

ПАРАЛЛЕЛЬНЫЙ ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ВЕРИФИКАЦИИ ЭКВИВАЛЕНТНОСТИ ЦИФРОВЫХ СХЕМ ДЛЯ ДВУХЯДЕРНЫХ СИСТЕМ

Д.Е. Иванов, к.т.н., с.н.с., доцент, Институт Прикладной Математики и Механики НАН
Украины,

E-mail: ivanov@iamm.ac.donetsk.ua

Верификация эквивалентности цифровых схем – одна из центральных задач технической диагностики. В статье предлагается модификация предложенного авторами ранее алгоритма верификации для двухядерных систем. Параллельная версия алгоритма позволяет существенно увеличить загрузку ядер процессора и, следовательно, повысить скорость работы. Проведенные эксперименты показывают ускорение работы в среднем в 1.87 раза для контрольных схем из каталога ISCAS-89.

1. Введение

Прогресс микроэлектроники ставит перед разработчиками задачи обработки схем очень большой размерности. Современные средства автоматизированного проектирования позволяют работать со схемами, содержащими десятки тысяч логических вентилей, в том числе с последовательностными схемами, содержащими тысячи элементов состояний. Безусловно, работа со схемами такой размерности требует также средств их оптимизации. Такие средства, обычно, изменяют логическую структуру схемы, минимизируя некоторые параметры, например число вентилей в логическом блоке. Однако они должны оставлять неизменной логику функционирования. В связи с этим одной из центральных задач технической диагностики цифровых схем является задача проверки эквивалентности двух заданных схем.

Для решения данной задачи ранее предложены точные алгоритмы, которые базируются, в основном, на булевых преобразованиях схемы [1-2]. Однако с ростом размерности схем такой подход всё чаще даёт отрицательные результаты. Это связано с переполнениями памяти для стеков возвратов в данных алгоритмах. Таким образом, будучи прерванными из-за проблемы переполнения памяти, точные алгоритмы вообще не дают никакого результата: ни об эквивалентности, ни о неэквивалентности заданных схем. Это заставляет разрабатывать новые нетрадиционные подходы к решению подобных задач. Одним из таких подходов является применение генетических алгоритмов [3].

Генетические алгоритмы уже применялись авторами для решения ряда традиционных задач технической диагностики. В частности были описаны алгоритмы построения тестовых последовательностей [4], иницирующих последовательностей [5-6] и последовательностей, которые проверяют эквивалентность двух схем [7]. Во всех упомянутых генетических алгоритмах особи представлены в виде входных двоичных последовательностей. Вычисление их оценочных функций основано на моделировании работы цифровых схем (исправном или с неисправностями). Также для повышения скорости вычисления фитнес-функций, авторами был предложен распределённый алгоритм моделирования цифровых схем с неисправностями [8]. В нём в качестве вычислительной платформы использовался кластер, построенный на обычной локальной сети.

В данной статье предлагается иной подход к повышению быстродействия генетических алгоритмов. Он основан на доступности многоядерных процессоров. Для алгоритма верификации эквивалентности двух схем предлагается параллельный алгоритм вычисления оценочной функции. Оценка особи вычисляется путём распараллеливания процесса моделирования работы двух верифицируемых схем на вычислительные ядра процессора. Показано, что такой подход позволяет существенно повысить скорость работы алгоритма на двухядерных рабочих станциях.

Данная статья имеет следующую структуру. Во втором разделе описана постановка задачи верификации последовательностных схем. В третьем разделе кратко описан предложенный ранее алгоритм верификации эквивалентности цифровых схем. В следующем разделе описаны источники параллелизации алгоритма и описана новая процедура вычисления фитнес-функции. В пятом разделе приведены экспериментальные данные по повышению быстродействия работы алгоритма. Направления дальнейших исследований и выводы рассмотрены в разделе шесть.

2. Источники параллелизма.

В настоящее время многоядерные процессоры стали стандартом «де-факто» для персональных рабочих станций, а в некоторых областях применения (например, серверы) фактически вытеснили одноядерные системы. Число ядер в коммерчески доступных процессорах доходит до четырёх (процессоры серии Intel Core Quad), то в ближайшем будущем это число возрастет до 32 [9].

Казалось бы, увеличение вычислительной мощности должно вести к пропорциональному росту производительности вычислительных систем. Однако на самом деле этого не происходит. Для подтверждения этого факта авторы провели следующий эксперимент. На вычислительных системах с одноядерным и многоядерным процессорами

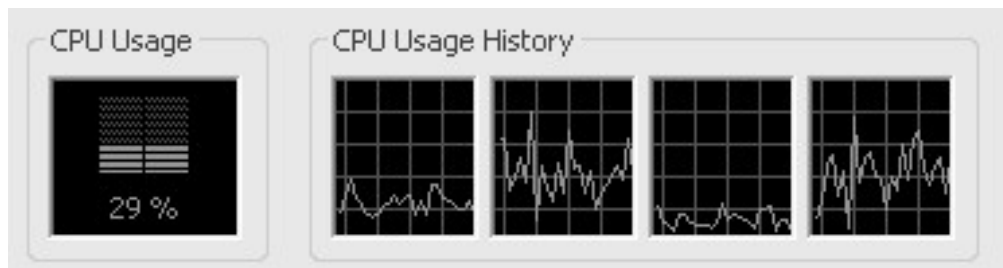


Рис.1. Загрузка ядер процессора приложением с одним вычислительным потоком.

запускались характерные задачи САПР по обработке цифровых схем, в частности, генетический алгоритм генерации входных тестовых последовательностей. С помощью стандартной утилиты ОС Windows Task Manager наблюдалась вычислительная загрузка ядер процессоров. Для одноядерного процессора данная загрузка была равна 100%. Для процессора с двумя ядрами большая часть вычислительной работы пришлась на одно ядро (загрузка 100%), тогда как второе ядро процессора практически простаивало, обслуживая в основном запросы ОС и других приложений. Для четырёх ядерного процессора ситуация оказалась ещё хуже (рис.1). Ни одно из ядер процессора не имело полной загрузки. На диаграмме видны два ядра с переменной средней загрузкой около 35-40%. Причиной данной ситуации является технология процессоров Intel Core2Duo, которая пытается один вычислительный процесс распределить на несколько ядер с целью унификации их загрузки. Таким образом видно, что добавление вычислительных ядер не только не улучшает выполнение существующих алгоритмов, а в некоторых случаях производительность даже несколько падает. Аналогичные результаты авторы наблюдали и при выполнении других программных компонент системы АСМИД-Е, а также иных САПР.

Видно, что многоядерность не стала панацеей от проблемы резкого снижения удельной производительности процессоров в расчёте на один транзистор [11]. Для одноядерных процессоров проблема частично решалась путём внедрения аппаратных разновидностей параллелизма: параллельная работа нескольких арифметических устройств; конвейеризация; применение встроенных параллельных вычислений с аппаратной поддержкой команд (MMX и подобные расширения). Однако с приходом многоядерности проблема встала ещё острее. Отсюда возникает задача повышения эффективности алгоритмов САПР в части унификации ядер процессоров. Здесь возможны два пути развития: 1) создание принципиально новых алгоритмов, позволяющих эффективно использовать всю вычислительную мощь современных процессоров; 2) модификация существующих алгоритмов для параллельных/многоядерных систем. Однако загрузка ядер вычислительной работой не является самоцелью. Она является только средством улучшения характеристик алгоритмов: времени работы, качества поиска и т.д.

Генетические алгоритмы генерации входных последовательностей основаны на моделировании работы ЦУ. Анализ общих путей параллелизации алгоритмов моделирования выполнен в работе [12], где выделяются три основных пути:

- алгоритмический параллелизм, использующий технологию конвейеров, при этом различные этапы алгоритма выполняются на разных процессорах;
- параллелизм данных: различные процессоры используются для моделирования работы схемы на разных входных векторах;
- модельный параллелизм, при котором различные процессоры используются для вычисления логических значений разных вентилей.

Второй путь неоднократно применялся для решения задачи ускорения моделирования и построения тестов, основанном на моделировании [13-14]. Для апробации данных алгоритмов авторы использовали дорогостоящие сети рабочих станций DEC Alpha.

Авторы также выбрали для исследования второй путь. Мы предлагаем параллельную версию ранее известного алгоритма. Его отличительной чертой будет параллельная организация вычислительного процесса для некоторых критических участков алгоритма, что даст существенное повышение его производительности. Также алгоритм ориентирован не на кластерную архитектуру, а на рабочую станцию с двухядерным процессором.

3. Алгоритм верификации.

Алгоритм верификации эквивалентности двух последовательностных схем предложен авторами в [7]. Здесь мы только кратко опишем его структуру.

В предложенном ранее алгоритме мы переформулировали задачу верификации эквивалентности на противоположную: генетический алгоритм построения ищет входные последовательности, которые показывают неэквивалентность двух схем.

Определение 1. Если существует такая последовательность s_k , на которой при одинаковых начальных состояниях схемы A_0 и A_1 имеют различные выходные реакции, то данные схемы не являются эквивалентными.

$$A_0 \neq A_1 \text{ если } \exists s_k : A_0(s_k) \neq A_1(s_k). \quad (1)$$

Поэтому предложенный ранее алгоритм не является точным в следующем смысле: он не даёт точный ответ об эквивалентности двух сравниваемых схем, но он с большой вероятностью подтверждает неэквивалентность этих схем.

Для кодирования особей и популяций используется широко распространенное кодирование особей в виде двоичных последовательностей, а популяций в виде набора таких

последовательностей [4, 5, 14]. Там же подробно описаны генетические операции, применяемые при построении новых поколений особей.

Алгоритм продолжает работу до выполнения одного из условий:

- либо найдена последовательность s , выходные реакции на которую схем $A_0(s)$ и $A_1(s)$ различны (показана неэквивалентность схем);
- либо достигнуто одно из ограничивающих условий: время моделирования, предельное число поколений или предельное число поколений без улучшения оценочной функции.

Построение различающих последовательностей основано на моделировании работы двух верифицируемых схем. Это отражено в построении различающей функции: поиск направляется в области, где выше активность обеих схем, а также выше различия этой активности на множестве контрольных точек. Фитнесс-функция имеет следующий вид:

$$f(A_0, A_1, s_i) = \sum_{j=1}^{\text{длина } s_i} f(A_1, A_0, s_{ij}) = \sum_{j=1}^{\text{длина } s_i} (c_1 * n_1 + c_2 * n_2 + c_3 * n_3), \quad (2)$$

где:

$c_1 - c_3$ - нормирующие константы;

$n_1 = \sum_{g \in G} \text{различие}(g, A_0, A_1, s_{ij})$, G – множество всех внешних выходов двух схем; n_1 - число различных значений на выходах схемы. Данный параметр является определяющим, поскольку наличие даже одного расхождения на внешних входах двух схем, говорит о том, что различающая последовательность построена;

$n_2 = \sum_{g \in G1} \text{различие}(g, A_0, A_1, s_{ij})$, $G1$ – множество псевдовыходов двух схем; n_2 - число различных псевдовыходов схемы. В случае, когда различные значения в двух проверяемых схемах, ещё не достигли выходов, решающим является распространения таких отличий на псевдовыходы (триггеры) схемы;

$n_3 = \sum_{g \in G2} \text{различие}(g, A_0, A_1, s_{ij})$, $G2$ – множество всех контрольных точек схемы; n_3 - число контрольных точек двух схем с различными значениями сигналов. В случае, когда различие в поведении двух схем не достигло ни внешних выходов, ни псевдовыходов схемы, необходимо строить последовательности, для которых является важным различие сигналов внутри комбинационных блоков. В нашем случае мы предполагаем, что проектировщику известно поведение всей схемы, поэтому множество $G2$ совпадает с множеством всех вентилях схемы.

Функция «различие(g)» вычисляется на основе моделирования и определяется следующим образом:

$$\text{различие}(g, A_0, A_1, s_{ij}) = \begin{cases} 1, & \text{если значения сигналов на выходе блока } g \text{ различны в двух} \\ & \text{схемах } A_0 \text{ и } A_1 \text{ при моделировании на входном наборе } s_{ij}; \\ 0, & \text{в противном случае.} \end{cases} \quad (3)$$

Таким образом, при задании фитнес-функции описанным выше способом, алгоритм строит входные последовательности, которые будут продуцировать различие на внешних выходах схемы, что соответствует определению 1.

Именно итеративное вычисление оценочной функции для всех особей популяции определяет временную сложность алгоритма.

4. Модификация для двухядерных систем.

В данном разделе будет предложена модификация алгоритма, рассчитанная на работу в двухядерных рабочих станциях. Данная модификация направлена на повышение загрузки обоих ядер в таких системах. В предложенном ранее алгоритме существовал один вычислительный поток. При этом возникала ситуация, неполной загрузки вычислительных ядер системы. Новый алгоритм призван улучшить ситуацию путём более равномерной загрузки ядер вычислениями. Это достигается за счёт организации двух потоков вычисления. Причём оба этих потока должны нести высокую вычислительную нагрузку. В этом случае процессор системы направит два таких вычислительных потока на различные вычислительные ядра, что и повысит их загрузку.

Предложенный ранее алгоритм большую часть процессорного времени выполняет моделирование работы исправных схем:

- в фазе построения начальной популяции необходимо выполнять моделирование для оценки качества потенциальных особей;
- в основной фазе генетического алгоритма для оценки фитнес-функции каждой особи.

При этом каждая такая процедура подразумевает моделирование двух сравниваемых схем. Распределив моделирование схем на разные ядра процессора, мы улучшим загрузку процессорных ядер и получим увеличение скорости работы всего алгоритма.

Рассмотренный ранее алгоритм позволяет произвести такое распараллеливание вычислительной работы без существенного изменения самого алгоритма. Для этого следует отметить, что для вычисления фитнес-функции одной особи в популяции фактически необходимо произвести моделирование работы двух схем: эталонной и модифицированной. Поскольку мы предполагаем, что модифицированная схема имеет такую же последовательностную структуру, то вычислительные ресурсы, необходимые для их моделирования, также должны быть одинаковыми. Также заметим здесь, что вычисление фитнес-функций особей является наиболее трудоёмкой задачей генетических алгоритмов,

основанных на моделировании. Таким образом, для равномерной загрузки ядер центрального процессора, следует распределить моделирование двух верифицируемых схем по ядрам.

Для реализации данного подхода модифицируем алгоритм вычисления фитнес-функции отдельной особи.

В описанном ранее алгоритме не организовывались потоки моделирования работы схем, а вызывалась функция «Моделирование_работы_исправной_схемы(описание_схемы, входная_последовательность)». В терминах потоков моделирования вызов данной функции соответствует созданию потока, запуску его на выполнение и ожиданию окончания выполнения. Тогда вычисление оценочной функции для одной особи в терминах потоков выглядит следующим образом:

```
Вычисление_Фитнесс(схема_1, схема_2, последовательность)
{
    Поток_1=Создать_Поток_Моделирования(схема_1, последовательность);
    Поток_1->Ждать_Завершения();
    Поток_2=Создать_Поток_Моделирования(схема_2, последовательность);
    Поток_2->Ждать_Завершения();
    Фитнесс=Сравнение_схем(схема_1, схема_2);
}
```

В такой интерпретации видно, что для создания и запуска второго потока моделирования необходимо ждать завершения работы первого потока. Структурная схема использования ядер процессора в этом случае выглядит следующим образом (рис.2а). Видно, что при такой организации вычислительного процесса, общая загрузка процессора не может быть более 50%, поскольку фактически второе ядро простаивает. Очевидно, что для систем с несколькими вычислительными ядрами такой подход является неприемлемым.

Ситуация легко исправляется, если оба потока моделирования заставить выполняться

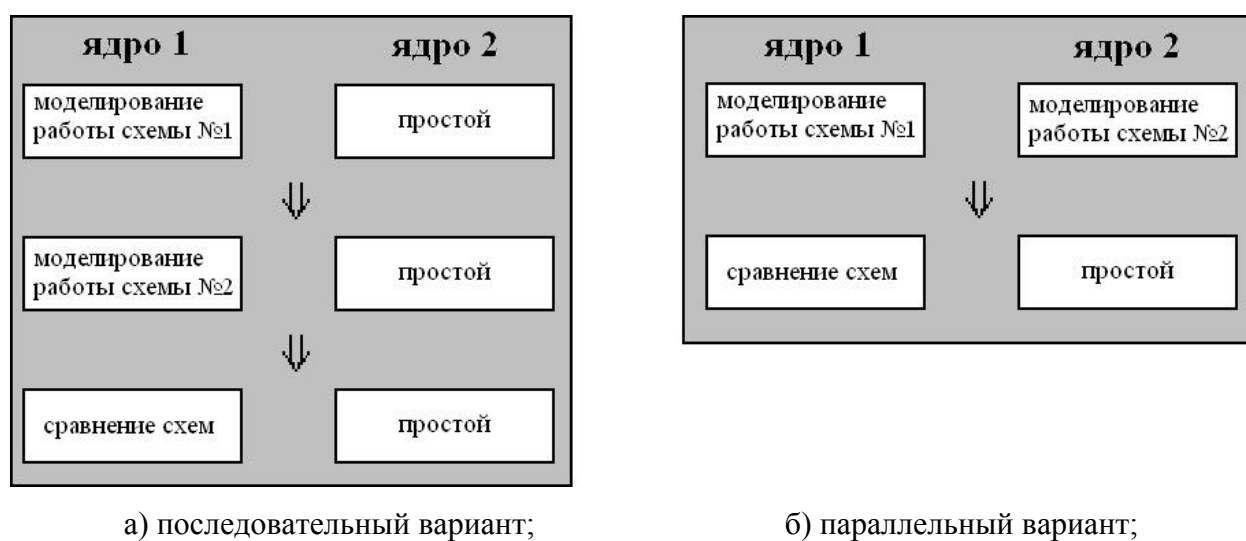


Рис.2. Загрузка вычислительных ядер процессора при вычислении фитнес-функции одной особи.

одновременно на разных вычислительных ядрах. Структурно это изображено на рис.2б. Псевдокод такой процедуры вычисления фитнес-функции в терминах потоков приведён ниже.

```
Вычисление_Фитнесс(схема_1, схема_2, последовательность)
{
    Поток_1=Создать_Поток_Моделирования(схема_1, последовательность);
    Поток_2=Создать_Поток_Моделирования(схема_2, последовательность);
    Поток_1->Ждать_Завершения();
    Поток_2->Ждать_Завершения();
    Фитнесс=Сравнение_схем(схема_1, схема_2);
}
```

Остальная часть алгоритма остаётся неизменной. Отметим, что внесённые в алгоритм изменения касаются только процедур моделирования и не затрагивают «генетическую надстройку» над ними.

Далее мы будем называть два варианта данного алгоритма «последовательная» (старая) и «параллельная» (новая) версии алгоритма.

5. Экспериментальные данные.

Описанный в предыдущем разделе алгоритм был реализован программно в среде программирования C++ Builder 6. Объём программной реализации составил около 2000 строк кода.

Данная среда программирования содержит средства разработки многопоточных приложений. Для этого в неё включен невизуальный класс TThread. Создавая объект данного класса программист создаёт вычислительный поток, привязанный к данному приложению. Возможно управление данным потоком (запуск, остановка) и передача параметров при запуске. В нашем случае в качестве параметров передаются ссылки на описание моделируемых схем и входной последовательности.

Машинные эксперименты проводились на двух различных платформах, реализующих вычислительные системы с числом вычислительных ядер процессора от 1 до 4. Система с одним и двумя вычислительными ядрами процессора построена на базе двухядерного процессора Intel Core 2 Duo E4500 с 1Гбайт оперативной памяти. Для одноядерной реализации в BIOSе системы отключалось одно ядро процессора. Система с тремя и четырьмя вычислительными ядрами базировалась на Intel Core Quad Q6600 с 2Гбайт оперативной памяти. При этом для трёхядерной системы средствами BIOS также отключалось одно ядро процессора.

Чтобы выяснить влияние предложенной параллельной модификации алгоритма на его работу, выполнялось фиксированное число итераций алгоритма. При этом число вызовов

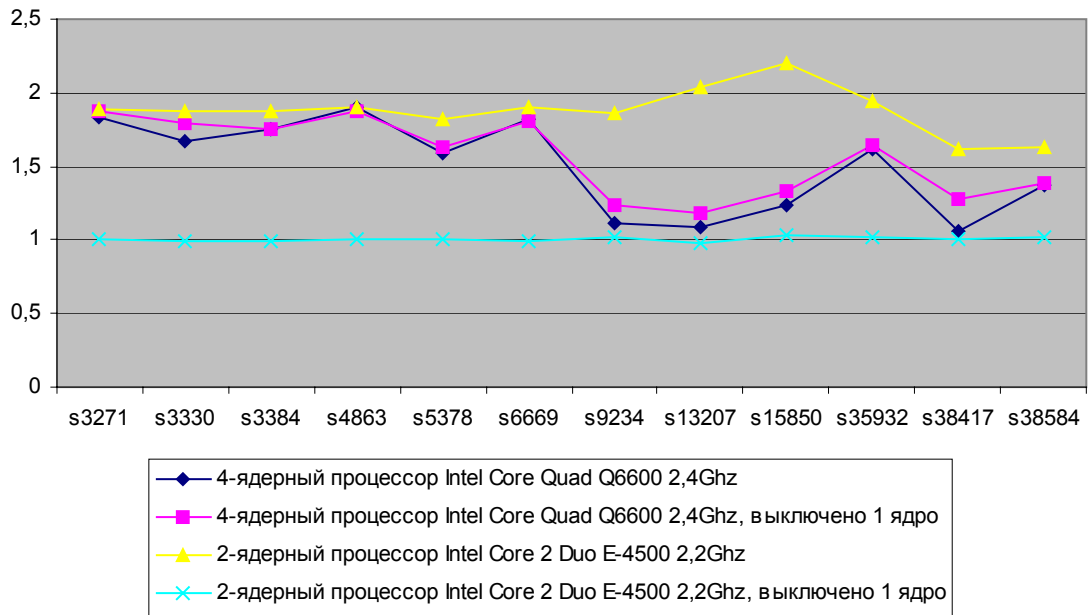


Рис.3. Увеличение быстродействия для контрольных схем на разных экспериментальных платформах.

процедуры вычисления фитнес-функции составило 7000. Другие параметры запуска алгоритма: число особей популяции

Для сравнения результатов для двух данных систем, измеренное в результате экспериментов время было преобразовано в относительное ускорение. В качестве эталона на каждом типе систем запускалась последовательная версия алгоритма. Далее мы будем приводить цифровые данные только в виде ускорения работы алгоритма.

Машинные эксперименты проводились на схемах из международного каталога ISCAS-89 [17]. На рис.3 приведены диаграммы, показывающие ускорение работы алгоритма для 12 схем на всех экспериментальных платформах. Наибольшее ускорение получено для двухядерной системы, наименьшее – для одноядерной. Средние графики с примерно одинаковыми числовыми значениями соответствуют трёх- и четырёхядерным системам. Нижний график соответствует одноядерной системе. При этом коэффициент ускорения колеблется в окрестностях единицы для всех схем, показывая, что такая модификация алгоритма не влияет на быстродействие алгоритма в одноядерных системах.

На рис.4 показано среднее увеличение быстродействия по всем схемам для различных экспериментальных платформ. Видно, что наибольшее среднее увеличение быстродействия модифицированной версии алгоритма (1.88 раза) соответствует двухядерной системе. Для трёх- и четырёхядерных систем ситуация ухудшается и увеличение быстродействия составляет 1.57 и 1.5 раза соответственно. По-видимому, это объясняется тем, что процессор

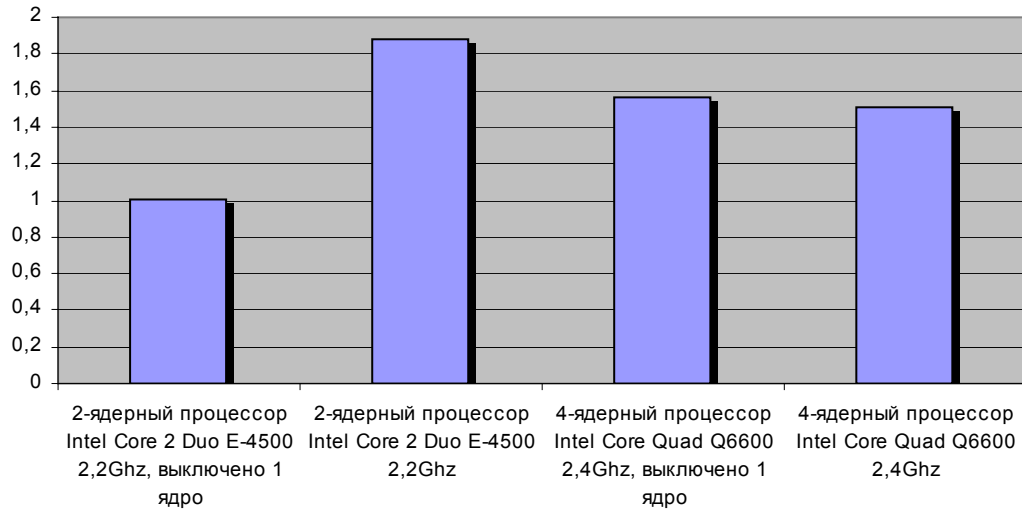


Рис.4. Среднее увеличение быстродействия для различных экспериментальных платформ.

пытается распределить каждый из вычислительных потоков на более чем одно ядро с целью избежать их простоя.

Для параллельной реализации алгоритма принято вычислять параметры ускорение и эффективность реализации [18].

Ускорение, рассчитываемое для параллельной реализации алгоритма для p процессоров определяется формулой:

$$S_p(n) = \frac{T_1(n)}{T_p(n)}, \quad (4)$$

где p – число процессоров в параллельной реализации алгоритма, n – параметр вычислительной сложности алгоритма, $T_i(n)$ – время выполнения параллельного алгоритма на системе с i процессорами.

Эффективность использования процессоров при параллельной реализации алгоритма рассчитывается по формуле:

$$E_p(n) = \frac{T_1(n)}{p \cdot T_p(n)} = \frac{S_p(n)}{p}. \quad (5)$$

Для нашего подхода под числом процессоров будем понимать число ядер процессора. Тогда параметр $E_p(n)$ будет выражать эффективность использования ядер процессора.

Доля последовательного кода (доля последовательных вычислений) f служит для оценки свойств алгоритма к распараллеливанию и определяется из закона Амдаля:

Табл.1. Значения параметров ускорения, эффективности использования ядер и доли последовательных вычислений для контрольных схем ISCAS-89.

	s3271	s3330	s3384	s4863	s5378	s6669	s9234	s13207	s15850	s35932	s38417	s38584	средн.
S_p	1,87	1,87	1,87	1,90	1,82	1,90	1,86	2,00	2,20	1,94	1,62	1,63	1,88
E_p	0,94	0,94	0,94	0,95	0,91	0,95	0,93	1,00	1,10	0,97	0,81	0,82	0,94
f	0,07	0,07	0,07	0,05	0,10	0,05	0,08	0,00	0,00	0,03	0,23	0,23	0,06

$$f = \frac{p/S_p - 1}{p - 1}. \quad (6)$$

Приведём числовые значения параметров ускорения, эффективности и последовательной части кода для контрольных схем для двухядерной системы (табл.1). Отметим, что приводить значение предельного ускорения алгоритма (при $p \rightarrow \infty$) не имеет смысла, поскольку предложенный алгоритм не является масштабируемым в традиционном смысле параллельных вычислений. Однако узкая оптимизация позволила достичь очень высоких параметров для двухядерных систем: средняя эффективность загрузки ядер процессора =0,94, средняя доля последовательных вычислений =0,06.

6. Выводы.

В статье предложена модификация ранее описанного авторами алгоритма верификации эквивалентности цифровых последовательностных схем. Данная модификация заключается в распараллеливании процедуры вычисления фитнес-функции отдельной особи с целью повышения быстродействия алгоритма. Модифицированный алгоритм рассчитан на работу в многоядерных рабочих станциях. Проведённые машинные эксперименты показывают, что наибольшее ускорение достигается на двухядерных системах (число ядер равно числу вычислительных потоков).

Однако продолжающийся рост числа ядер в настольных рабочих станциях делает возможным дальнейшее усовершенствование алгоритма. Одним из возможных путей такой модификации является модификация процедуры оценки особей таким образом, чтобы распараллелить этот процесс по всем ядрам вычислительной системы. Таким образом удастся достичь большей загрузки ядер, а значит эффективности использования вычислительной системы.

Литература

1. S.-Y. Huang, K.-T. Cheng, *Formal Equivalence Checking and Design Debugging*, Kluwer Academic Publishers, Boston, 1998.- 229p.

2. A. Ghosh, S. Devadas and A.R. Newton, *Sequential Logic Testing and Verification*, Kluwer Academic Publishers, 1992.- 214p.
3. Goldberg D.E., *Genetic Algorithm in Search, Optimization, and Machine Learning*.- Addison-Wesley.- 1989.- 432p.
4. Иванов Д.Е., Скобцов Ю.А. *Генерация тестов цифровых устройств с использованием генетических алгоритмов* // Труды института прикладной математики и механики НАН Украины. – Т.4. – Донецк, ИПММ. – 1999. – С.82-88.
5. Д.Е. Иванов, Ю.А. Скобцов, А.И. Эль-Хатиб *Построение инициализирующих последовательностей синхронных цифровых схем с помощью генетических алгоритмов*.- Проблемы інформаційних технологій.-2007.-№1.-с.158-164.
6. Skobtsov Y.A., El-Khatib, Ivanov D.E. *Distributed Genetic Algorithm of Test Generation For Digital Circuits* // Proceedings of the 10th Biennial Baltic Electronics Conference.-Tallinn Technical University,2006.-p.281-284.
7. Иванов Д.Е. *Генетический алгоритм проверки эквивалентности последовательностных схем* // «Радіоелектроніка. Інформатика. Управління».- Запоріжжя, ЗНТУ.- 2009.- №1(20).- С.118-123.
8. Д.Е. Иванов, Ю.А. Скобцов, Эль-Хатиб А.И. *Распределённое параллельное моделирование цифровых схем с неисправностями* // Наукові праці Донецького національного технічного університету. Серія: “Обчислювальна техніка та автоматизація”. Випуск 107.-Донецьк: ДонНТУ. – 2006.- С.128-134.
9. Intel® Software Insight. Multi-core Capability. July 2005.
10. Скобцов Ю.А., Иванов Д.Е. *Автоматизированная система моделирования и генерации тестов АСМИД-Е* // Техническая диагностика и неразрушающий контроль. - 2000. - №2. - С.54-59.
11. Ю.С. Затуливер, Е.А. Фищенко *Компьютер ПС-2000:Многопроцессорная архитектура, опередившая время* // Пленарные и избранные доклады 4-й Международной конференции «Параллельные вычисления и задачи управления», Москва, 27-29 октября 2008.
12. Roger D. Chamberlain. *Parallel Logic Simulation of VLSI Systems* // Proceedings of the 32nd ACM/IEEE conference on Design automation, 1995. P.p. 139-143.
13. F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, *A Parallel Genetic Algorithm for Automatic Generation of Test Sequences for Digital Circuits* // International Conference on High-Performance Computing and Networking, Brussels (Belgium), April, Lecture Notes In Computer Science; Vol. 1067.- 1996, P:454-459.
14. Krishnaswamy D., Hsiao M.S., Saxena V., Rudnick E.M., Patel J.H., Banerjee, P. *Parallel genetic algorithms for simulation-based sequential circuit test generation* //

- Proceedings Tenth International Conference on VLSI Design, 1997, 4-7 Jan 1997.- P:475 – 481.
15. F. Corno, M. Sonza Reorda, M. Rebaudengo, *Experiences in the use of evolutionary techniques for testing digital circuits* // Aproc. of Conf. Applications and science of neural networks, fuzzy systems, and evolutionary computation, San Diego CA, ETATS-UNIS (20/07/1998) 1998 , vol. 3455, pp. 128-139.
 16. Барашко А.С., Скобцов Ю.А., Сперанский Д.В. *Моделирование и тестирование дискретных устройств.* – Киев:Наукова думка, 1992. – 288 с.
 17. Brgles F., Bryan D., Kozminski K. *Combinational profiles of sequential benchmark circuits* // International symposium of circuits and systems, ISCAS-89. – 1989. – P.1929-1934.
 18. Гергель В.П., Стронгин Р.Г. *Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебное пособие.* - Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2003. - 184 с.