УДК 681.518

Д.Е. ИВАНОВ, Ю.А. СКОБЦОВ, А.И. ЭЛЬ-ХАТИБ

Институт прикладной математики НАН Украины, Донецкий национальный технический университет, Украина

РАСПРЕДЕЛЕННЫЕ АЛГОРИТМЫ МОДЕЛИРОВАНИЯ И ГЕНЕРАЦИИ ТЕСТОВ

Рассматриваются проблемы построения распределённых алгоритмов генерации тестовых последовательностей и моделирования цифровых схем с неисправностями. Описаны различные пути организации параллельных вычислений.

Генетические алгоритмы, генерация тестов, моделирование с неисправностями, распределённые вычисления, многопроцессорные системы

Генетические алгоритмы (ГА) успешно используются при генерации проверяющих тестов цифровых схем [1] (с 90-х годов) наряду с детерминированными структурными методами. Опыт показывает, что генетические алгоритмы дают лучшие результаты для схем, ориентированных на обработку данных, в то время как детерминированные методы более успешно работают для сильно последовательностных схем со сложной управляющей логикой. В данной работе представлены распределённые ГА, которые позволяют расширить область эффективного применения ГА в данной проблемной области. Также описан распределённый алгоритм моделирования цифровых схем с неисправностями, построенный по аналогичной схеме.

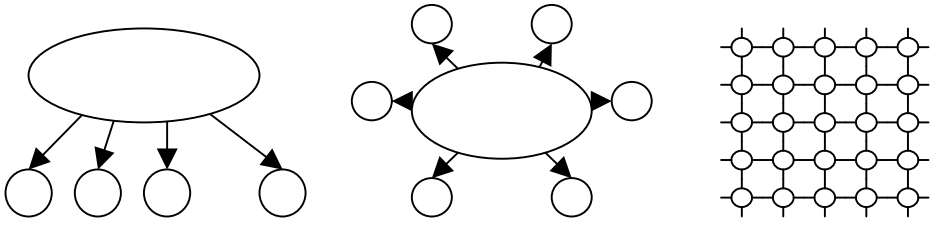
Распределённые генетические алгоритмы.

Присущий ГА "внутренний" параллелизм и заложенная в них возможность распределенных вычислений способствовали развитию параллельных ГА (ПГА). Первые работы в этом направлении появились в 60-х годах, но только в 80-е годы, когда были разработаны доступные средства параллельной реализации, исследования ПГА приняли систематический массовый характер и практическую направленность. В этом направлении разработано множество моделей и реализаций, некоторые из которых представлены ниже[2].

При параллелизации ГА ожидаются следующие преимущества:

1) поиск альтернативных решений одной и той же проблемы;
2) параллельный поиск из различных точек в пространстве решений;
3) допускают хорошую реализацию в виде островов или клеточной структуры;
4) большая эффективность поиска даже в случае реализации не на параллельных вычислительных структурах;
5) хорошая совместимость с другими эволюционными и классическими процедурами поиска;
6) существенное повышение быстродействия на многопроцессорных системах.

Рассмотрим современные основные методы реализации ПГА. Наиболее известной является представленная на рис.1.а) ***глобальная параллелизация***. Эта модель основана на простом (классическом) ГА с вычислениями,



а) б) в)

Рис.1. Различная реализация параллельных ГА.

выполняемыми параллельно. Она быстрее, чем классический ГА, выполняемый последовательно, и обычно не требует баланса по загрузке процессоров в системе, поскольку на разных процессорах чаще всего вычисляются значения фитнесс-функций для различных особей (имеющие примерно одинаковую вычислительную сложность). Исключение составляет генетическое программирование, где различные особи могут сильно отличаться по своей сложности (древовидные или граф- структуры). Эту модель часто называют "***рабочий - хозяин***" («клиент-сервер»). Многие исследователи используют пул процессоров для повышения скорости выполнения алгоритма, поскольку независимые запуски алгоритма на различных процессорах выполняются существенно быстрее, чем на одном процессоре. Отметим, что в этом случае нет никакого взаимодействия между различными прогонами алгоритма. Это чрезвычайно простой метод выполнения одновременной работы (если это возможно) и он может быть очень полезным. Например, данный подход может быть использован для решения одной и той же задачи с различными начальными условиями. В силу своей вероятностной природы ГА позволяют эффективно использовать этот метод. При этом ничего нового в сам алгоритм по сути ничего не вносится, но выигрыш во времени может быть значительным.

На рис.1 б) представлена также чрезвычайно популярная ***"модель островов"*** **(coarse grain)**, где множество подалгоритмов совместно работают параллельно, обмениваясь в процессе поиска некоторыми особями. Эта модель допускает прямую реализацию на компьютерных системах с MIMD-архитектурой. При этом каждый "остров" соответствует своему процессору.

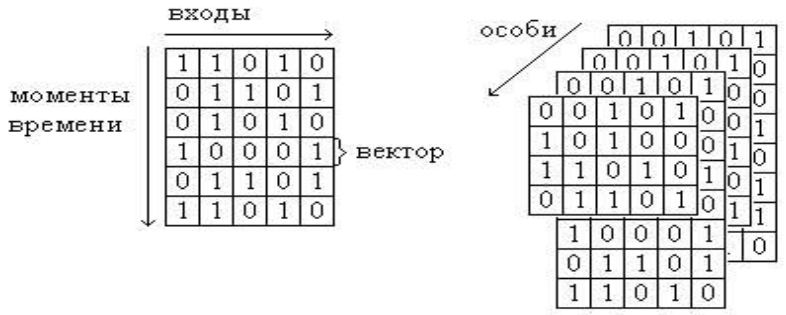
В **клеточных ГА (fine grain)** (рис.1. в) параллелизация обычно реализуется на компьютерных системах с SIMD-архитектурой, где каждый процессор представляет подпопуляцию (из одной особи). Хотя известны работы, в которых используются однопроцессорные компьютеры и системы с MIMD-архитектурой.

Генетические алгоритмы генерации тестов.

Целью автоматической генерации проверяющих тестов является построение входной последовательности двоичных наборов, которые проверяют любой физический дефект, возможный в процессе производства (или) эксплуатации логической схемы. Однако существует огромное число возможных потенциальных физических дефектов. Поэтому обычно рассматривают некоторый класс неисправностей, которые моделируют реальные физические дефекты. Таким образом, неисправность является моделью (одного или нескольких) физических дефектов. На практике чаще всего рассматривают одиночные константные неисправности, построение тестов для которых обычно дает удовлетворительные результаты. В случае необходимости (при низкой полноте тестов) тест достраивается для других типов неисправностей, таких как короткие замыкания, транзистор постоянно открыт (закрыт), задержки распространения сигналов и т.п. Известно, что генерация проверяющих тестов является NP-трудной задачей, т.е. в худшем случае решается путем перебора [1]. Поэтому, несмотря на то, что разработано множество последовательных структурных методов генерации тестов, которые для многих классов схем дают хорошие результаты, были предприняты попытки параллелизации алгоритмов построения тестов[3,4].

При генерации тестов для цифровых схем с применением ГА в качестве особи используется тестовая последовательность (рис.2а). Популяция состоит из фиксированного числа тестовых последовательностей, возможно, различной длины. Для выбранного таким образом представления особей и популяций применяются следующие проблемно ориентированные генетические операторы [1]:

1) Скрещивание. Реализуется два вида операции (рис.3): вертикальное и горизонтальное скрещивания, которые выполняются соответственно с вероятностями P_v и $P_h = 1 - P_v$.



а) особь б) популяция

Рис.2. Кодирование особей и популяций в ГА.

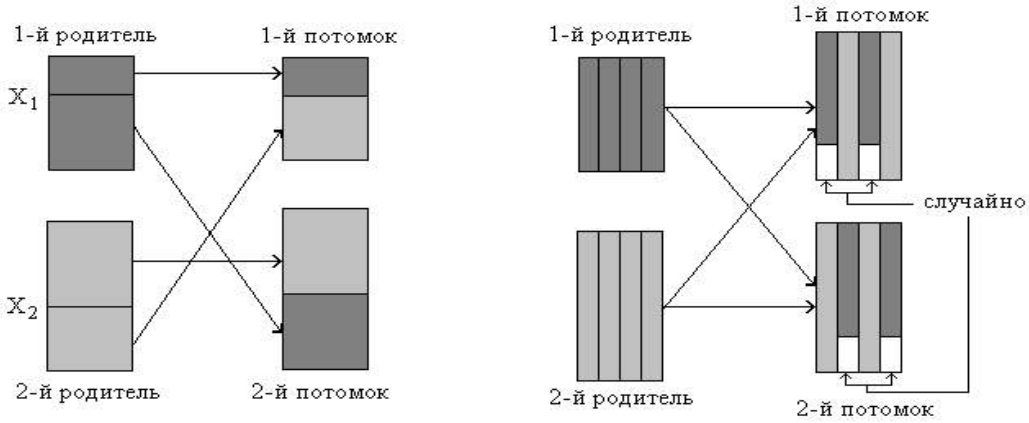


Рис.3. Операции горизонтального и вертикального скрещивания ГА.

2) Мутация. Применяются три вида операции соответственно с вероятностями P_{m_1}, P_{m_2} и P_{m_3}:

– удаление одного входного вектора из случайно выбранной позиции. Применение данной операции позволяет уменьшать длину генерируемой тестовой последовательности в том случае, когда удалённый вектор не ухудшает тестовые свойства последовательности;
– добавление одного входного вектора в случайную позицию, что также позволяет расширять область поиска решений;
– случайная замена битов в тестовой последовательности.

Поскольку целью генерации тестов является построение последовательности, на которой максимально отличаются значения сигналов в исправной и неисправной схемах, то качество тестовой последовательности (fitness-функция) оценивается как мера отличия значений сигналов в исправной и неисправной схемах. В простейшем случае для этого используются программы логического моделирования исправных цифровых схем, позволяющие оценить значения сигналов на двух соседних (во времени) тестовых наборах. На основе полученных данных моделирования разработаны fitness-функции следующего вида:

$$h(v, f) = c_1 f_1(v, f) + c_2 * f_2(v, f) \qquad (2.1)$$

где f_1 - и f_2 - число изменений сигналов на выходах логических вентилей и триггеров соответственно ($c_2 >> c_1$.). Целевая функция для последовательности наборов определяется как взвешенная сумма фитнесс-функций отдельных наборов

$$H(s, f) = \sum_{i=1}^{i=\text{длина}} L^i * h(v_i, f) \qquad (2.2)$$

где s - анализируемая последовательность; v_i - вектор из рассматриваемой последовательности, i – позиция

вектора в последовательности, f - заданная неисправность, $0 < L < 1$.

Если не удается построить тест с использованием целевой функции указанного типа, то применяются программы моделирования неисправных цифровых схем, требующие большие машинные ресурсы. Целевые функции при этом основаны на подсчете различий сигналов в исправной и неисправной схемах и имеют примерно такой же вид как и в предыдущем случае [6].

Параллельные генетические алгоритмы генерации тестов

В данной работе для параараллелизации ГА мы используем модель «рабочий - хозяин», поскольку она требует наименьших изменений в существующей версии программного обеспечения, реализующего ГА генерации тестов и дает неплохие результаты.

При этом каждый процессор имеет свою копию популяции. Затраты по вычислению значений фитнесс-функций (с использованием логического моделирования) равномерно распределяется по всем процессорам. Для всех процессоров используется один и тот же список неисправностей. Поэтому для n особей и P процессоров мы каждому процессору относим P/n особей. Значения фитнесс-функций вычисляются соответствующими (рабочими) процессорами и посылаются в один процессор (хозяин), который собирает всю информацию и передает ее всем процессорам. Каждый процессор имеет информацию о значениях фитнесс-функции для всех особей и может генерировать следующее поколение на этой основе.

Итак, процессор–хозяин выполняет центральную часть (ядро) алгоритма генерации тестов, в то время как логическое моделирование (исправных и неисправных) цифровых схем реализуется на процессорах – рабочих. Наиболее критичным по затратам вычислительных ресурсов является моделирование неисправных схем. Известны различные методы организации распределенного моделирования неисправностей, которые, в основном основаны на разбиении: 1) схем на подсхемы; 2) тестовой последовательности на подпоследовательности. Мы будем использовать комбинированный подход, сочетающий эти два метода.

На первом и втором этапах моделируемые входные последовательности распределяются между рабочими процессорами. На первом этапе каждый рабочий процессор загружается моделированием одной подпоследовательности. Для баланса список непроверенных неисправностей разбивается на примерно одинаковые подмножества.

В конце каждого из трех этапов помещаются точки синхронизации. Когда процессор-хозяин достигает эти точки, он переходит в режим ожидания, пока все рабочие процессоры не закончат свои задания, что гарантирует глобальную корректность алгоритма.

При этом работа между процессором-хозяином и рабочими распределяется следующим образом.

Процессор-хозяин:

- выполняет все вход-выходные операции с пользователем и файловой системой: он читает описание схемы и список неисправностей и записывает построенную входную тестовую последовательность;
- первоначально запускает «рабочие» процессы на доступных ресурсах;
- распределяет копии (внутреннего представления) схем и списков неисправностей каждому рабочему процессору;
- организует управление процессом генерации тестов: по мере моделирования неисправностей на последовательностях входных наборов посылает соответствующие сообщения активации рабочих процессоров; по окончании задания рабочим процессором процессор-хозяин принимает полученные результаты и соответственно изменяет глобальные структуры данных (общий список неисправностей, значения фитнесс-функций для особей и и т.п.).

Рабочий процессор хранит локальную копию внутреннего представления схемы и списка неисправностей. Каждый «рабочий» принимает входную последовательность от «хозяина» и определяет, какие неисправности проверяются этой последовательностью, путем логиче-

ского моделирования вычисляет значения фитнесс-функции для особей. Полученный результат посылает хозяину и ожидает следующего задания.

Поскольку размер популяции много больше числа процессоров достигается хороший баланс в загрузке процессоров. Для каждого рабочего процессора изменение локального списка согласно передаваемой информации о проверяемых и непроверяемых неисправностях от других рабочих процессоров требует достаточно много ресурсов и является критической.

Окончательные результаты (тестовые входные последовательности и полнота обнаружения неисправностей) близки к тем, что получены на однопроцессорной компьютерной системе с использованием аналогичного алгоритма. Качество решения (полнота теста) при этом не теряется и в большинстве случаев улучшается, а время генерации тестов существенно сокращается.

Диаграмма потоков данных в данном алгоритме приведена на рис.4.

Распределённое моделирование цифровых устройств.

Описанный выше алгоритм распределённой генерации тестовых последовательностей во многом основан на алгоритме распределённого моделирования цифровых схем. Опишем кратко и этот алгоритм.

Данный алгоритм также основан на модели «рабочий-хозяин». Он построен по принципу разбиения списка моделируемых неисправностей с выделением одного процессора в качестве сервера. В качестве среды обмена информацией выступает протокол TCP/IP, что позволяет включать в вычислительный кластер как машины с одной вычислительной платформой, так и построенные на разных платформах. Алгоритм распределённого моделирования схем с неисправностью распадается на два независимых алгоритма. Первый алгоритм реализуется на головном процессоре, который называется «хозяином» (сервером), и осуществляет управляющие функции, а также файловый ввод-вывод. Сервер начинает работу с ввода описания схемы, полного списка неисправностей и тестовой последовательности. Данные процедуры осуществляются с помощью файлового ввода-вывода. После этого происходит запуск сокета-сервера и поиск сокетов-клиентов. Если во время описка обнаружены процессы-клиенты, то полный список неисправностей разбивается пропорционально их числу. Далее в цикле для каждого процесса-клиента выполняются следующие действия. Клиенту передаётся описание схемы, часть списка неисправностей и тестовая последовательность. После чего процес-сервер переходит в режим ожидания результатов. После получения списков непроверенных неисправностей от каждого из процессов-клиентов процесс-сервер формирует полный отчёт моделирования: полнота тестовой последовательно-

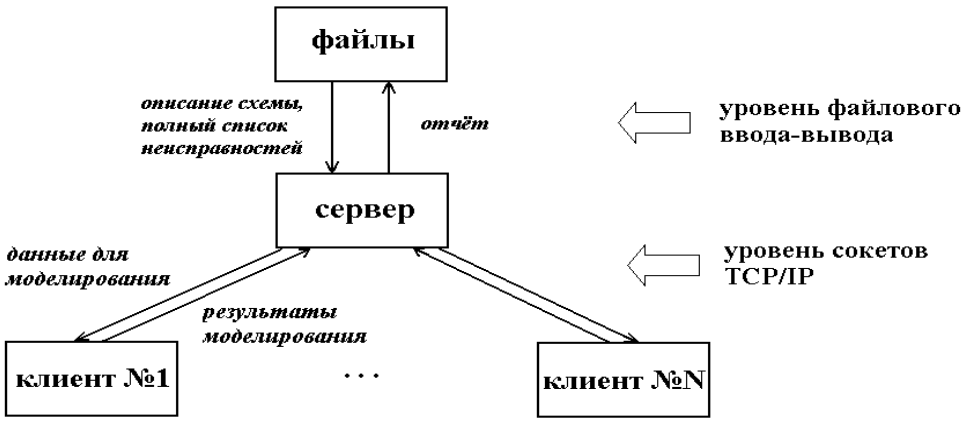


Рис.4 Потоки данных в распределённых алгоритмах генерации тестов и моделирования.

сти, условная полнота, время моделирования на каждом процессе-клиенте, время обмена информацией с каждым процессом-клиентом и общее время работы.

Второй алгоритм непосредственно выполняет моделирование схем с неисправностями и работает на нескольких доступных процессорах-«клиентах». Число доступных процессоров в кластере определяется сервером до начала моделирования. В предлагаемом алгоритме каждый клиент выполняет моделирование заданной тестовой последовательности только на части полного списка неисправностей. Разбиение полного списка неисправностей осуществляется сервером до начала моделирования и, следовательно, является статическим. На каждом элементе кластера, который используется для моделирования в качестве клиента, реализован алгоритм параллельного по неисправностям моделирования с динамическим сжатием неисправностей [5]. Данный алгоритм показывает хорошие временные характеристики для тестовых схем из каталога ISCAS-89 [6]. Он претерпел незначительные изменения, которые касаются только ввода схемы, списка неисправностей и тестовой последовательности.

Следует заметить, что описанное взаимодействие между процессором-хозяином и процессорами-клиентами также соответствует диаграмме на рис.4.

Проведённые машинные эксперименты подтверждают эффективность предложенной схемы параллелизации алгоритма моделирования. Однако полностью линейного увеличения быстродействия достичь не удаётся. Основным препятствием для этого являются необходимость многократной (по числу задействованных процессоров-клиентов) передачи описания схемы, а также избыточное моделирование работы исправной схемы на каждом рабочем процессоре.

Заключение.

В статье приведён обзор различных способов организации распределённых алгоритмов на вычислительном кластере. Предложены распределённые алгоритмы генерации тестовых последовательностей цифровых устройств и моделирования таких устройств с неисправностями. Для распараллеливания вычислений использовалась одна и та же схема «рабочий-хозяин» и технология блокирующих сокетов, что позволяет строить вычислительные кластеры на процессорах с различной вычислительной платформой. Предложенные распределённые алгоритмы позволяют добиться существенного ускорения в решении указанных задач, одновременно обеспечивая равномерную загрузку процессоров клиентов.

В качестве дальнейших исследований можно отметить исследование возможности построения ГА генерации тестов с использованием других схем распараллеливания вычислений, а также сравнительный анализ получаемых результатов.

Литература

1. Y.A.Skobtsov, V.Y.Skobtsov, D.E.Ivanov. Evolutionary approach to the test pattern generation for the sequential circuits // Радиоэлектроника и информатика.- 2003, №3.- с.46-51.
2. Скобцов Ю.А., Эль-Хатиб А.И. Параллельные генетические алгоритмы // Наукові праці Донецького національного технічного університету Серія:“Обчислювальна техніка та автоматизація” Випуск 90.-Донецьк : ДонНТУ. – 2005.-с.137-144.
3. D.Krishnaswamy, M.Hsiao, V.Saxena, E.M.Rudnick, J.P.Patel. Parallel genetic algorithms for simulation-based sequential circuit test generation // IEEE VLSI Design Conference, 1997.- pp.475-481.
4. F.Corno, P.Prinetto, M.Rebaudengo, M.Sonza Reorda, E.Veiluva. Aportable ATPG tool for parallel and distributed systems // Proc VLSI Teat Symp.- pp.29-34.
5. Иванов Д.Е., Скобцов Ю.А. Параллельное моделирование неисправностей для последовательностных схем // Искусственный интеллект. – 1999. - №1. – С.44-50.
6. Brgles F., Bryan D., Kozminski K. Combinational profiles of sequential benchmark circuits // International symposium of circuits and systems, ISCAS-89. – 1989. – P.1929-1934.