УДК 519.7, 519.17

Использование параллельных вычислений при автоматизированном проектировании СБИС

Д.И. Черемисинов, Л.Д. Черемисинова

Объединенный институт проблем информатики Национальной академии наук Беларуси,

cher@newman.bas-net.by, cld@newman.bas-net.by

***Аннотация* — Предлагается методика и программные средства, обеспечивающие совместимость по данным и организацию обмена данными между кластерным компьютером и системами автоматизации проектирования для решения трудоемких задач из области логического проектирования. Описывается технология и результаты прогонки примеров выполнения этапа верификации результатов проектирования в рамках системы автоматизации логического проектирования заказных КМОП СБИС при обеспечении удаленного решения задачи нахождения корней логических уравнений на базе высокопроизводительного кластерного компьютера.**

***Ключевые слова* — автоматизация проектирования, параллельные вычисления, кластерный компьютер, грид-система, верификация, выполнимость КНФ.**

I. Введение

Практически все задачи из области теории дискретных автоматов и логического проектирования носят комбинаторный характер и имеют экспоненциальную сложность. Отличительной особенностью логико-комбинаторных задач [1] является то, что основными объектами преобразований для них являются логические объекты (переменные, булевы и *k*-значные векторы и матрицы, графы и т. д.). Для задач такого класса характерно то, что объемы перерабатываемой информации при поиске решений относительно невелики, а сами процессы переработки достаточно сложны, так как зачастую сопряжены с необходимостью перебора и анализа значительного числа вариантов промежуточных решений. В силу этого становится проблематично решать такие задачи большой размерности за приемлемое время даже с использованием самых быстродействующих персональных компьютеров.

В последнее время при решении сложных вычислительных задач все чаще используют многопроцессорные вычислительные системы, наиболее популярным типом которых являются кластеры. В рамках программы «СКИФ» Союзного государства России и Беларуси создан кластерный суперкомпьютер семейства «СКИФ» кластерного типа [2].

Эффективное использование многопроцессорной техники предполагает разработку параллельных алгоритмов решения задач. При этом основной подход состоит в распараллеливании существующих последовательных алгоритмов решения задач. По сути, это предполагает разбиение процесса решения задачи на ряд независимых подпроцессов, которые выполняются параллельно (но каждый последовательным способом) на разных процессорах вычислительной системы. Этот подход хорошо работает, когда задача может быть разделена по данным, обрабатываемым каждым процессором, а общее решение складывается из частных решений. Гораздо сложнее осуществить распараллеливание комбинаторных задач, когда невозможно простое разделение по данным.

Проблема, однако, заключается в том, как надо разбивать задачу на подзадачи так, чтобы обеспечить наивысшую степень ускорения, которое зависит от многих факторов, в том числе от неравномерной загрузки процессоров системы, от накладных расходов на коммуникации, от конкуренции процессоров за доступ к разделяемым данным, от избыточности просматриваемой области поиска и др. Более того, при одном и том же способе распараллеливания эффективность параллельного алгоритма в значительной степени зависит и от решаемой задачи, и от архитектуры многопроцессорной вычислительной системы. Несмотря на множество работ по этой проблеме, эффективного ее решения на данный момент не найдено.

Известно, что к решению логических уравнений в конъюнктивной или дизъюнктивной нормальной форме (КНФ или ДНФ) сводится решение подавляющего большинства задач верификации [3] и синтеза цифровых СБИС, эта операция входит в глубинные циклы алгоритмов решения этих задач. Эффективность проектирования можно существенно повысить за счет распараллеливания алгоритма решения логических уравнений для выполнения на кластерных системах.

В настоящей работе предлагается сервис грид-системы для выполнения трудоемких расчетов при

проектировании СБИС, в частности для решения логических уравнений. Сервисный подход для управления работой программой кластерного компьютера привлекателен тем, что ориентирован на использование открытых, универсальных протоколов и интерфейсов. Предлагается методика и программные средства, обеспечивающие совместимость по данным и организацию обмена данными между суперкомпьютером кластерного типа и системой автоматизации проектирования для решения трудоемких задач. Описывается технология (и результаты прогонки тестовых примеров) выполнения этапа верификации результатов проектирования в рамках системы автоматизации логического проектирования заказных КМОП СБИС [4] на основе удаленного решения задачи нахождения корней логических уравнений на базе суперкомпьютера.

II. Организация вычислений на суперкомпьютере кластерного типа

Суперкомпьютер семейства СКИФ [2], как и большинство кластерных компьютеров, имеет архитектуру Беовульф [5] на основе автономных вычислительных узлов, соединенных собственной компьютерной сетью, и использования Linux в качестве операционной системы. Всем кластером управляет головной узел (front-end node), он же является файл-сервером для вычислительных узлов, консолью кластера и шлюзом во внешнюю сеть (на рис. 1 это управляющая машина). Остальные узлы кластера используются только для выполнения программного кода, а для обмена данными между выполняющимися на них процессами используется метод рассылки сообщений – MPI [6].

Рис. 1. Конфигурация аппаратуры при выполнении MPI-программы

Исполнимый код MPI-программы копируется в память каждого из вычислительных узлов кластера, выделенных управляющей машиной для ее выполнения. Все они включаются в работу одновременно и работают параллельно. Множество вычислительных узлов, занятых MPI-программой, меняется от запуска к запуску. В MPI-программе есть выделенный (корневой) процесс (первый из назначенных узлов для ее выполнения), который обеспечивает прием параметров задачи (исходных данных) и выдачу решения, которые хранятся затем в сетевой файловой системе, доступной для всех вычислительных узлов. Каждый из запущенных процессов «распознает» и выполняет свои участки программы, при необходимости обмениваясь сообщениями с другими процессами. В настоящее время MPI является основной технологией программирования для многопроцессорных систем с распределенной памятью.

Пользователь запускает MPI-программу через управляющую машину кластера, которая доступна по сети через протокол Интернет. Входные данные программы должны быть заранее размещены в файловой системе, туда же MPI-программой помещается результат, доступный после окончания ее работы.

Для запуска заданий на кластере используется система управления заданиями PBS (Portable Batch System) [7] – программа, работающая на управляющей машине кластера (рис. 2). PBS позволяет повысить эффективность загрузки вычислительных ресурсов, уменьшая их простой. Запуск MPI-программы выполняется специальной командой PBS с помощью запускающего скрипта (командного файла), в котором указывается запускаемая программа и требуемые ресурсы (число процессоров, время решения и т.п.).

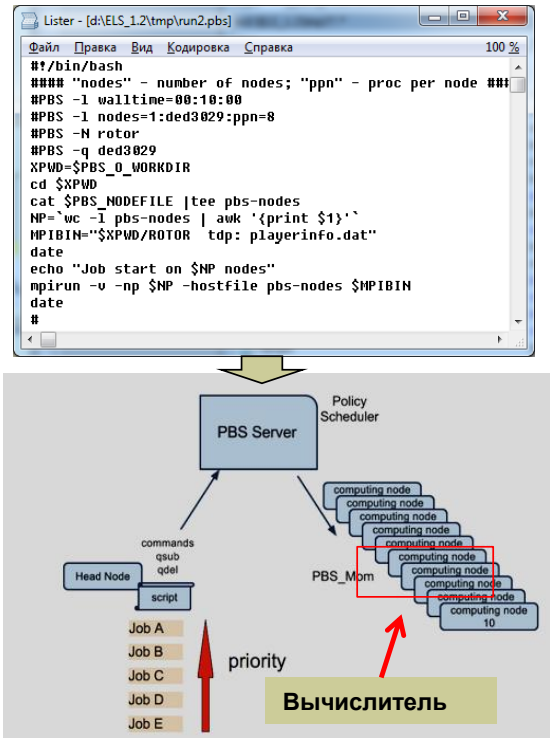

Рис. 2. Запуск MPI-программы

Для связи с окружающей средой управляющая машина кластера имеет OpenSSH сервер [8] (открытая безопасная оболочка) – программу, предоставляющую шифрование сеансов связи по компьютерным сетям с использованием протокола SSH. OpenSSH сервер предоставляет сервис для зашифрованного удаленного управления управляющей машиной кластера и может

аутентифицировать пользователей, используя встроенные механизмы аутентификации: публичные ключи и клавиатурный ввод (пароли и запрос-ответ).

III. Программа решения логических уравнений на кластерном компьютере

Логическими уравнениями называются выражения вида $F(X) = 1$ и $F(X) = 0$, где $F(X)$ – формула булевой алгебры, задающая некоторую связь между переменными $x_i \in X$ и выделяющая из множества всех наборов значений этих переменных те, при которых формула F принимает значение 1 (или 0) [1]. Эти наборы значений переменных называются корнями (или решениями) уравнения. Задача поиска решения системы логических уравнений сводится к задаче поиска решения одного логического уравнения: $F_1 \wedge F_2 \wedge \ldots \wedge F_m = 1$ (или $F_1 \vee F_2 \vee \ldots \vee F_m = 0$). Одним из важнейших классов логических уравнений являются уравнения, в которых формула $F(X)$ задана в виде КНФ K или ДНФ D: $K = 1$, $D = 0$ (в силу двойственности $K = \neg D$).

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 [3] или проверке ДНФ на тавтологию [1]. Эти ключевые операции часто фигурируют во внутренних циклах алгоритмов, решающих эти задачи. Проблема выполнимости КНФ K состоит в проверке существования корня уравнения $K = 1$, т.е. такого набора значений переменных, который обращает в 1 КНФ K (как и каждый дизъюнкт) и называется выполняющим. Задачу выполнимости КНФ называют в литературе SAT-задачей [3].

Построение SAT-решателей, работающих в грид-системах, является сравнительно новым направлением, активно развивающимся последние годы [9-17]. Распараллеливание алгоритмов решения SAT-задач может производиться по двум направлениям. Первое направление основывается на организации динамического распределения нагрузки процессоров. Распараллеливание осуществляется на уровне базового алгоритма SAT-решателя и предполагает интенсивный обмен накапливаемыми конфликтными ограничениями между вычислительными узлами. Данный подход допускает эффективную реализацию лишь в средах с быстрым межпроцессорным взаимодействием [10, 11].

В основе второго направления лежит статическое распределение нагрузки процессоров [13-16]. В этом случае осуществляется декомпозиция исходной SAT-задачи на семейство непересекающихся подзадач, каждая из которых обрабатывается SAT-решателем на отдельном вычислительном узле вычислительной системы. Этот подход не требует интенсивного межпроцессорного взаимодействия и, следовательно, хорошо подходит для реализации в грид-системах.

Методы распараллеливания задач поиска корней логических уравнений с предварительным распределением нагрузки основаны на расщеплении анализируемых булевых формул, используя разложение Шеннона по k переменным. Задача разбивается на не более чем $l = 2^k$ подзадач (не более, так как некоторые из подзадач могут иметь тривиальное решение, не требующее обсчета на процессоре). При этом, хотя время решения задачи существенно зависит от выбора переменных для разложения, используется тривиальная процедура разложения, так как она выполняется целиком на корневом процессоре в режиме последовательного выполнения. Полученные частные задачи решаются параллельно l процессорами, решающими свою задачу над своими исходными данными до конца, а корневой процессор собирает результаты частичных решений и формирует из них итоговое решение общей задачи. Если хотя бы для одной частной задачи найдется решение, то по нему однозначно находится набор значений переменных, выполняющих исходную КНФ.

В основе реализованного в настоящей работе метода лежит идея [16] 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. $K \equiv 0$).

Анализ матрицы $\boldsymbol{T}$ производится методом обхода дерева поиска выполняющего вектора $\boldsymbol{t}$. Текущая ситуация, представляемая в узле дерева, характеризуется значением вектора $\boldsymbol{t}$, отдельные компоненты которого определены (т.е. имеют значение 0 или 1), а другие (отмеченные «–») подлежат доопределению в процессе движения по дереву. Троичный вектор $\boldsymbol{t}$ однозначно определяет минор $\boldsymbol{T}(\boldsymbol{t})$ матрицы $\boldsymbol{T}$, расположенный на пересечении ее строк, соответствующих невыполнимым относительно вектора $\boldsymbol{t}$ дизъюнктам, и столбцов, отмеченных его неопределенными компонентами.

Распределение нагрузки производится корневым процессом динамически через магазин миноров по мере их формирования при обходе дерева поиска. Минор матрицы $\boldsymbol{T}$ задается вектором $\boldsymbol{t}$. Когда какой-либо подчиненный процесс при достраивании вектора $\boldsymbol{t}$ в процессе обхода своего поддерева поиска: 1) находит искомый вектор $\boldsymbol{t}$ (выполняющий все строки обрабатываемого минора), тогда задача решена и все остальные процессы прекращают работу, 2) убеждается в невыполнимости минора, тогда он освобождается и получает очередной минор из магазина или минор, полученный (по запросу корневого процесса) на очередном шаге обхода дерева

поиска некоторым занятым процессом (делит с ним нагрузку). Поиск выполняющего вектора ***t*** продолжается до тех пока, пока он ни будет найден (КНФ выполнима) или пока все подчиненные процессы не закончат работу (КНФ не выполнима).

В мультипроцессорных системах с индивидуальной памятью, к каким относится семейство СКИФ, текст MPI-программы перед ее исполнением копируется в память каждого из обрабатывающих процессов. При этом каждый из них в процессе работы должен «распознавать» и выполнять только свои участки программы, при необходимости обмениваясь сообщениями с другими процессами. Параллельная программа делится на общие, корневые и подчиненные участки, исполняемые всеми, корневым и подчиненными процессами. Для организации обмена сообщениями предусматриваются специальные приемо-передающие буферы. Для рассматриваемой задачи через них передаются векторы, задающие миноры и выполняющие наборы, сигналы окончания работы. При реализации параллельной программы был выделен минимально необходимый набор MPI-операций: для запуска и завершения работы, определения выделенной конфигурации кластера, выделения и распознавания блоков программы для процессов, работы с приемо-передающими буферами (опроса, приема и посылки), операций обмена сообщениями через буферы. Так как взаимодействие между процессами производится только через обмен сообщениями, то в участках программ для подчиненных процессов выделяются места, где производится опрос буферов на предмет приема сообщений от корневого процесса (чтобы принять сигнал окончания работы или разделить анализируемый минор).

Описываемый метод реализован MPI-программой «Вычислитель» для кластерного компьютера. Исходными данными для программы служит описание КНФ в текстовом виде или в формате DIMACS [18]. Формат DIMACS представляет собой стандартный интерфейс для SAT-решателей и используется для ввода заданий в известных соревнованиях SAT-решателей. В данном случае он используется, так как позволяет компактно представлять редкие троичные матрицы, каковые получаются при формальной верификации схемных описаний. Результатом выполнения программы является текст, содержащий режим и длительность решения, а также выполняющий вектор, если КНФ выполнима.

IV. Архитектура распределенной программы для кластерного компьютера

Грид-сервис можно рассматривать как надстройку над обычными Web-сервисами. Между грид-сервисами и Web-сервисами обнаруживается ряд различий, из которых главное состоит в том, что грид-сервис предназначен для ограниченного круга пользователей, имеющих потребность в высокопроизводительных вычислениях, в то время как Web-сервисы ориентируются на массового пользователя. Сервисный подход для управления работой программой кластерного компьютера привлекателен тем, что построен на использовании стандартных, открытых, универсальных протоколов и интерфейсов.

Грид-сервис для решения систем логических уравнений для кластерного суперкомпьютера имеет типовую структуру грид-систем для высокопроизводительных вычислений и представляет собой трехслойную систему из программы для суперкомпьютера, сервиса связи и агента пользователя.

Основным компонентом этой системы является программа для суперкомпьютера «Вычислитель». Агент пользователя взаимодействует с «Вычислителем» через сервис связи. Пользователь может управлять грид-сервисом посредством такого агента как веб-браузер или прямо из САПР СБИС. В распределенных системах сервис представляет собой программу, работающую без взаимодействия с пользователем (не имеющую интерфейса пользователя). Назначением сервиса является обслуживание других программ – клиентов сервиса, которые могут быть агентами пользователя или другими сервисами. Сервис связи с функциями аутентификации и авторизации пользователей, поддержки динамически генерируемых страниц, поддержки HTTPS для защищённых соединений с клиентами создает канал связи с суперкомпьютером стандартными, открытыми, универсальными протоколами и интерфейсами (рис. 3).

Рис. 3. Распределенная программа грид-сервиса решения логических уравнений на суперкомпьютере

Грид-сервис построен на основе общепринятых открытых стандартов и межплатформенного связующего программного обеспечения (middleware), являющегося посредником между программами, выполняемыми на удаленных компьютерах. Сервис связи – middleware грид-сервиса – базируется на технологии интеграции веб-служб протоколом SOAP (Simple Object Access Protocol) – протоколом обмена структурированными сообщениями в распределённой вычислительной среде. SOAP основан на языке XML и расширяет протокол прикладного уровня. Агенты, взаимодействующие по этому протоколу, имеют простое поведение: запрос/ответ. Агент-отправитель SOAP отправляет посредством HTTP-запроса XML-сообщение и получает результат в HTTP-отклике от

агента-получателя. Проколом SOAP создается канал связи между системой управления заданиями кластерного компьютера и сервером связи с клиентом пользователя (рис. 3).

Протокол работы грид-сервиса состоит из следующих шагов (рис. 4):

1) транспортировать файл с анализируемой КНФ в файловую систему на управляющей машине кластера;

2) запустить программу «Вычислитель» решения систем логических уравнений на суперкомпьютере;

3) возвратить файл с результатами решения на клиентский компьютер.

V. Сервис связи

Клиентская часть грид-сервиса может взаимодействовать с программой на суперкомпьютере посредством интерфейса Middleware UNICORE (Uniform Interface to Computing Resources) [19], который является открытым программным продуктом, совместимым со стандартными протоколами. Архитектура UNICORE трёхслойная. В неё входят: пользовательский уровень, сервер UNICORE и суперкомпьютер.

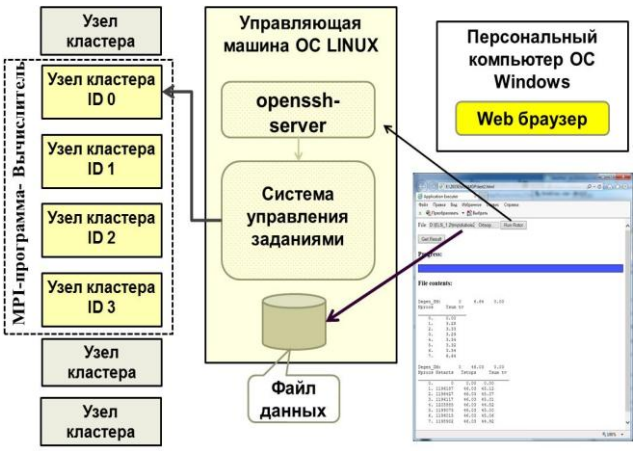

Рис. 4. Архитектура сервиса решения систем логических уравнений

Пользовательский уровень и UNICORE сервер связаны открытой сетью и взаимодействуют между собой через протокол SSL (Secure Socket Layer). Для установления клиент-серверного соединения протокол SSL использует криптографию с открытым ключом, так что компонент пользовательского уровня должен иметь секретный ключ шифрования.

Интерфейс UNICORE ориентирован на пакетную обработку заданий в суперкомпьютере и использует объектно-ориентированный подход, поэтому ключевым является понятие абстрактного задания. Задание описывает работу, которая должна быть выполнена суперкомпьютером в собственном формате представления заданий UNICORE. В задании указывается, какие данные импортируются перед выполнением программы в файловую систему кластерного компьютера и то, что нужно экспортировать после того, как задание завершится. Эти операции импорта и экспорта выполняются интерфейсом UNICORE.

Кластерный суперкомпьютер в модели UNICORE представлен как виртуальный сайт с единым файловым пространством. Каждый сайт состоит из интерфейса к локальной операционной системе и пакетной системы управления заданиями. Используемая версия Unicore 6 использует стандарты, основанные на XML: SOAP для связи пользовательского уровня и сервера, JSDL для представления работы, SAML для аутентификации и авторизации между пользовательским уровнем и сервера и XACML для аутентификации и авторизации между сервером и суперкомпьютером.

VI. Интеграция программного модуля решения систем логических уравнений в маршрут проектирования СБИС

Методика запуска на кластерном компьютере программ решения комбинаторных задач, возникающих в процессе проектирования интегральных схем, отработана на примере разработанного в лаборатории логического проектирования ОИПИ НАН Беларуси программного комплекса ЭЛС «Энергосберегающий логический синтез» [4]. 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 [20]. 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 (рис. 5) 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. Формальная верификация основана на сведении задачи к проверке выполнимости некоторой КНФ K, которая отражает структуру сравниваемых описаний и выполнимость которой 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 решено решать эту задачу с привлечением средств кластерного компьютера.

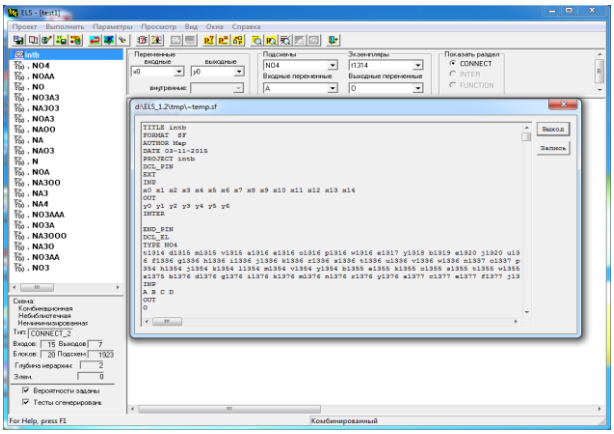

Рис. 5. Пример экрана системы ЭЛС в состоянии выполнения операции верификации

Так как набор проектных операций в системе ЭЛС на уровне интерфейса пользователя фиксирован, то для интеграции грид-сервиса решения логических уравнений программа верификации была модифицирована. После получения КНФ в процессе выполнении операции верификации в среде комплекса ЭЛС вместо вызова внутренней программы решения логических уравнений (SAT Solver Minisat) выполняются следующие действия: файл с анализируемой КНФ, заданной в формате DIMACS, транспортируется на управляющую машину кластера; запускается программа «Вычислитель» на суперкомпьютере; файл с результатами решения возвращается в систему ЭЛС. Если файл содержит найденный вектор (что говорит о том, что КНФ выполнима), делается вывод о неэквивалентности сравниваемых описаний, иначе описания эквивалентны.

С целью испытания предлагаемого подхода к распределенному решению задачи верификации в среде программного комплекса ЭЛС был верифицирован ряд 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.

Кроме того, с целью определения функциональной работоспособности программы «Вычислитель» на суперкомпьютере она была подвергнута испытаниям на потоке случайных КНФ с заданными параметрами и на нескольких КНФ [18], которые использовались для проверки SAT-решателей в международных соревнованиях этих программ и для которых известен результат решения. Результаты решения «Вычислителем» совпали с известными решениями.

Параметром оценки эффективности при решении задачи с применением суперкомпьютера является коэффициент ускорения $S = t(1) / t(l)$, где $t(1)$ – время, затраченное на решение задачи последовательным алгоритмом на одном процессоре, а $t(l)$ – время решения той же задачи параллельным алгоритмом с использованием l процессоров мультипроцессорной вычислительной системы. При $S = l$ получается линейное ускорение, пропорциональное числу процессоров. Поведение программы «Вычислитель» зависит от решаемой задачи. Если КНФ выполнима, то возможно «сверхлинейное» ускорение [17]. Если же КНФ невыполнима, то ускорение подчиняется закону Амдала и меньше линейного [17], так как для доказательства отсутствия решения каждый из подчиненных процессоров должен обследовать все свое подпространство до конца.

VII. Заключение

Разработка алгоритмов решения логико-комбинаторных задач для многопроцессорных суперкомпьютеров является высокоинтеллектуальной деятельностью, успешность которой существенно зависит от мастерства разработчика. Разработанный грид-сервис, обеспечивая управление параллельными вычислениями из САПР СБИС, дает возможность ускорить решение задач, возникающих при выполнении многих проектных оптимизационных процедур. Грид-сервис является открытым для включения в его состав новых параллельных алгоритмов и программ.

Литература

[1] Закревский А.Д., Поттосин Ю.В., Черемисинова Л.Д. Логические основы проектирования дискретных устройств. М.: Физматлит, 2007. – 589 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] Ganai M., Gupta A. SAT-Based Scalable Formal Verification Solutions. New York: Springer-Verlag, 2007. – 338 p.

[4] 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.

[5] Sterling T., Becker D., Savarese D., et al. Beowulf: A Parallel Workstation for Scientific Computation // Proc. of Intern. Conf. on Parallel Processing, Oconomowoc, 1995, pp. 11–14.

[6] Message Passing Interface Forum. MPI: A Message-Passing Interface standard, version 1.1. Available at: http://www.mpi-forum.org/docs (date of access: 10.02.2016).

[7] PBS (Portable Batch System) Basics for UNIX. Available at: http://tx.technion.ac.il/usg/PBS/PBS_basics.pdf (date of access: 10.02.2016).

[8] Сервер OpenSSH. Available at: https://help.ubuntu.com/lts/serverguide/openssh-server.html (date of access: 10.02.2016).

[9] Hyvarinen A.E.J., Junttila T.A., Niemela I. Partitioning SAT Instances for Distributed Solving // Lecture Notes in Computer Science. Springer, Heidelberg, 2010, Vol. 6397, pp. 372 – 386.

[10] Schulz S., Blochinger W. Parallel SAT Solving on Peer-to-Peer Desktop Grids // Journal of Grid Computing, 2010, Vol. 8, № 3, pp. 443–471.
[11] Hyvarinen A., Niemela I., Junttila T. Grid-Based SAT Solving with Iterative Partitioning and Clause Learning // LNCS, 2011, Vol. 6876, pp. 385–399.
[12] Posypkin M. Semenov A., Zaikin O. Using BOINC desktop grid to solve large scale SAT problems // Computer Science Journal, 2012, Vol. 13, № 1, pp. 25–34.
[13] Тимошевская Н.Е. Параллельные методы обхода дерева // Математическое моделирование, 2004, №1, с. 105–114.
[14] Schubert T., Lewis M., Becker B. PaMiraXT: Parallel SAT Solving with Threads and Message Passing // Journal on Satisfiability, Boolean Modeling and Computation, 2009, Vol. 6, pp. 203–222.
[15] Hamadi Y., Jabbour S., Sais L. ManySAT: a Parallel SAT Solver // Journal on Satisfiability, Boolean Modeling and Computation, 2009, Vol. 6, pp. 245–262.
[16] Торопов Н. Р. Параллельная проверка ДНФ на тавтологию // Информатика, 2005, № 2, с. 35–42.
[17] Черемисинов Д.И. Проектирование и анализ параллелизма в процессах и программах. – Минск: Беларуская навука, 2011. –300 с.//
[18] CNF Files. Available at: http://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html (date of access: 10.02.2016).
[19] Операционный центр национальной грид-сети Республики Беларусь. Available at: http://noc.grid.by/index.php?n=Main.Download (date of access: 10.02.2016).
[20] Черемисинова Л.Д., Новиков Д.Я. Программные средства верификации описаний комбинационных устройств в процессе логического проектирования // Программная инженерия, 2013, № 7, с. 8–15.

Parallel computing using for VLSI computer-aided design

D.I. Cheremisinov, L.D. Cheremisinova

United Institute of Informatics Problems of National Academy of Sciences of Belarus

cher@newman.bas-net.by, cld@newman.bas-net.by

***Keywords* — design automation, parallel computing, cluster computer, grid system, verification, Boolean satisfiability.**

ABSTRACT

The paper is devoted to the problem of CAD tools efficiency enhancement for the sake of combinatorial tasks computational speedup by means of code parallelization. The goal of the paper is twofold: first, to propose a framework simplifying the creation of distributed software for the supercomputer family "SKIF" [2], that is Beowulf cluster [5] having master-worker architecture and consisting of a large number of interconnected stand-alone (with individual memory) computational nodes and a master front-end node. The nodes run under Linux and collaborate through master node using MPI [6].

The second goal is to show how the parallel calculations conducted by the supercomputer can be integrated with CAD tools implemented in Windows environment. Because solving the logical equation in the form of conjunctive normal form (Boolean satisfiability – SAT) is the most often used approach as the underlying model for a significant and increasing number of EDA algorithms [1], its parallel implementation in the cluster like computing environment was suggested (similar as in [16]).

The methodology and software are suggested which provide data compatibility and interworking between cluster computer of Beowulf type and CAD system that are intended for solving laborious tasks of logical design. The suggested grid service is three-layer system consisting of MPI-program for supercomputer, link services and user agent and is based on using open protocols and middleware.

As an example the technology and results of fulfilling verification [3, 20] within the environment of program tool of logical design of custom CMOC VLSI [4] were researched. Verification was based on satisfiability problem solving that has been accomplished in remote access by the developed parallel program using high-performance supercomputer.

REFERENCES

[1] Zakrevskij A.D., Pottosin Yu.V., Cheremisinova L.D. Logicheskie Osnovy Proektirovaniya Diskretnykh Ustroistv, Moscow, Phizmatlit, 2007, 589 p. (in Russian).
[2] Abramov S.M., Paramonov N.N., Anistchenko V.V., Ablamejko S.V. Principles of Construction of Supercomputer family "SKIF" // Informatika, 2004, № 1, c. 89–106 (in Russian).
[3] Ganai M., Gupta A. SAT-Based Scalable Formal Verification Solutions. New York: Springer-Verlag, 2007. – 338 p.
[4] Bibilo P.N., Cheremisinova L.D., Kardash S.N., et al. Low-Power Logical Synthesis of CMOS Circuits Automation // Programnaia indzeneria, 2013, № 8, c. 35–41.
[5] Sterling T., Becker D., Savarese D., et al. Beowulf: A Parallel Workstation for Scientific Computation // Proc. of Intern. Conf. on Parallel Processing, Oconomowoc, 1995, pp. 11–14.
[6] Message Passing Interface Forum. MPI: A Message-Passing Interface standard, version 1.1. Available at: http://www.mpi-forum.org/docs (date of access: 10.02.2016).
[7] PBS (Portable Batch System) Basics for UNIX. Available at: http://tx.technion.ac.il/usg/PBS/PBS_basics.pdf (date of access: 10.02.2016).
[8] OpenSSH Server. Available at: https://help.ubuntu.com/lts/serverguide/openssh-server.html (date of access: 10.02.2016)) (in Russian).
[9] Hyvarinen A.E.J., Junttila T.A., Niemela I. Partitioning SAT Instances for Distributed Solving // Lecture Notes in

Computer Science. Springer, Heidelberg, 2010, Vol. 6397, pp. 372–386.
[10] Schulz S., Blochinger W. Parallel SAT Solving on Peer-to-Peer Desktop Grids // Journal of Grid Computing, 2010, Vol. 8, № 3, pp. 443–471.
[11] Hyvarinen A., Niemela I., Junttila T. Grid-Based SAT Solving with Iterative Partitioning and Clause Learning // LNCS, 2011, Vol. 6876, pp. 385–399.
[12] Posypkin M. Semenov A., Zaikin O. Using BOINC desktop grid to solve large scale SAT problems // Computer Science Journal, 2012, Vol. 13, № 1, pp. 25–34.
[13] Timoshevskaia N.E. Parallel Methods of tree-walk // Matematicheskoe modelirovanie, 2004, №1, c. 105–114 (in Russian).
[14] Schubert T., Lewis M., Becker B. PaMiraXT: Parallel SAT Solving with Threads and Message Passing // Journal on Satisfiability, Boolean Modeling and Computation, 2009, Vol. 6, pp. 203–222.
[15] Hamadi Y., Jabbour S., Sais L. ManySAT: a Parallel SAT Solver // Journal on Satisfiability, Boolean Modeling and Computation, 2009, Vol. 6, pp. 245–262.
[16] Toropov N.R. A Parallel Tautology DNF Checking // Informatika, 2005, № 2, c. 35–42 (in Russian).
[17] Cheremisinov D.I. Design and the analysis of parallelism in processes and programs. Minsk: Belaruskaja navuka, 2011. – 300 p. (in Russian).
[18] CNF Files. Available at: http://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html (date of access: 10.02.2016).
[19] Operational Center of National grid-system of Republik of Belarus. Available at: http://noc.grid.by/index.php?n=Main.Download (date of access: 10.02.2016) (in Russian).
[20] Cheremisinova L.D., Novikov D.Ya. Software tools for verification of descriptions of combinational circuits during the process of logic design // Programnaia indzeneria, 2013, № 7, c. 8–15 (in Russian).