УДК 681.518

Д.Е. ИВАНОВ
Институт прикладной математики и механики НАН Украины

ПОДХОДЫ К ПОСТРОЕНИЮ ПАРАЛЛЕЛЬНЫХ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ИДЕНТИФИКАЦИИ ЦИФРОВЫХ СХЕМ ДЛЯ МНОГОЯДЕРНЫХ СИСТЕМ

В статье предлагаются практические подходы к адаптации параллельных генетических алгоритмов генерации идентифицирующих последовательностей цифровых схем для рабочих станций с многоядерными процессорами. Выделены три подхода к построению таких алгоритмов. Распараллеливанию подвергаются процедуры моделирования работы цифровой схемы на заданной входной последовательности, которые либо сами формируют процедуру оценки особи, либо являются её частью. Программная реализация основана на многопоточном программировании. Приведены результаты машинных экспериментов на схемах ISCAS-89.

The article offers a practical approach to the adaptation of the parallel genetic algorithms of the identifying sequences generation of digital circuits for workstations with multi-core processors. Highlighted are three approaches to constructing such algorithms. Paralleling procedures are modeling digital circuits on a given input sequence, which either by themselves form a procedure for evaluating an individual or is a part of it. The software implementation is based on many-threaded programming. The results of experiments on the circuits ISCAS-89 are given.

Ключевые слова: генетический алгоритм, параллельные вычисления, многоядерные процессоры.

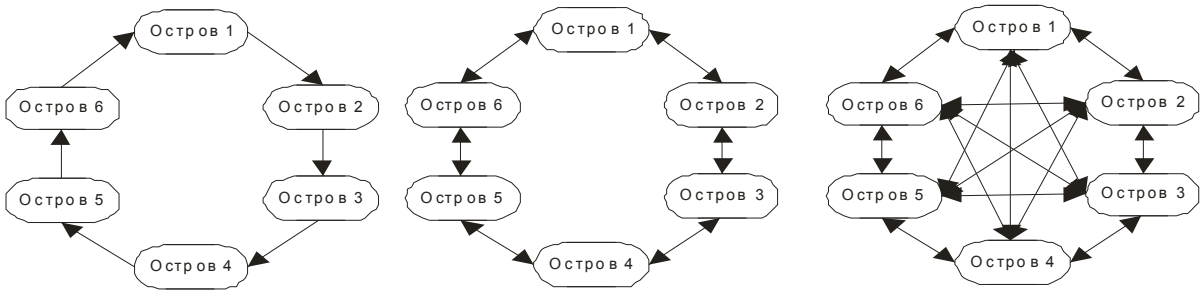
Введение

В настоящее время средства автоматизированного проектирования позволяют эффективно разрабатывать цифровые схемы, содержащие десятки и сотни тысяч логических вентилей, а также тысячи элементов состояний. В последнее время в этом направлении получили широкое распространение генетические алгоритмы (ГА) [1–3]. Их преимущество над структурными методами построения идентифицирующих последовательностей заключается в способности обрабатывать схемы очень большой размерности. Это связано с тем, что для оценки строящейся последовательности они используют информацию, полученную из процедур моделирования. Фактически, при этом происходит подмена задачи синтеза задачами анализа.

Недостатком такого подхода является скорость работы алгоритмов. Причина кроется в том, что основная вычислительная нагрузка ложится на процедуры оценки качества потенциальных решений, которые в ГА построения тестов представлены процедурами моделирования. Это заставляет искать пути выхода из ситуации. Например, в работе [4] предложен ГА построения тестов для многопроцессорной вычислительной системы. Эксперименты проводились на дорогостоящем кластере, состоящем из 16 рабочих станций Dec Alpha 3000/500. Авторы данной статьи также ранее описывали параллельные генетические алгоритмы, которые основаны как на модели «островов» [5], так и на модели «рабочий-хозяин» [6], которые ориентированы на кластер, построенный на обычной топологии Ethernet.

В данной работе предлагается иной подход к повышению быстродействия генетических алгоритмов. Он основан на доступности многоядерных процессоров [7]. Массовое их внедрение в рабочие станции не влечёт за собой автоматического увеличения быстродействия при использовании существующего программного обеспечения. Система мониторинга загрузки процессора покажет либо 100% загрузки одного из ядер, либо неравномерную загрузку всех ядер. Это связано с тем, что существующее программное обеспечение САПР разработано в эпоху одноядерных процессоров и базируется на последовательных алгоритмах. Очевидно, что для более полного использования вычислительных возможностей современных процессоров программное обеспечение также должно быть адаптировано. Возможны два пути развития:

- создание принципиально новых алгоритмов, позволяющих эффективно использовать всю вычислительную мощь современных процессоров;
- модификация уже существующих алгоритмов под актуальную аппаратную платформу.



а) обмен по кольцу **б) двунаправленный обмен по кругу** **в) обмен каждый с каждым**

Рис. 1. Различные топологи взаимодействия компонент распределенных ГА

Для реализации авторы выбрали второй подход. В данной статье будут описаны способы адаптации для многоядерных вычислительных систем описанных ранее параллельных алгоритмов построения идентифицирующих последовательностей.

На первый взгляд такая адаптация кажется очевидной, поскольку для этого достаточно каждому компьютеру кластера в существующем параллельном алгоритме поставить в соответствие ядро процессора для нового алгоритма. Однако такое прямое копирование, скорее всего, не даст приемлемого результата. Это связано со структурой вычислительных сред. В первом случае параллельная структура представлена системой с распределённой памятью, тогда как во втором – с общей и к тому же с ограниченной кэш-памятью. Поэтому авторы предлагают другой подход. Поскольку самыми трудоёмкими процедурами в генетическом алгоритме являются функции оценки особей, то именно они становятся узким местом в повышении производительности. Оптимизация данного фрагмента алгоритма для работы на многоядерных процессорах позволит существенно повысить скорость работы всего ГА. Все предлагаемые подходы основаны на понятии многопоточности, которая является основным инструментом построения приложений для многоядерных процессоров.

Данная статья имеет следующую структуру. Во введении обоснована актуальность проводимых исследований. Во втором разделе мы описываем адаптацию к многоядерным системам параллельных ГА по схеме «островов». В следующем разделе для таких систем описаны ГА, которые построены на модели «рабочий–хозяин». В заключении делаются выводы и указываются возможные дальнейшие исследования в данном направлении.

Схема «островов» локальных параллельных ГА

В этом разделе описывается адаптация к многоядерным системам параллельных ГА построения идентифицирующих последовательностей, которые основаны на схеме «островов». Обычно такие распределённые ГА разрабатываются для кластерных архитектур, которые состоят из нескольких независимых рабочих станций со своей локальной памятью. На каждой из таких систем выполняется своя копия ГА (построение новых поколений популяций). При этом генетические параметры работы этих алгоритмов должны несколько отличаться друг от друга. Это позволит направить поиск на каждом таком «острове» в отличном от других направлении. Через заданное количество итераций (достаточно редко) острова производят обмен лучшими особями, что позволяет корректировать направление поиска для менее удачных островов.

Эффективность параллельных ГА, построенных по такой схеме, определяет именно взаимодействие компонент. Можно выделить основные характеристики такого обмена:

- топология, которая определяет, какие подпопуляции будут считаться соседними и, соответственно, между какими островами будет возможен обмен особями (рис.1);
- время изоляции, которое определяет, сколько эпох ГА не будет производиться миграция;
- степень миграции, которая определяет количество особей, участвующих в миграции;
- стратегия отбора особей для миграции;
- стратегия удаления особей из подпопуляции при проведении миграции;
- стратегия репликации мигрирующих особей.

Заметим также, что хотя популяции развиваются независимо и равноправно, в реализациях такого вида алгоритмов выделяется сервер, который инициализирует работу и реализует топологию обмена данными. Более того, синхронизация работы копий ГА на островах требует способа их взаимодействия. Точная и корректная реализация такого взаимодействия различных копий ГА в модели островов является сложной задачей [8]. Для этого реализуются протоколы, в которых фиксируются точки обмена информацией между островами и сервером.

Адаптация алгоритма заключается в организации многопоточного приложения, в котором каждый поток реализует ГА для своего острова. Также выделяется один управляющий поток. Каждому потоку назначается одно ядро процессора. На этом ядре реализуется как основной цикл ГА острова, так и процедуры моделирования для оценки особей, которые развиваются на этом острове.

Такая схема адаптации имеет целый ряд недостатков:

- несмотря на то, что популяции развиваются на одной вычислительной системе с общей памятью, приходится реализовывать протоколы взаимодействия островов, которые как усложняют реализацию, так и замедляют её; более того, синхронизация работы копий ГА влечёт к росту простоя ядер процессора;
- число ядер в современных процессорах существенно ниже, чем число островов в реализации параллельных ГА;
- ограничения структуры процессоров, когда общий кэш выделятся на группу ядер.

Последнее замечание является существенным. Каждая копия ГА должна иметь доступ к полному описанию схемы, которая в памяти инструментальной ЭВМ представлена совокупностью таблиц [9], размер которых прямо зависит от размерности схемы. Возможно два способа предоставить копии ГА доступ к описанию схемы: 1) каждый «остров» имеет свою копию таблиц описания схемы; 2) все «острова» за данными обращаются к одной общей копии структуры данных. В первом случае каждый остров будет генерировать обращение к своей области данных, однако проходить они будут через общий кэш. Во втором

случае обращения будут производиться к одному участку памяти, однако гораздо чаще. Таким образом, узким местом в обоих подходах становится пропускная способность подсистемы кэш-основная память.

Отмеченные недостатки заставляют искать другие практические пути адаптации параллельных ГА к многопроцессорным рабочим станциям.

Схема «рабочий–хозяин» локальных параллельных ГА

Схема «рабочий–хозяин» предполагает, что в параллельном ГА существует только один основной цикл построения популяций, который соответствует процессору «хозяину». Все остальные процессоры являются «рабочими» и они реализуют процедуры вычисления оценок особей.

Как было отмечено выше, именно эти процедуры несут наибольшую вычислительную нагрузку в ГА построения входных идентифицирующих последовательностей, и лишь незначительную – процедуры построения популяций. Это связано с тем, что вычисление оценки основано на моделировании работы схемы на данной входной последовательности. Для решения проблемы в распределённых вычислительных средах авторами был предложен алгоритм моделирования, ориентированный на кластерную среду [6]. Реализация данного алгоритма для многоядерных систем [10] показывает, что можно добиться существенного увеличения быстродействия за счёт организации параллельного моделирования на системах такого класса.

Описываемый в данной реализации подход к адаптации заключается в распараллеливании процедур моделирования оценки особей. В этом случае главный цикл ГА реализуется потоком-«хозяином», который поручает другим вычислительным потокам-«рабочим» производить такую оценку. Таким образом, построение ГА соответствует схеме «хозяин–рабочий».

Для распараллеливания процедур вычисления оценочных функций авторы выделили три подхода. Как отмечено выше, все они основаны на многопоточном программировании.

Первый подход предназначен для многоядерных систем и тех алгоритмов, которые допускают распараллеливание внутри процедуры оценки одной особи. Авторы реализовали его для алгоритма построения верифицирующих последовательностей [11]. В данном алгоритме оценочная функция входной последовательности строится на основании моделирования и сравнения поведения двух верифицируемых схем. Последовательная реализация фрагмента кода вычисления оценочной функции в терминах потоков имеет вид:

```
Вычисление_Фитнесс(схема_1, схема_2, последовательность)
{
  Поток_1=СоздатьПотокМоделирования(схема_1, последовательность);
  Поток_1->ЖдатьЗавершения();
  Поток_2=СоздатьПотокМоделирования(схема_2, последовательность);
  Поток_2->ЖдатьЗавершения();
  Фитнесс=Сравнениесхем(схема_1, схема_2);
}
```

Видно, что моделирование второй схемы в последовательном алгоритме не начинается до завершения моделирования первой, что и даёт в результате загрузку одного ядра и простой второго. Это не позволит получить общую загрузку процессора выше 50% для любой системы с более чем двумя вычислительными ядрами.

Ситуация легко исправляется, если оба потока моделирования заставить выполняться одновременно на разных вычислительных ядрах. Псевдокод такой процедуры вычисления фитнесс-функции в терминах потоков приведён ниже.

```
Вычисление_Фитнесс(схема_1, схема_2, последовательность)
{
  Поток_1=СоздатьПотокМоделирования(схема_1, последовательность);
  Поток_2=СоздатьПотокМоделирования(схема_2, последовательность);
  Поток_1->ЖдатьЗавершения();
  Поток_2->ЖдатьЗавершения();
  Фитнесс=СравнениеСхем(схема_1, схема_2);
}
```

В данной реализации создаются сразу два вычислительных потока, каждый из которых выполняет моделирование одной схемы. После этого происходит сравнение результатов моделирования и вычисление оценки. Поскольку предполагается, что схемы имеют одинаковую последовательностную структуру, а отличаются в реализации комбинационных блоков, то сложность процедур их моделирования примерно одинакова и для разных ядер время моделирования будет также примерно одинаковым. Поскольку в нашем примере реализуется два параллельных потока, то наибольшее ускорение должно быть получено для двухядерных систем.

Для проверки эффективности предложенной модификации алгоритма были проведены эксперименты на контрольных схемах из международного каталога ISCAS-89 [12]. Здесь и далее программная реализация алгоритмов проводилась в среде C++ Builder 6 с использованием потоковых

классов TThread. Апробация проводилась на четырёх платформах с разным количеством ядер процессора. Стратегия экспериментов (как и в двух дальнейших случаях) заключалась в фиксации некоторого числа поколений в генетическом алгоритме. Это, с одной стороны, позволяло исследовать зависимость от оптимизации процедур моделирования, а с другой стороны учитывала, что в генетическом алгоритме помимо моделирования есть «накладные расходы», которые выражаются в процедурах построения популяций. Поскольку платформы не являлись эквивалентными, то для каждой из них измерялось время работы как последовательной, так и параллельной версий алгоритма. Результаты экспериментов приведены в табл.1. На основании этих данных были вычислены средние для всех схем значения ускорений для каждой платформы. Диаграмма на рис.2 показывает сравнение этих значений. Видно, что наилучшее ускорение 1.88 раза получено для платформы с двухядерным процессором. Дальнейшее увеличение числа ядер не улучшает ситуации. Этот факт является очевидным, поскольку описанная модификация является двухпоточной.

Таблиця 1

Время работы модификаций алгоритма на различных инструментальных платформах

модификация	схема											
	s3271	s3330	s3384	s4863	s5378	s6669	s9234	s13207	s15850	s35932	s38417	s38584
2-ядерный процессор Intel Core 2 Duo E-4500 2,2Ghz, выключено 1 ядро, время работы, сек.												
последовательное	102	73	89	232	63	276	60	111	187	827	517	599
параллельное	102	74	90	232	63	280	59	114	181	812	512	587
2-ядерный процессор Intel Core 2 Duo E-4500 2,2Ghz, время работы, сек.												
последовательное	100	73	88	226	62	278	54	98	165	780	468	561
параллельное	53	39	47	119	34	146	29	48	75	402	289	345
4-ядерный процессор Intel Core Quad Q6600 2,4Ghz, выключено 1 ядро, время работы, сек.												
последовательное	103	79	91	229	72	270	73	99	155	689	414	487
параллельное	55	44	52	122	44	149	59	84	116	420	325	350
4-ядерный процессор Intel Core Quad Q6600 2,4Ghz, время работы, сек.												
последовательное	106	80	95	232	76	280	77	103	159	702	412	498
параллельное	58	48	54	122	48	154	69	95	128	436	391	363

Для параллельной версии алгоритма принято вычислять параметры ускорения, эффективности и доли последовательного кода [13]. Ускорение, рассчитываемое для параллельной реализации алгоритма, определяется формулой:

$$S_p(n) = \frac{T_1(n)}{T_p(n)}, \qquad (1)$$

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.

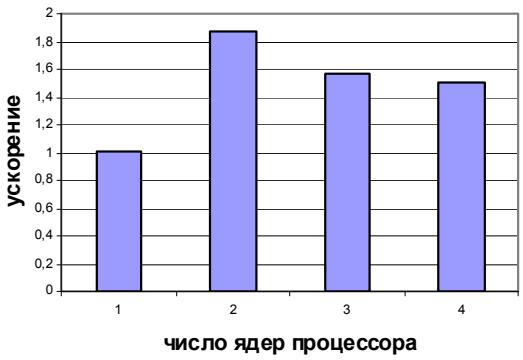


Рис. 2. Среднее увеличение быстродействия для различных экспериментальных платформ

Эффективность использования процессоров при параллельной реализации алгоритма рассчитывается по формуле:

$$E_p(n) = \frac{T_1(n)}{p \cdot T_p(n)} = \frac{S_p(n)}{p}. \qquad (2)$$

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:

$$f = \frac{p/S_p - 1}{p - 1}. \qquad (3)$$

Для системы с лучшим значением среднего ускорения мы вычислили данные характеристики в соответствии с формулами 1–3, которые приведены в табл. 2. Заметим, что среднее значение эффективности загрузки ядер равно 0.94, что является очень высоким показателем.

Второй подход можно применять для тех алгоритмов, в которых вид оценочной функции не допускает явного распараллеливания вычислений внутри неё. В этом случае в виде отдельных исполняемых потоков можно оформить полностью процедуру вычисления оценочной функции отдельных особей. Привязав потоки к

ядрам моделирования, мы получим ситуацию, когда на первом ядре вычисляется оценка первой особи, на втором ядре – для второй и т.д. Число особей в генетическом алгоритме обычно велико (100 и

более) в сравнении с числом ядер инструментальной ЭВМ, и поэтому необходимо дополнительное исследование, которое должно показать количество одновременно выполняемых потоков с наилучшей загрузкой вычислительных ядер.

Таблиця 2

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

Вычисление оценок особей популяции осуществляется в функции *«ОценитьПопуляцию»*. В первоначальной реализации данного генетического алгоритма [11] эта процедура имела следующий вид:

```
ОценитьПопуляцию(Популяция, ЧислоОсобей)
{
  for( int i=0 ; i<ЧислоОсобей ; i++ ){
    ОценитьОсобь(Популяция->Особь[i]);
  }
}
```

Видно, что такая организация процесса является полностью последовательной: вычисление оценки последующей особи (индекс $i+1$) будет начато только после окончания вычислений для текущей особи (индекс i). Построенные таким образом вычисления, как и в первом случае, будут давать загрузку только одного ядра, а остальные ядра процессора будут простаивать. На самом же деле ситуация выглядит ещё хуже. Процессор не направит всю вычислительную нагрузку в одно ядро, а распределит её между всеми имеющимися ядрами. Например, для четырёхядерного процессора будет выделено одно ядро с большей нагрузкой ($\approx 70\%$), остальные ядра будут иметь загрузку меньше 10%.

Чтобы избежать данного недостатка, предлагается изменить работу процедуры вычисления оценок особей. Для этого необходимо вычислять оценки особей в популяции параллельно. Причём число параллельно работающих процедур *«ОценитьОсобь»* должно соответствовать числу вычислительных ядер процессора. Изменённая процедура оценки особей популяции для произвольного заранее заданного числа потоков (переменная *ЧислоПотоков*) приведена ниже.

```
ОценитьПопуляцию(Популяция, ЧислоОсобей, ЧислоПотоков)
{
  for( int i=0 ; i<ЧислоОсобей/ЧислоПотоков ; i++ ){
    j=i*ЧислоПотоков;
    выполнять_параллельно_для j, j+1, … , j+ЧислоПотоков-1
    {
       ОценитьОсобь(Популяция->Особь[j]);
    }
    ЖдатьОкончанияПотоков j, j+1, … , j+ЧислоПотоков-1;
  }
}
```

Реальное повышение производительности будет сильно зависеть от механизма переключения потоков операционной системы и механизма доступа процессора к памяти, который должен обеспечивать вычислительные ядра необходимыми для работы данными.

Для проверки эффективности предложенной модификации алгоритма авторы проводили эксперименты с программной реализацией на системе с 4-ядерным процессором Intel Core2Quad. Мы изменяли число одновременных потоков от одного до четырёх (число ядер в инструментальной ЭВМ). Также для сравнения мы провели эксперименты с числом потоков, которое заведомо (и существенно) больше числа ядер в системе. Числовые результаты приведены в табл.3. Строка, в которой для выполнения указана одна функция, соответствует последовательному коду и берётся в качестве эталона. Значения среднего ускорения для всех схем приведены на сравнительной диаграмме (рис. 3, столбец «подход 2»). Эксперименты показали, что ускорение работы алгоритма растёт и стабилизируется на отметке 2.05. Для некоторых контрольных схем получено ускорение 3.30 раза. Существенную отрицательную роль играет тот факт, что чип Intel Core2Quad не является чисто четырёхядерным процессором, а содержит два двухядерных процессора, причём каждая пара содержит свой кэш данных.

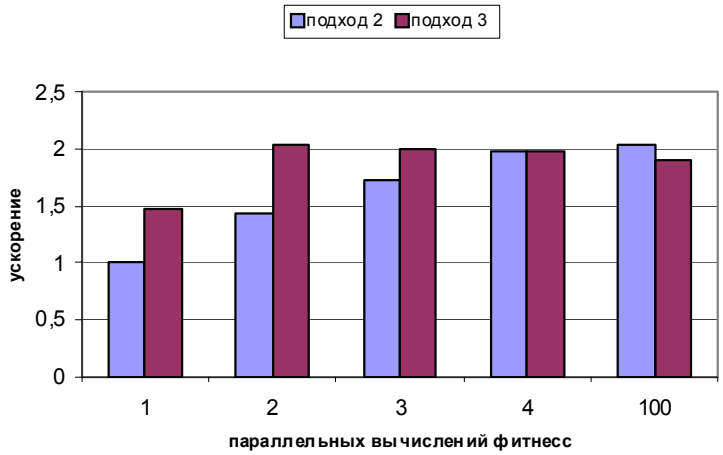


Рис. 3. Среднее ускорение работы модификаций алгоритма

Таблица 3

Время работы модификаций алгоритма для контрольных схем ISCAS-89

параллельных функций	параллельных потоков	время работы, сек.											
		s3271	s3330	s3384	s4863	s5378	s6669	s9234	s13207	s15850	s35932	s38417	s38584
1	1	106	78	94	225	74	300	72	101	158	711	416	491
2	2	63	47	56	131	49	173	69	108	135	446	345	370
3	3	47	38	44	97	43	127	73	111	132	343	319	319
4	4	39	33	37	79	38	103	75	119	129	302	304	389
по числу особей	100	33	30	32	68	41	91	74	127	151	359	358	367

Третий подход является комбинацией первых двух. В этом случае в качестве параллельных потоков оформляются процедуры вычисления оценочных функций, которые сами внутри являются дополнительно распараллеленными. В такой реализации в псевдокоде выше процедура *«ОценитьОсобь»* также будет содержать параллельный код, который соответствует модернизированному псевдокоду первого подхода. В данной реализации также необходим выбор оптимального числа одновременных потоков моделирования. Для экспериментов также использовался генетический алгоритм верификации эквивалентности цифровых схем. Инструментальная платформа и проводимые эксперименты соответствуют предыдущему случаю. Результаты экспериментов приведены в табл.4. Среднее значение ускорения и сравнение с предыдущим методом приведены на диаграмме (рис. 3, столбец «подход 3»). Лучшее ускорение 2.06 раза достигнуто для случая, когда параллельно вычисляются две процедуры *«ОценитьОсобь»*. При этом реальное число вычислительных потоков равно четырём в силу дополнительной параллельности самой функции. Также эта модификация оказалась лучшей в сравнении с предыдущим подходом. Поэтому для неё мы также вычислили характеристики параллелизации. Они приведены в табл. 5. Заметим, что в данном случае наблюдается существенный разброс значений. Например, лучшее достигнутое ускорение равно 3.15 раза, однако в двух худших случаях наблюдается даже замедление работы. Все такие «провалы» соответствуют труднотестируемым схемам, в которых трудно проявить различие в поведении схем и распространить его влияние на внешние выходы. Это сильно снизило среднее значение ускорения до 2.06 раза. Однако и такие результаты показывают, что утилизация ядер вырастает более чем в два раза в сравнении с исходной немодифицированной версией алгоритма.

Таблица 4

Время работы модификаций алгоритма для контрольных схем ISCAS-89

параллельных функций	параллельных потоков	время работы, сек.											
		s3271	s3330	s3384	s4863	s5378	s6669	s9234	s13207	s15850	s35932	s38417	s38584
1 поток	2	57	45	53	121	47	148	71	105	137	477	374	400
2 потока	4	38	33	35	72	39	95	79	119	132	287	304	285
3 потока	6	38	32	36	77	38	103	77	122	132	292	308	297
4 потока	8	35	30	33	71	36	98	77	124	191	288	322	301
по числу особей	200	36	35	36	70	46	97	80	133	154	362	359	368

Таблица 5

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	2,79	2,36	2,69	3,12	1,90	3,15	0,91	0,85	1,20	2,48	1,37	1,72	2,06
E_p	0,70	0,59	0,67	0,78	0,47	0,79	0,23	0,21	0,30	0,62	0,34	0,43	0,51
f	0,14	0,23	0,16	0,09	0,37	0,09	1,13	1,24	0,78	0,20	0,64	0,44	0,46

Выводы

В работе исследуется задача построения параллельных версий генетических алгоритмов генерации идентифицирующих последовательностей цифровых схем. Отмечено два возможных направления решения задачи: реализация новых алгоритмов, либо разработка новых параллельных модификаций существующих алгоритмов, в которых распараллеливаются наиболее ресурсоёмкие процедуры. Во втором направлении выделено и реализовано три подхода. Они заключаются в многопоточной реализации процедур вычисления оценочных функций, которые являются процедурами моделирования поведения схем на заданной входной последовательности. Проведённые машинные эксперименты показали, что можно добиться существенного ускорения работы алгоритмов, несмотря на то, что структура самого генетического алгоритма не является параллельной. При этом дополнительным ограничивающим фактором для повышения быстродействия является внутренняя структура современных многоядерных процессоров.

Литература

1. Goldberg D.E. Genetic Algorithm in Search, Optimization, and Machine Learning / D.E. Goldberg.- N.-Y.: Addison-Wesley, 1989. – 432p.

2. Скобцов Ю. А. Основы эволюционных вычислений / Скобцов Ю. А. – Донецк : ДонНТУ, 2008. – 326 с.

3. Иванов Д. Е. Генетические алгоритмы построения идентифицирующих последовательностей для цифровых схем с памятью / Д. Е. Иванов // Наукові праці Донецького національного технічного університету. Серія: “Обчислювальна техніка та автоматизація”. – Донецьк : ДонНТУ. – 2008. – Випуск 14(129). – С. 97–106.

4. Corno F. A Parallel Genetic Algorithm for Automatic Generation of Test Sequences for Digital Circuits / F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda // Proc/ of International Conference on High-Performance Computing and Networking, Brussels (Belgium), April, Lecture Notes In Computer Science.- 1996. – Vol.1067.- P.454-459.

5. Skobtsov Y.A. Distributed Genetic Algorithm of Test Generation For Digital Circuits/ Y.A. Skobtsov, El-Khatib, D.E. Ivanov // Proceedings of the 10th Biennial Baltic Electronics Conference. – Tallinn Technical University,2006. – P.281-284.

6. Skobtsov Y.A. Distributed Fault Simulation and Genetic Test Generation of Digital Circuits / Y.A. Skobtsov, El-Khatib, D.E. Ivanov // Proceedings of IEEE East-West Design&Test Workshop (EWDT’06). – 2006. – P.89-94.

7. Wirt R. Intel® Software Insight. Multi-core Capability.- USA: Intel Corporation, 2005. – July. – 11 p.

8. Иванов Д. Е. Взаимодействие компонент в распределённых генетических алгоритмах генерации тестов / Д. Е. Иванов, П. А. Чебанов // Наукові праці Донецького національного технічного університету. Серія: “Обчислювальна техніка та автоматизація”. – Донецьк : ДонНТУ. – 2009. – Випуск 16(147). – С. 121–127.

9. Иванов Д. Е. Система моделирования и генерации тестов цифровых схем / Д. Е. Иванов, Ю. А. Скобцов // Наукові праці Донецького державного технічного університету. Серія “Обчислювальна техніка та автоматизація». – Донецьк. – 1999. – Випуск 12. – С. 143–150.

10. Ivanov D.E. Parallel fault simulation on multi-core processors / D.E. Ivanov // Радіоелектронні і комп’ютерні системи.- 2009.- №6 (40).- С.109-112.

11. Иванов Д. Е. Генетический подход проверки эквивалентности последовательностных схем / Д. Е. Иванов // «Радіоелектроніка. Інформатика. Управління». – Запоріжжя, ЗНТУ. – 2009. – № 1(20). – С. 118–123.

12. Brgles F. Combinational profiles of sequential benchmark circuits / F. Brgles, D. Bryan, K. Kozminski // International symposium of circuits and systems, ISCAS-89.- 1989.- P.1929-1934.

13. Гергель В. П. Основы параллельных вычислений для многопроцессорных вычислительных систем : учебное пособие / В. П. Гергель, Р. Г. Стронгин. – Нижний Новгород : Изд-во ННГУ им. Н.И. Лобачевского, 2003. – 184 с.

Надійшла 18.1.2011 р.