УДК 519.7, 519.17

РЕШЕНИЕ КОМБИНАТОРНЫХ ЗАДАЧ ЛОГИЧЕСКОГО ПРОЕКТИРОВАНИЯ И ЗАЩИТЫ ИНФОРМАЦИИ НА КЛАСТЕРНОМ КОМПЬЮТЕРЕ

Д.И. Черемисинов
Ведущий научный сотрудник ОИПИ НАНБ
кандидат технических наук, доцент

Л.Д. Черемисинова
Главный научный сотрудник ОИПИ НАНБ
доктор технических наук, профессор

Объединений институт проблем информатики НАНБ, Республика Беларусь
United Institute of Informatics Problems, National Academy of Sciences of Belarus, Republic of Belarus
E-mail: {cher, cld}@newman.bas-net.by

Д.И. Черемисинов
Ведущий научный сотрудник лаборатории логического проектирования ОИПИ НАНБ, доцент кафедры инженерной психологии и эргономики БГУИР, кандидат технических наук, доцент. Область научных интересов: логическое проектирование дискретных устройств управления, разработка моделей, методов и программных средств преобразования структурно-функциональных описаний СБИС.

Л.Д. Черемисинова
Главный научный сотрудник лаборатории логического проектирования ОИПИ НАНБ, профессор кафедры инженерной психологии и эргономики БГУИР, доктор технических наук, профессор. Область научных интересов: логические методы в приложении к вычислительной технике, теория дискретных управляющих устройств, автоматизация логического проектирования.

Аннотация. Рассматривается проблема подготовки программы для ее выполнения на многопроцессорной системе кластерного типа. В области программирования параллелизм является средством повышения эффективности вычислений. Эффективность параллельной программы традиционно связывают с достижением более высокой производительности по сравнению с ее последовательным вариантом. Рассматривается параллельные алгоритмы и программы решения комбинаторной задачи выполнимости КНФ (проверка ДНФ на тавтологию) для кластерного компьютера. Приводится сравнение эффективности решателя задачи выполнимости для кластера типа Беовульф и кластера Hadoop.

Ключевые слова: Параллельное программирование, Вычислительный кластер, Выполнимость КНФ, *MPI*, кластер Hadoop, MapReduce.

Введение – параллельные, распределенные и кластерные вычисления. Параллельные вычисления – это форма вычислений, при которой несколько вычислений в ходе взаимодействующих вычислительных процессов выполняются в перекрывающиеся периоды времени – одновременно. Вычислительный процесс может быть реализован в виде процесса операционной системы, либо же вычислительные процессы могут представлять собой набор потоков выполнения внутри одного процесса ОС. В то время как распределенная вычислительная система всегда использует параллельные вычисления, параллельные вычисления не обязательно требуют распределенной вычислительной системы. В кластерных вычислениях ресурсы распределенной компьютерной сети объединяются для обслуживания

одного пользователя или задачи. Вычислительный кластер представляет собой вычислительную сеть, которая действует как единая система. Современные кластерные компьютеры относятся к классу МКМД (множественный поток команд – множественный поток данных) или MIMD (Multiple Instruction – Multiple Data) ЭВМ по классификации Флинна. Кластеры составляют подкласс МКМД – системы с индивидуальной памятью. Для кластеров типа Beowulf (application cluster), характерна архитектура из автономных вычислительных узлов, объединенных компьютерной сетью, и использование в качестве операционной системы узла Linux. Головной (управляющий) узел (front-end node) управляет всем кластером и является файл-сервером для вычислительных узлов. Он также является консолью кластера и шлюзом во внешнюю сеть. Большие системы Беовульф имеют более одного управляющего узла.

Кластеры являются наиболее популярным типом многопроцессорной вычислительной системы (мультипроцессора). Способность мультипроцессора выполнять работу, пропорциональную числу процессоров, называется масштабируемостью. К сожалению, непосредственно измерить масштабируемость трудно, поэтому используется некоторая другая метрика. Общепринята оценка масштабируемости ускорением вычислений, поскольку ускорение вычислений на кластере можно узнать, измеряя время выполнения программ.

Ускорение (*speedup*), получаемое при использовании параллельного алгоритма для p процессоров, по сравнению с последовательным вариантом выполнения вычислений определяется величиной

$$S_p=T_1/T_p,$$

т.е. как отношение времени решения задач на однопроцессорной ЭВМ *лучшим из доступных алгоритмом* к времени решения той же задачи на мультипроцессоре *лучшим из доступных параллельным алгоритмом*. *Эффективность* использования процессоров в мультипроцессоре определяется соотношением $E_p=S_p/p$. Величина эффективности E_p определяет среднюю долю времени выполнения алгоритма, в течение которой процессоры реально задействованы для решения задачи. Если величины вычислительной работы для одного процессора и мультипроцессора одинаковы, то в случае, когда $S_p=p$ и $E_p=1$ эффективность будет наилучшей. Этот случай называется линейным ускорением. Ясно, что чем больше ускорение, тем лучше архитектура мультипроцессора.

Большинство программ для кластерных компьютеров используют для обмена данными между процессами метод рассылки сообщений. Основные идеи этого метода были предложены еще в конце 1960-ых, а в начале 1990-ых использовались несколько различных и несовместимых систем рассылки сообщений. Нужда в обеспечении мобильности и стандартизации привели к созданию MPI (Message Passing Interface) [1]. Программный интерфейс MPI представляет собой описание точного формата вызовов подпрограмм и смысла этих подпрограмм, составляющих библиотеку функций для использования при программировании на языках C или C++. Реализации MPI основывается на открытом стандарте, разработанным открытым образом (общедоступен не только сам стандарт, комментарии-рекомендации к нему, но и рабочие документы MPI-Форума). MPI предоставляет собой средство взаимодействия процессов, т. е. вычислений выполняющихся в разных адресных пространствах.

Во время выполнения MPI-программа сразу запускается на фиксированном числе вычислительных узлов кластера. Это число является параметром программы, его максимальная величина определяется размером машины (числом вычислительных узлов кластера). В MPI-программе есть выделенный процесс (соответствующий узел кластера имеет виртуальный идентификатор 0), который и обеспечивает прием параметров задачи и выдачу решения (рисунок 1).

Комбинаторные задачи логического проектирования и защиты информации. Проблема использования потенциала кластерных компьютеров семейства «СКИФ» [2] и средств для автоматизации их программирования – Т-системы (системы автоматического динамического распараллеливания программ) в области проектирования цифровых микросхем и защиты информации имеет существенное значение в повышении доли информационных технологий в экономике. Потребность в решении комбинаторных задач и, следовательно, в разработке комбинаторно-логических алгоритмов возникает во многих областях, в том числе таких, как автоматизированное проектирование, создание систем искусственного интеллекта, разработка сетей связи и криптографии. Практические задачи проектирования и защиты информации имеют параметры сложности, превосходящие возможности традиционной вычислительной техники.

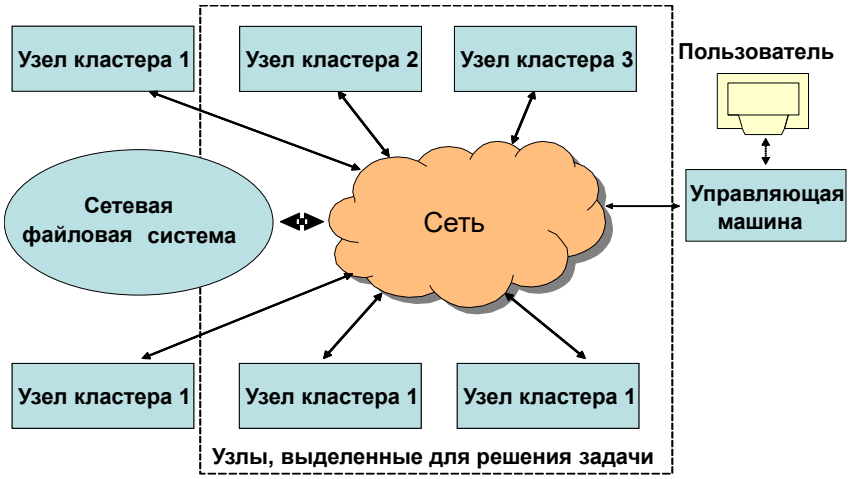

Рисунок 1. Конфигурация аппаратуры привыполнении MPI-программы

Под комбинаторно-логическими задачами подразумеваются *перечислительные* и *поисковые* задачи на конечных множествах, элементами которых служат объекты, представляющие собой комбинации элементов других конечных множеств – сочетания, перестановки, разбиения, покрытия, решения систем логических уравнений и т.п. Отличительной особенностью комбинаторно-логических задач [3] является то, что основными объектами преобразований для них являются не числа, как для задач численного анализа, а *логические объекты* (логические переменные, булевы и *k*-значные векторы, булевы и *k*-значные матрицы, графы и т.д.). Другой важной особенностью задач рассматриваемого класса является то, что объемы перерабатываемой информации при поиске решений относительно невелики, а сами процессы переработки достаточно сложны. Почти все комбинаторно-логические алгоритмы имеют экспоненциальную сложность. Вместе с тем на практике необходимо уметь решать комбинаторно-логические задачи достаточно больших размерностей.

Анализ применимости Т-системы и использования метода автоматического распараллеливания для основных комбинаторно-логических алгоритмов показывает бесперспективность этого метода построения кластерных программ. Все эти задачи конечны, т.е. для них существует тривиальный алгоритм перебора по дереву возможных решений, который изначально может быть реализован параллельно, следовательно, поддается автоматическому распараллеливанию. Методы распараллеливания обхода дерева поиска рассмотрены в [4]. Однако, логико-комбинаторные задачи сложности, встречающейся в практике, не могут

быть решены методом исчерпывающего обхода дерева, и в ходе поиска используют эвристики для исключения большей части дерева. Например, задача нахождения кратчайшего пути в графе, содержащем 100 вершин, методом полного обхода пространства решений требует 25×10^{136} лет работы ЭВМ производительностью 12.5 Tera-FLOPS (типа кластерного компьютера СКИФ-1000). В то время как эвристическим методом она решается за доли секунды на одном процессоре. Следует заметить, что оба метода дают одинаковые решения конкретной задачи поиска кратчайшего пути в графе меньшей размерности, но один из них может решить эту задачу для размерности большей, чем 100, а применимость другого невозможна и для 100 вершин.

Тривиальные параллельные алгоритмы, в которых распараллеливание достигается использованием для просмотра дерева поиска решений нужного количества процессоров, описаны в литературе для всех задач проектирования СБИС. Так в книге [5], описаны параллельные алгоритмы для размещения и трассировки, проверки соблюдения правил проектирования, логического синтеза, генерации тестов, моделирования ошибок в схемах и моделирования на поведенческом уровне. Все эти параллельные алгоритмы оказались не эффективными. Таким образом, Т-система неприменима не потому, что она плоха, а потому что проблемная область не допускает автоматического распараллеливания. Сам метод на основе использования последовательных алгоритмов не позволяет получить именно эффективный параллельный алгоритм.

Закон Амдала. Линейное ускорение считается теоретическим пределом эффективности вычислений на мультипроцессоре. Обоснованием этого свойства параллельной обработки служит закон Амдала [6]. Ключевым моментом закона Амдала является доля последовательной обработки относительно полного времени выполнения параллельной программы, когда в параллельной программе используется единственный процессор. Эта доля независима от числа процессоров. Пусть f – доля таких вычислений в общем объеме, $0 < f < 1$. Максимальное ускорение S, достижимое на вычислительной системе из p процессоров, можно оценить при помощи следующей формулы:

$$S \le \frac{1}{(f + (1-f)P^{(-1)})}.$$

Из закона Амдала следует, что всегда при любом сколь угодно большом числе процессоров, независимо от качества реализации параллельной части кода, $K < 1/f$. Таким образом, если например половина программы выполнятся всеми процессорами, то больше чем в 2 раза ускорить решение в принципе невозможно ни на какой параллельной вычислительной системе. Кроме того, из закона Амдала следует, что $f \le \frac{P-K}{K(P-1)}$, то есть если мы, например, хотим на 10 процессорах получить ускорение в 9 раз, то нам необходимо, чтобы 99 % кода программы выполнялось параллельно ($f \le 1.2$ %). Наличие даже небольших частей программы, выполняемых на всех процессорах, существенно снижает параллельную эффективность программы. Закон Амдала считается существенным теоретическим ограничением роста эффективности кластерного компьютера путем увеличения числа вычислительных узлов.

Кластер Apache Hadoop. Достижения в области цифровых датчиков, систем связи, вычислений и хранения данных позволили создавать огромные коллекции данных, представляющие огромную ценность для бизнеса, науки, правительств и общества. Огромные объемы данных требуют автоматического или полуавтоматического анализа – программных средств для выявления закономерностей, выявления аномалий и извлечения знаний. Задачи анализа «больших данных» требуют и огромных вычислений. Однако в отличии от традиционных сложных вычислений, сложность решения задач анализа «больших данных» не связана с алгоритмами, а определяется именно большим объемом данных.

Тривиальные для обычных научно-технических вычислений проблемы становятся трудно решаемыми для «больших данных». Так, хотя один терабайт данных может храниться на диске стоимостью всего 100 долларов, для передачи такого количества данных требуется один час или более в кластере типа Беовульф и примерно один день по обычному «высокоскоростному» Интернет-соединению.

Эти ограничения на скорость передачи обусловили появление нового типа вычислительных кластеров, состоящих из компьютеров, используемых в качестве ресурсов для хранения данных. В этих кластерах данные обрабатываются в местах их хранения, а передаются каналам связи результаты анализа гораздо меньшего объема. Программа для такого кластера на основе информации о том, на каких машинах расположены блоки данных, запускает на них же вычислительные процессы для анализа этих данных – «воркеры». Это позволяет выполнить большую часть вычислений локально, т.е. без передачи данных по сети. Кластерный фреймворк для создания программ такого типа называется MapReduce, а кластер, обеспечивающий работу этого фреймворка – Hadoop. Обычная конфигурация кластера Hadoop [7] состоит из одного управляющего узла – Job tracker, выполняющего процессы инициатора работ (TaskTracker) и NameNode, с помощью которого определяется местоположение данных, и набора рабочих машин, на каждой из которых одновременно выполняется сервис данных (DataNode) и воркер (TaskTracker).

Кластерный фреймворк MapReduce использует оригинальную модель программирования [8] для обработки и генерации больших массивов данных. Интерфейс фреймворка содержит всего две функции: Map и Reduce, которые должен задать программист. Функция Map по исходной паре ключ/значение генерирует набор промежуточных пар ключ/значение, а функция Reduce объединяет все промежуточные значения, связанные с одним и тем же промежуточным ключом. Многие реальные задачи «больших данных» выражаются этой моделью. Метрика масштабируемости «ускорение» непосредственно не применима к кластеру Hadoop потому, что работа фреймворка MapReduce не может быть выполнена на одном процессоре. Так как в программе фреймворка MapReduce нет последовательно выполняемых частей, ее масштабируемость не ограничено по закону Амдала.

Параллельные вычислений при автоматизированном проектировании СБИС. Известно, что к решению логических уравнений в конъюнктивной или дизъюнктивной нормальной форме (КНФ или ДНФ) сводится решение подавляющего большинства задач верификации [9] и синтеза цифровых СБИС, эта операция входит в глубинные циклы алгоритмов решения этих задач.

Логическими уравнениями называются выражения вида $F(X) = 1$ и $F(X) = 0$, где $F(X)$ – формула булевой алгебры, задающая некоторую связь между переменными $xi \in X$ и выделяющая из множества всех наборов значений этих переменных те, на которых формула F принимает значение 1 (или 0) [9]. Эти наборы значений переменных называются корнями (или решениями) уравнения. Задача поиска решения системы логических уравнений сводится к задаче поиска решения одного логического уравнений: $F1 \wedge F2 \wedge \ldots \wedge Fm = 1$ (или $F1 \vee F2 \vee \ldots \vee Fm = 0$). Одним из важнейших классов логических уравнений являются уравнения, в которых формула $F(x1, x1, \ldots, xn)$ задана в виде КНФ К или ДНФ D: $K = 1$, $D = 0$ (в силу двойственности $K = \overline{D}$).

Широкий класс практически значимых задач, связанных с управлением и обработкой информации в дискретных системах, а также задач синтеза и верификации в микроэлектронике и других, сводится к проблеме булевой выполнимости КНФ [10] или проверке ДНФ на тавтологию [9]. Эти ключевые операции часто фигурируют во внутренних циклах алгоритмов, решающих эти задачи. Проблема выполнимости КНФ *К* состоит в проверке существования корня уравнения $K = 1$, т.е. такого набора значений переменных, который обращает в 1 КНФ *К* (как и каждый дизъюнкт) и называется выполняющим. Задачу выполнимости КНФ называют в литературе SAT-задачей (propositional satisfiability) [10].

Свойства параллельных алгоритмов. Алгоритмы решения комбинаторно-логических задач по способу декомпозиции исходной задачи на частные подзадачи, решаемые подчиненными процессорами, и по правилам формирования из частичных решений итогового решения исходной задачи распадаются на два следующих класса [11].

В алгоритмах совместного решения (класс J в [11]) исходная задача решаются совместно (jointly) N-1 процессорами. Каждый из них решает свою частную задачу над своими исходными данными до конца, а корневой процессор собирает результаты частичных решений и формирует из них итоговое решение общей задачи.

Время выполнения алгоритма класса J определяется временем работы процессора, выполняющего наибольший объем работы. Если имеется несбалансированность загрузки процессоров, то часть процессоров вынуждена простаивать, пока остальные заканчивают свою вычислительную работу. Поскольку время решения задачи алгоритмом из класса J определяется временем «самого медленного» (последнего из закончивших работу) процессора, то при декомпозиции исходной задачи на частные подзадачи следует стремиться к наиболее равномерному распределению нагрузки между процессорами. Если доля последовательных вычислений f стремится к 0 при росте числа процессоров, то ускорение будет максимальным S = n (ускорение пропорционально числу процессоров – линейное).

Кроме вычислений при работе кластерной программы время тратиться на обмен данными. Межпроцессорная передача сообщений характеризуется скоростью передачи и временем задержки (latency), необходимым для подготовки аппаратуры к передаче сообщения. При распределении работы по обходу дерева по типу лучших последовательных алгоритмов оказывается необходимым, чтобы процессы в узлах кластера обменивались почти всем содержимым используемой памяти. В этом случае отношение времени обмена данными к времени вычислений оказывается больше единицы, что ведет не к ускорению, а к замедлению вычислений по сравнению с однопроцессорным компьютером.

Алгоритмы, работающие в режиме соревнования (класс C в [11]), отличаются от алгоритмов класса J тем, что построены так, что решение любой частной задачи, полученное любым из процессоров в ходе соревнования (competition) по времени с другими, является решением общей задачи. Когда какой-либо из процессоров находит решение, все остальные должны прекратить работу, т.е. время решения задачи класса C определяется «самым быстрым» процессором.

К классу *C* относятся, например, задачи, решаемые *конкурирующими* алгоритмами, когда подчиненные процессоры решают одну и ту же исходную задачу, каждый своим алгоритмом, соревнуясь по времени. Алгоритмы класса *C* характеризуются тем, что их работа характеризуется в некоторых случаях «сверхлинейным» ускорением [12]. Эффект «сверхлинейного» ускорения состоит в том, что наблюдаемое ускорение больше числа процессоров: $S > n$. В алгоритмах класса *C* сверхлинейность объясняется удачным разбиением пространства поиска. Эффект «сверхлинейного» ускорения зависит от исходных данных, и по параметрам исходных данных его почти никогда невозможно предсказать.

В большинстве задач дискретной оптимизации доказательство точного решения можно получить только перебором. При нахождении решения происходит разбиение по некоторым формальным критериям множества решений на подмножества, затем делается попытка доказать, что в конкретном подмножестве не содержится оптимальных решений. Как правило, это делается с помощью процедуры граничной оценки подмножества, которая дает оценку наилучшего возможного решения подмножества и сравнение полученного значения с рекордом – решением, лучшим из известных. В случае, если удастся доказать, что подмножество не содержит оптимальных решений, оно удаляется из дальнейшего рассмотрения, «отсеивается». В противном случае само подмножество разбивается и для каждого нового подмножества применяется процедура оценки. Если в подмножестве остается

только одно решение и оно лучше рекорда, последнее объявляется новым рекордом. Ветвление продолжается до тех пор, пока все подмножества не будут отсеяны. Рекорд, оставшийся после отсева всех подмножеств, является оптимальным решением. Здесь часть алгоритма, связанная с нахождением рекорда, является алгоритмом класса C, а доказательство оптимальности рекорда – алгоритмом класса J. Общее поведение будет зависеть от распределения объема работы между частями в зависимости от конкретных данных. Если рекорд будет обнаружен сразу, ускорение будет меньше линейного, в противном случае возможно сверхлинейное ускорение. Предсказать характер поведения по данным бывает трудно.

Параллельные решатели задачи выполнимости КНФ. Алгоритмы решения задачи выполнимости КНФ интенсивно исследуются из-за важности практических применений. Регулярно проводятся соревнования программ (SAT solver competitions), на которых выявляются при решении искусственных и реальных задач лучшие программы. Из-за практической важности решения задачи выполнимости большой размерности интенсивно исследуются и параллельные алгоритмы ее решения. В [13] описан параллельный решатель задачи выполнимости КНФ, испытанный на кластерах семейства «СКИФ». Эта MPI-программа применяется в составе разработанного в лаборатории логического проектирования ОИПИ НАН Беларуси программного комплекса ЭЛС «Энергосберегающий логический синтез» [14]. Комплекс ЭЛС предназначен для автоматизации проектирования многоуровневых логических схем из библиотечных элементов заказных СБИС, выполненных по КМОП технологии, с оптимизацией схем по площади и энергопотреблению.

Одним из ключевых этапов проектирования в рамках программного комплекса ЭЛС является верификация проектных решений на всех шагах проектирования. Программа верификации работает для любых пар проектных состояний (функциональных или структурных описаний), которые могут принадлежать одному и тому же проекту (или разным).

В рамках комплекса реализованы процедуры верификации на основе моделирования описания верифицируемого устройства на области задания исходной спецификации, на основе формального доказательства функциональной идентичности (или реализуемости, в случае описаний с функциональной неопределенностью) проектов или на основе гибридных подходов (рис. 2). Формальная верификация основана на сведении задачи к проверке выполнимости некоторой КНФ K, которая отражает структуру сравниваемых описаний и выполнимость которой (существование корня уравнения в форме K = 1) свидетельствует о нарушении эквивалентности сравниваемых описаний (или реализуемости). Как правило, анализируемая на выполнимость КНФ имеет значительный размер, что сказывается на быстродействии программы верификации в целом, именно поэтому эта задача решается с привлечением средств кластерного компьютера.

В основе решателя задачи выполнимости лежит идея [15] гибридного распределения нагрузки между подчиненными процессами, когда сначала вся нагрузка делится, а затем освободившиеся процессоры подключаются для решения подзадач, решаемых загруженными процессорами. В матричной интерпретации задача сводится к задаче анализа на выполнимость троичной матрицы $\boldsymbol{T}$, строки которой задают дизъюнкты исходной КНФ. Проверить троичную матрицу $\boldsymbol{T}$ на выполнимость означает найти троичный вектор $\boldsymbol{t}$, который обращает в 1 каждый дизъюнкт. Или убедиться в том, что такого вектора не существует (когда матрица $\boldsymbol{T}$ не выполнима, т. е. $K \equiv 0$).

При доказательстве вырожденности матрицы решается задача класса J, время решения которой определяется «самым медленным» процессом, доказавшим, что последний из непроверенных еще миноров также оказался вырожденным. При обнаружении вектора, ортогонального всем строкам матрицы, решается задача из класса C, время решения которой определяется «самым быстрым» процессом, первым обнаружившим такой вектор.

Проведенные эксперименты показали, что k-кратное ускорение параллельной проверки троичной матрицы на вырожденность с использованием k подчиненных процессов, по сравнению с однопроцессорным вариантом, вполне достижимо при надлежащем распределении нагрузки между процессами.

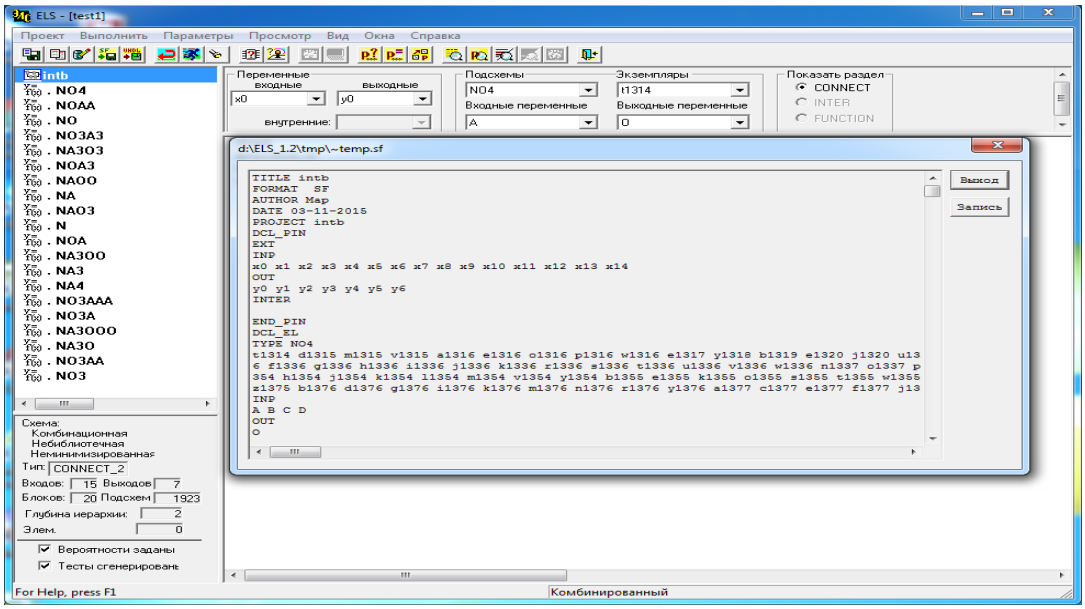

Рисунок 2. Пример экрана системы ЭЛС в состоянии выполнения операции верификации

Решатель задачи выполнимости реализован MPI-программой для кластерного компьютера. Исходными данными для программы служит описание КНФ в текстовом виде или в формате DIMACS. Формат DIMACS представляет собой стандартный интерфейс для SAT-решателей и используется для ввода заданий в известных соревнованиях программ для решения проблемы SAT. В данном случае он используется, так как позволяет компактно представлять редкие троичные матрицы. Результатом выполнения программы является текст, содержащий режим и длительность решения, а также выполняющий вектор, если КНФ выполнима.

MPI-программа решателя задачи выполнимости использована в среде программного комплекса ЭЛС для верификации ряда структурных описаний, возникающих в процессе проектирования. О размерности решаемых задач можно судить по следующему примеру: верифицируемые схемы состояли из 1923 элементов КМОП библиотеки, КНФ разрешения в этом случае зависела от 13103 переменных и содержала 38635 дизъюнктов. Общее ограничение программы «Вычислитель» на предельный размер КНФ составляют ~10^9 дизъюнктов.

Следует заметить, что мультипроцессорные структуры с индивидуальной памятью (к каким относится семейство СКИФ) ориентированы в основном на задачи класса *J* (реализованы эффективные операции по сбору данных корневым процессором у всех подчиненных, имеются операции синхронизации процессоров и пр.). В то же время при решении задач класса *C* для прерывания работы активных процессоров в тот момент, когда один из них нашел решение, требуется посылка сообщения. Обнаружение такого сообщения выполняют все процессоры, что по закону Амдала снижает параллельную эффективность программы.

В статье [16] представлены достижения в применении подхода MapReduce для решения задач оптимизации с использованием Hadoop в системах облачных вычислений. Предложены модели MapReduce для решения 3-SAT, хорошо известной версии проблемы выполнимости КНФ. В работе описывается несколько вариантов решателя 3-SAT на платформе MapReduce. Для испытаний использовался кластер из серверов AMD Opteron 6272 2.09GHz (24 ядра с 72GB оперативной памяти каждый). Решатель, дающий точное решение проблемы 3-SAT на этом кластере решал задачу выполнимости КНФ с 100 переменными и 200 дизъюнктами более 2 суток. Для сравнения параллельный решатель из [13] на кластере с 8 ядрами решает задачу выполнимости с 13103 переменными и 38635 дизъюнктами за секунды.

Заключение. Разработка комбинаторно-логических алгоритмов для параллельных компьютеров является высокоинтеллектуальной деятельностью, в которой успехи довольно редки. Показательным примером может служить задача выполнимости конъюнктивной нормальной формы (КНФ) – propositional satisfiability (SAT). Алгоритмы ее решения интенсивно исследуются из-за важности практических применений. Решение этой задачи требуется при проектировании СБИС, в задачах искусственного интеллекта, составлении расписаний и т.д.

Следует заметить, что мультипроцессорные структуры с индивидуальной памятью (к каким относится семейство СКИФ) ориентированы на взаимодействие путем рассылки сообщений, т.е. на задачи класса *J* (реализованы эффективные операции по сбору данных корневым процессором у всех подчиненных, имеются операции синхронизации процессоров и пр.). При решении задач класса *C* отсутствует эффективное средство прерывания работы активных процессоров в тот момент, когда один из них нашел решение.

Чрезвычайно низкие показатели решателя задачи выполнимости, использующая модель MapReduce, связана с необходимостью использовать для межпроцессных обменов сообщений большого объема. Это совершенно не подходящий тип параллельных алгоритмов для кластеров Hadoop. Функциональный стиль модели MapReduce, как заявляют ее авторы [8], обеспечивает автоматическое параллельное выполнение простых локальных вычислений над большими объемами распределенных данных. В задачи выполнимости исходные данные относительно не велики по объему, а вычисления трудоемки.

Литература

[1] Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI. – Мн.: БГУ, 2002. – 323 с.

[2] Абрамов С.М., Парамонов Н.Н., Анищенко В.В., Абламейко С.В. Принципы построения суперкомпьютеров семейства «СКИФ» и их реализация // Информатика . – 2004. – № 1. – С. 89–106.

[3] Закревский, А. Д. Алгоритмы синтеза дискретных автоматов / А. Д.Закревский – М.: Наука, 1971. – 512 с.

[4] Тимошевская Н. Е. Параллельные методы обхода дерева // Математическое моделирование. 2004. Т. 16. №1. С. 105-114.

[5] Banerjee, P. Parallel Algorithms For VLSI Computer-Aided Design / P.Banerjee – Prentice Hall, Englewoods Cliffs, NJ, 1994. – 699 p.

[6] Amdahl, G. Validaty of the Single Processor Approach to Achieving Large Scale Computing / G. Amdahl // Capabilities. AFIPS Conference Proceedings, V. 30, 1967. – P.483.

[7] Perera S., Gunarathne T. Hadoop MapReduce Cookbook / Perera S., Gunarathne T. –Packt Publishing, 2013. – 300 p.

[8] Dean J. Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters // Commun. ACM, v.51, n. 1, 2008. – P. 107-113.

[9] Закревский А.Д., Поттосин Ю.В., Черемисинова Л.Д. Логические основы проектирования дискретных устройств. М.: Физматлит, 2007. – 589 с.

[10] Ganai M., Gupta A. SAT-Based Scalable Formal Verification Solutions. New York: Springer-Verlag, 2007. – 338 p.

[11] Торопов Н.Р. Параллельные логико-комбинаторные вычисления в среде MPI / Н.Р. Торопов // Информатика. – 2005. – № 3. – С. 82–90.

[12] Черемисинов Д.И. Проектирование и анализ параллелизма в процессах и программах – Минск: Беларус. навука, 2011. –300 с.

[13] Черемисинов Д.И., Черемисинова Л.Д. Использование параллельных вычислений при автоматизированном проектировании СБИС // Проблемы разработки перспективных микро- и наноэлектронных систем - 2016. Сборник трудов / под общ. ред. академика РАН А.Л. Стемпковского. – М.: ИППМ РАН, 2016. Часть I. – С. 32-39.

[14] Бибило П.Н., Черемисинова Л.Д., Кардаш С.Н. и др. Автоматизация логического синтеза КМОП схем с пониженным энергопотреблением // Программная инженерия, 2013, № 8, с. 35–41.

[15] Торопов Н. Р. Параллельная проверка ДНФ на тавтологию // Информатика, 2005, № 2, с. 35–42.

[16] Barreto M., Nesmachnow S. Tchernykh A. Hybrid Algorithms for 3-SAT Optimisation Using MapReduce on Clouds // Int. J. Innov. Comput. Appl., 2018, v. 9, n. 1. – P. 44-64.

THE DECISION OF COMBINATORIAL PROBLEMS OF LOGICAL DESIGN AND INFORMATION PROTECTION ON A CLUSTER COMPUTER

D.I. CHEREMISINOV
Leading Researcher UIIP NANB Candidate of Technical Sciences, Associate Professor

L.D. CHEREMISINOVA
Chief Researcher, UIIP NASB doctor of technical sciences, professor

United Institute of Informatics Problems, National Academy of Sciences of Belarus, Republic of Belarus
E-mail: {cher, cld}@newman.bas-net.by

Abstract. The problem of preparing a program for its execution on a cluster-type multiprocessor system is considered. In the field of programming, parallelism is a means of increasing computation efficiency. The effectiveness of a parallel program is traditionally associated with the achievement of higher performance compared with its serial version. The parallel algorithms and programs for solving the combinatorial problem of CNF satisfiability for a cluster computer are considered. A comparison is made of the efficiency of the satisfiability solver for a Beowulf-type cluster and a Hadoop cluster.

Keywords: parallel programming, cluster computer, propositional satisfiability, *MPI*, Hadoop, MapReduce.