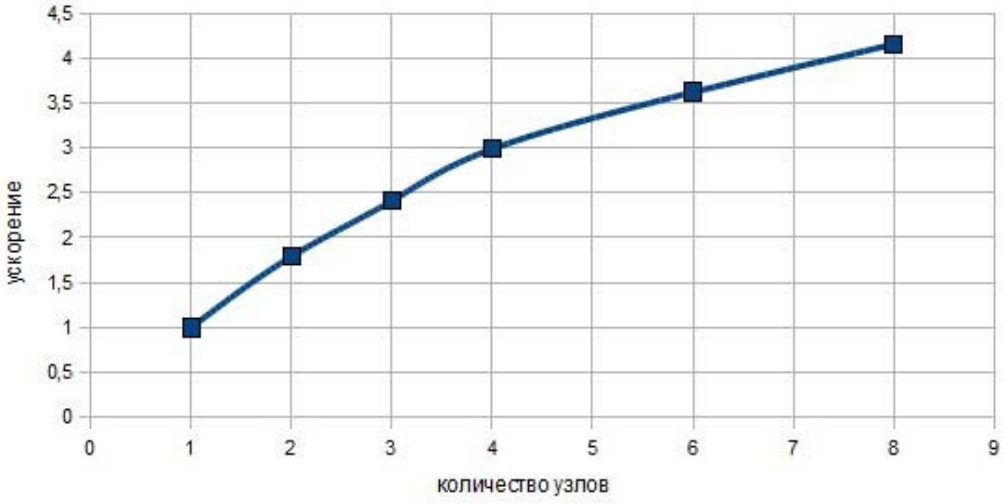

Рис. 5. Зависимость ускорения алгоритма от числа узлов кластера для задачи Ханойские башни, 6 колец (на каждом узле выполняется по 4 нити)

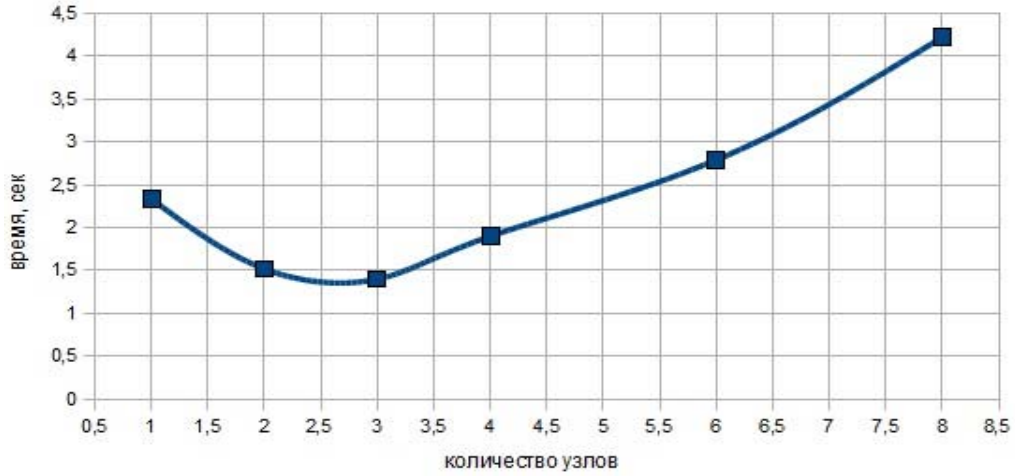

Рис. 6. Зависимость времени выполнения алгоритма от числа узлов кластера для задачи Steamroller (на каждом узле выполняется по 4 нити)

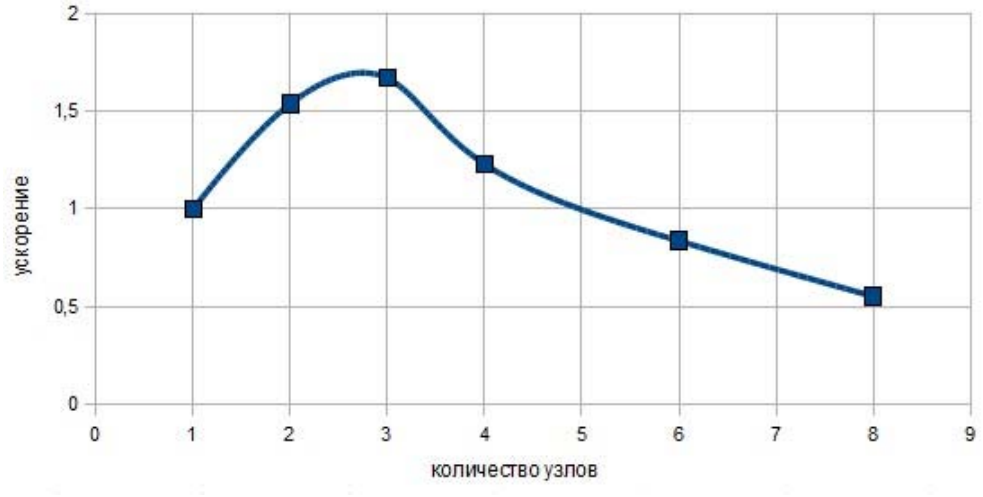

Рис. 7. Зависимость ускорения алгоритма от числа узлов кластера для задачи Steamroller (на каждом узле выполняется по 4 нити)

Заключение

В настоящей статье были рассмотрены способы повышения эффективности решения задачи логического вывода путем применения алгоритмов распараллеливания, основанных на принципе резолюции, на многоядерных и кластерных компьютерных системах. Представлены результаты исследований алгоритмов и предложены управляющие эвристики для убыстрения вычислений.

Выполненные исследования эффективности алгоритмов параллельного логического вывода на многопроцессорных компьютерных системах показывают, что распараллеливание может приводить к существенному сокращению времени вывода. Однако для этого требуется большая предварительная работа по выбору степени распараллеливания, необходимого для этого количества узлов и процессов, целого ряда других параметров, таких как квант времени при обмене данными между узлами, уменьшение влияния или отказ от использования семафоров и другие.

Очевидно, более глубокого исследования требуют рассмотренные управляющие эвристики, используемые для повышения эффективности параллельного вывода, а также способы начального разбиения множества дизъюнктов на подмножества, распределяемые между параллельными процессами на начальном этапе выполнения алгоритмов.

Литература

1. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. М.: Наука. 1983.
2. Кутепов В.П., Фальк В.Н. Теория направленных отношений и логика. Известия РАН. Теория и системы управления. №5. 2000.
3. Kalas A.C., Kowalski R.A., Toni F.The role of abduction in logic programming. Handbook of logic in artificial intelligence and Logic Programming. Oxford University press. 1998. P. 235-324.
4. Bonacina Maria Paola. A taxonomy of parallel strategies for deduction. AMAI. 2000. P. 1-36.
5. Вагин В.Н., ГоловинаЕ.Ю., Загорянская А.А., Фомина М.В. Достоверный и правдоподобный вывод в интеллектуальных системах. М.: ФИЗМАТЛИТ. 2004.
6. Кутепов В.П., Фальк В.Н. Формы, языки представления, критерии и параметры сложности параллелизма. Программные продукты и системы. 2010. №3. С. 16-26.
7. Кутепов В.П. Интеллектуальное управление процессами и загруженностью в вычислительных системах. Известия РАН. Теория и системы управления. 2007. №5. С. 58-73.
8. Кутепов В.П., Вагин В.Н., Хотимчук К.Ю. Параллельный логический вывод на многоядерном компьютере (в печати).
9. Кутепов В.П., Кумачев М.М. Параллельная логический вывод на кластерных системах. Научный сервис в сети Интернет: суперкомпьютерные центры и задачи. Труды Международной суперкомпьютерной конференции (20-25 сентября 2010 г., г.Новороссийск). Доклад. М.: Изд-во МГУ. 2010.
10. Зарецкий Д.С. Параллельная реализация принципа резолюции с использованием управляющих эвристик. Научный сервис в сети Интернет: суперкомпьютерные центры и задачи. Труды Международной суперкомпьютерной конференции (20-25 сентября 2010 г., г. Новороссийск). Доклад. М.: Изд-во МГУ. 2010.

Кутепов Виталий Павлович. Заместитель заведующего кафедрой Московского энергетического института (Технический университета). Окончил Московский энергетический институт в 1961 году. Доктор технических наук, профессор. Область научных интересов: компьютерные системы, параллельные вычисления. E-mail: kutepov@appmat.ru

Кумачев Михаил Михайлович. Младший научный сотрудник Московского энергетического института. Окончил Московский энергетический институт в 2008 году. Область научных интересов: искусственный интеллект , параллельные вычисления. E-mail: mikhail.kumachev@gmail.com