УДК 681.518

Д.Е. Иванов

Масштабируемый параллельный генетический алгоритм построения идентифицирующих последовательностей для современных многоядерных вычислительных систем

Рассмотрена задача построения параллельных генетических алгоритмов генерации идентифицирующих последовательностей для цифровых устройств по схеме «мастер–рабочий». Исследована их масштабируемость на системах с большим числом вычислительных ядер.

The task of constructing the parallel genetic algorithms for generating the identifying sequences for synchronous sequential circuits based on the «master-slave» scheme is considered. The problem of their scalability on systems with a great number of cores is investigated.

Розглянуто задачу побудови паралельних генетичних алгоритмів генерації ідентифікуючих послідовностей цифрових пристроїв за схемою «майстер–робітник». Досліджено їх масштабованість на системах з великою кількістю обчислювальних ядер.

Введение. Генетические алгоритмы (ГА) [1–2] заняли прочное место в инструментарии разработчиков цифровых схем. Диапазон их применения очень широк: от оптимального кодирования состояний последовательностных схем [3] до построения идентифицирующих последовательностей [4–5]. Их популярность связана как с прозрачностью стратегии, так и с тем, что они позволяют обрабатывать схемы большой размерности, для которых структурные методы не дают результата.

Генетические алгоритмы построения входных идентифицирующих последовательностей основаны на моделировании работы цифровой схемы. Для оценки потенциальных решений задачи и в зависимости от требуемой точности/скорости работы они могут использовать исправное моделирование и моделирование с неисправностями. В некотором смысле такое построение решений – замена задачи синтеза на итеративную задачу анализа. Процедуры моделирования сами по себе требуют значительных вычислительных ресурсов, в частности времени. Итеративный многократный вызов таких процедур приводит к тому, что ГА построения идентифицирующих последовательностей достаточно медленные, что есть основным их недостатком.

Ключевые слова: генетический алгоритм, последовательностная схема, многоядерные вычислительные системы, параллельные вычисления.

Для решения данной проблемы предложено несколько подходов. В работе [6] разработан параллельный генетический алгоритм (ПГА) построения тестов, основанный на схеме островов. В нем происходит развитие нескольких независимых популяций с немного измененными генетическими параметрами. Такое построение алгоритма позволяет не только повысить скорость работы, но и качество получаемых решений. К недостаткам подхода следует отнести довольно сложную программную реализацию, которая, помимо использования специальных средств параллельного программирования, требует создания протокола взаимодействия компонент, что само по себе нетривиальная задача. Возможен также подход разработки параллельных версий процедур моделирования. Так, в [7] предложена параллельная версия хорошо известного алгоритма моделирования цифровых схем с неисправностями *PROOFS*. Его свойства, в том числе и способность к масштабированию, изучены для различных аппаратных платформ. Недостаток указанных подходов – они разработаны для дорогостоящих вычислительных систем.

В настоящее время стандартом «де-факто» рабочих станций становится применение многоядерных процессоров [8]. В коммерческих образцах процессоров уже сегодня число ядер достигает 4–6. При этом в компании *Intel* ведутся разработки чипов с числом вычислительных ядер до 80-ти, что привлекло интерес

к разработке параллельных алгоритмов для таких рабочих станций.

Авторы уже предлагали параллельные версии ГА для работы на многоядерных рабочих станциях. В отличие от упомянутых выше ПГА они строятся по схеме «*мастер–рабочий*». В частности для алгоритма построения последовательностей верификации эквивалентности заданных схем [9] предложена параллельная версия [10] и исследованы его характеристики по распараллеливанию на 4-ядерной рабочей станции.

Цель данной статьи – дальнейшее изучение свойства масштабируемости параллельных версий ГА на рабочей станции с большим числом вычислительных ядер. Под масштабируемостью программного обеспечения понимают способность к пропорциональному увеличению производительности при увеличении аппаратных средств. В свою очередь под увеличением аппаратных средств следует понимать увеличение числа вычислительных ядер в инструментальной системе. Основной вопрос проведенного исследования: являются ли предлагаемые параллельные версии ГА масштабируемыми, т.е. будет ли дальнейшее увеличение числа ядер в инструментальной системе подтверждено адекватным ростом скорости работы ПГА?

ГА построения идентифицирующих последовательностей

Напомним кратко схему генетического алгоритма. Структурно ГА представляют собой итеративный процесс построения новых популяций из текущей (рис. 1), цель которого – поиск субоптимального решения некоторой оптимизационной задачи. Критерий качества потенциальных решений обычно задается конструктивно в виде оценочной функции. Каждая популяция состоит из некоторого числа потенциальных решений – особей. Алгоритм заканчивает работу либо когда найдено решение, либо достигнуто максимальное число итераций. В ГА построения входных идентифицирующих последовательностей каждая особь представляет собой входную двоичную последовательность. При построении новых особей используются генетические операции: селекция, скрещивание и мутация. Не будем останавливаться подробно на данных моментах, поскольку они неоднократно подробно описывались, например в [11]. Для оценки качества последовательностей используется моделирование работы цифровой схемы на данной последовательности. Используется как исправное моделирование, так и моделирование с неисправностями. Именно эти процедуры определяют большую временную сложность данного класса ГА.

ГА построения входных идентифицирующих последовательностей условно делятся на два класса.

• *Одноуровневые* ГА. В алгоритмах данного класса имеется одна глобальная цель, и, следовательно, строится одна входная последовательность, а цикл построения популяций ГА вызывается только один раз. К алгоритмам данного класса, например, относятся задача построения инициализирующих последовательностей и задача проверки эквивалентности двух заданных последовательностных схем.

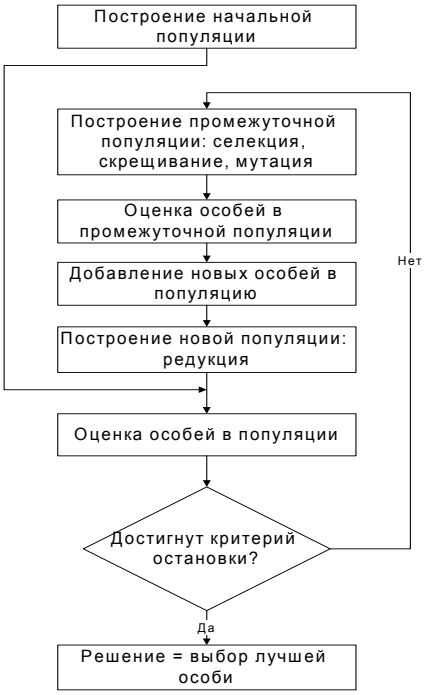

Рис. 1. Укрупненная схема генетического алгоритма

• *Двухуровневые* ГА. В данных алгоритмах итеративно происходит выбор промежуточных целей, для достижения которых вызывается основной цикл ГА. К такому классу относятся

алгоритмы построения тестовых и диагностических последовательностей. В качестве промежуточных целей в данных алгоритмах выступают задача тестирования одной неисправности из полного списка неисправностей (ГА построения тестов) или задача построения диагностической последовательности для выбранного класса неразличимых неисправностей (ГА построения диагностических последовательностей).

Для изучения свойств параллелизации из всех ранее разработанных алгоритмов выберем одноуровневый ГА построения последовательностей, которые проверяют эквивалентность двух заданных последовательностных схем [9]. На его основе в следующем разделе представим параллельную версию алгоритма. Отметим также, выбор данного алгоритма означает, что будут исследоваться свойства параллельности генетических алгоритмов, которые включают процедуры исправного моделирования цифровых схем. Изучение аналогичных свойств ПГА, включающих процедуры моделирования схем с неисправностями, требует отдельного исследования.

Параллельные версии ГА

Схемы построения параллельных ГА

Они делятся на два больших класса: «*островов*» и «*мастер–рабочий*».

Схема «*островов*» предполагает развитие нескольких относительно независимых популяций. Они проводят обмен лучшими особями через некоторое число поколений. Данная схема наиболее часто применяется в кластерных архитектурах с разделяемой памятью. Эффективность схемы «островов» зависит от очень большого числа параметров, среди которых:

- топология, определяющая, какие подпопуляции будут считаться соседними и, соответственно, между какими островами будет возможен обмен особями;
- время изоляции, определяющее, сколько эпох ГА не будет проводиться 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.

Отметим, 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 описания способа их взаимодействия. Точная и корректная реализация такого взаимодействия различных копий ГА в модели островов – сложная задача [11]. Для этого реализуются протоколы, в которых фиксируются точки обмена информацией между островами и сервером.

Схема «*мастер–рабочий*» построения ПГА реализует только один цикл развития популяций, осуществляемый на процессоре, который называется «*мастер*». На «*рабочие*» процессоры передается вычислительная нагрузка, связанная с оценкой особей. Эта работа как раз и может быть выполнена параллельно. Хотя организация параллельности по данной схеме требует некоторых ресурсов, выигрыш может оказаться весьма существенным. Особенно это заметно, если оценка вычисляется по сложным формулам либо, как в нашем случае, на основании моделирования. Данная схема более проста и потому выбрана для реализации

Построение параллельного ГА по схеме «мастер–рабочий»

Из описания структуры ГА видно, что он носит итеративный характер. При этом работа на текущей итерации существенно зависит от результатов, полученных в итоге работы предыдущей итерации. Такая последовательностная схема работы не позволяет напрямую построить параллельную версию алгоритма. Однако в ГА построения идентифицирующих последовательностей все-таки можно выделить достаточно крупные фрагменты, которые являются независимыми и позволяют параллельную реализацию. Это процедуры вычисления оценочных функций. В рассматриваемых алгоритмах эти процедуры есть процедуры моделирования работы цифровых схем, что пред-

полагает достаточно высокую вычислительную нагрузку. Благодаря этому, при работе алгоритма большая часть вычислительной нагрузки приходится именно на данные процедуры моделирования, и лишь небольшая их часть – на процедуры построения новых популяций, которые в принципе не распараллеливаются. Основываясь на этом можно предположить, что распараллеливания процедур вычисления оценок особей позволит достичь приемлемого масштабирования алгоритма.

Для иллюстрации механизма распараллеливания, применяемого в ГА данного типа, рассмотрим подробно построение процедуры вычисления оценок особей. Вначале рассмотрим последовательную реализацию данной процедуры. Ее псевдокод приведен ниже.

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

Процедура «*ОценитьОсобь*», вложенная во внешний цикл, в непараллельной модели вычислений для особи i выполняется только после завершения предыдущей итерации $i-1$. Если условно принять за один модельный такт время, необходимое для вычисления оценки одной особи в популяции, то схематично порядок вычисления при такой организации процесса можно изобразить так (рис. 2,*а*).

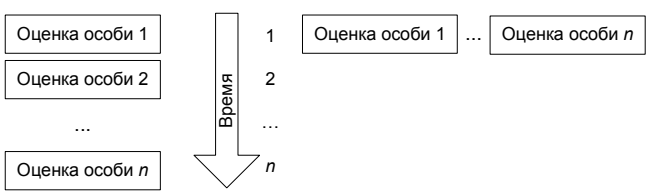

Рис. 2. Способы вычисления оценок особей в популяции: *а* – последовательный; *б* – параллельный

Очевидно, что для последовательной модели данный порядок вычислений естественен. Легко видеть также, что любые две последовательные процедуры «*ОценитьОсобь*» являются независимыми по данным, поскольку моделирование выполняется для разных входных последовательностей, и, следовательно, представляют собой независимые ветви программы. Поэтому можно организовать параллельное выполнение нескольких таких функций.

Основным средством организации параллельных вычислений в многопроцессорных системах с общей памятью есть потоки (*threads*). Данную парадигму поддерживают все современные среды программирования. Методами работы с потоками являются: создание, удаление, приостановка, запуск на выполнение. Если оформить процедуру оценки особи в виде потокового класса, то приведенный ранее фрагмент будет выглядеть так:

```
ОценитьПопуляцию(Популяция, ЧислоОсобей
{
  for( i=0 ; i<ЧислоОсобей ; i++ )
  {
    ОценитьОсобь->СоздатьПоток(Популя-
    ция[i]);
4    ОценитьОсобь->Выполнить();
    ОценитьОсобь->ЖдатьЗавершения();
    ОценитьОсобь->УдалитьПоток();
  }
}
```

Схематически параллельное вычисление оценок особей в популяции представлено на рис. 2,*б*. Организация такого процесса состоит в том, чтобы создать одновременно несколько экземпляров потокового класса *«ОценитьОсобь»*, запустить их на выполнение, передав индивидуальные входные параметры, и далее ожидать окончания выполнения для каждого экземпляра класса. Число особей в таких реализациях ПГА обычно гораздо больше числа вычислительных процессоров в системе (4–6 – в текущих маркетинговых процессорах *Intel*, до 64 – в специализированных системах). Поскольку предположительно один вычислительный поток будет загружать один процессор (одно ядро) в системе, то необходимо одновременно выполнять такое число потоков, которое равно числу процессоров. Таким образом, в коде появится дополнительный вложенный цикл. Приведем псевдокод параллельной реализации в терминах потоков.

```
ОценитьПопуляцию(Популяция, ЧислоОсобей)
{
  for( int i=0 ; i<ЧислоОсобей/ЧислоПо-
       токов ; i++ )
```

```
  {
    j=i*ЧислоПотоков;
    выполнять_параллельно_для j, j+1, …
     …, j+ЧислоПотоков-1
    {
      ОценитьОсобь->СоздатьПоток(Попу-
       ляция[j]);
      ОценитьОсобь->Выполнить();
    }
    ЖдатьОкончанияПотоков j, j+1, …
     …, j+ЧислоПотоков-1;
    УдалитьПотоки();
  }
}
```

Переменная «*ЧислоПотоков*» показывает, сколько вычислительных потоков будет выполняться одновременно. Обычно для многоядерных систем рекомендуется выбирать число потоков, равное числу вычислительных ядер системы [13], однако покажем, что в нашем случае это не так.

Эксперименты и результаты

Благодаря компании *Intel*, авторам предоставили доступ к лаборатории *MTL* (*Manycore Testing Lab*). Дистанционно была организована работа с рабочей станцией под управлением операционной системы *MS Windows Server* 2008 *R*2. Многоядерная ВС содержала два процессора *Intel*(*R*) *Xeon CPU X*5650 с частотой 2.67 Ггц (каждый по шесть вычислительных ядер), технология *HyperThreading* – выключена, объем оперативной памяти 16 Гб. Функция *GetSystemInfo*() также показывала наличие 12 вычислительных ядер в системе.

В таких условиях проведены машинные эксперименты, которые дадут ответ на следующие вопросы:

- является ли предложенная структура параллельного алгоритма масштабируемой;
- каков предел масштабируемости данной версии алгоритма;
- сколько вычислительных потоков необходимо запускать для наилучшей загрузки системы, и как это число связано с числом вычислительных ядер.

При проведении экспериментов использовались контрольные схемы *ISCAS*–89 [14], из которых сразу были исключены схемы малой размерности ввиду того, что для них время моделирования составляет порядка нескольких секунд и построение параллельных версий алгоритмов не актуально.

Первые же эксперименты показали, что нет необходимости привязывать вычислительные потоки к ядрам. Это в несколько раз уменьшало производительность параллельной версии алгоритма. Данный факт согласуется с выводами [15]: в параллельной вычислительной среде наибольшее ускорение получается в случае такой привязки потоков, которая допускает их миграцию между ядрами внутри одного сокета либо вычислительного узла. Поскольку наш вычислительный сервер фактически представляет собой один вычислительный узел с 12 ядрами, то наибольшее ускорение должно быть получено без привязки потоков к ядрам.

Одиночный запуск ПГА эмулировал работу генетического алгоритма на фиксированном числе итераций (число поколений = 50), что позволяло измерять ускорение работы ГА.

При проведении машинных экспериментов для выбранного множества схем выполнялись многократные запуски параллельной версии ГА с различным числом одновременных вычислительных потоков (переменная «*ЧислоПотоков*» в коде выше). Условно разделим эксперименты на две серии. В первой серии число потоков изменялось от одного до 12, где верхняя граница определялась числом физических вычислительных ядер. Во второй серии экспериментов число потоков изменялось от одного до 128. Данная серия экспериментов проводилась с целью выяснения максимально возможного ускорения ПГА. В качестве последовательной версии алгоритма выбран ПГА, в котором запускался один вычислительный поток оценки особи. Скорость работы такой реализации, вообще говоря, отличается от истинно последовательной версии алгоритма [7] и уменьшается на один–четыре процента, что связано с накладными расходами на организацию параллельных вычислений. Однако такую погрешность вполне можно считать приемлемой, поскольку авторы не имели возможности в инструментальной

Таблица 1. Ускорение работы ПГА в зависимости от числа потоков

Число потоков	Ускорение для заданной схемы, раз (для строки 1 приведено абсолютное время работы в сек.)											
	*s*3271	*s*3330	*s*3384	*s*4863	*s*5378	*s*6669	*s*9234	*s*13207	*s*15850	*s*35932	*s*38584	средн.
1	162	124	143	342	107	406	84	124	197	914	633	1,00
2	1,78	1,80	1,72	1,77	1,65	1,77	1,47	1,43	1,56	1,79	1,70	1,68
3	2,49	2,48	2,42	2,59	2,18	2,53	1,79	1,75	2,07	3,14	2,78	2,38
4	3,60	3,26	3,11	4,38	2,82	3,44	2,00	1,94	2,53	4,57	4,28	3,27
5	4,15	3,76	3,67	4,75	3,24	4,10	2,21	2,07	2,94	5,25	5,10	3,75
6	5,59	4,43	4,21	6,71	3,45	5,21	2,27	2,10	3,13	6,05	5,36	4,41
7	5,59	4,43	4,47	6,58	3,57	5,41	2,27	2,10	3,13	6,35	5,65	4,50
8	7,36	4,77	4,77	7,60	3,69	6,44	2,27	2,14	3,18	6,67	5,92	4,98
9	6,32	4,96	5,11	6,98	3,69	6,44	2,27	2,10	3,13	6,87	5,70	4,86
10	8,53	5,17	5,96	9,00	3,82	7,66	2,27	2,14	3,23	7,95	6,46	5,65
11	6,75	5,17	5,50	7,60	3,82	7,12	2,21	2,14	3,13	7,55	6,09	5,19
12	9,00	5,64	6,22	9,50	3,96	7,81	2,27	2,18	3,18	8,39	6,53	5,88

системе отключить от работы 11 ядер для получения абсолютно точных цифровых данных.

Объем числовых результатов, полученных при проведении экспериментов, очень велик. Приведем значения для ускорения работы ПГА. На основе данной информации и известных формул можно вычислить остальные характеристики параллельной версии алгоритма: эффективность использования ядер и долю последовательного кода [16]. Числовые результаты первой серии экспериментов приведены в табл. 1. Высокие числовые значения превосходят данные, приведенные в [7]. Однако необходимо отметить, что в указанной работе приводятся данные по ускорению для параллельной версии программы моделирования с неисправностями. Авторам в настоящее время не известны аналогичные результаты для параллельных генетических алгоритмов. Графики ускорения работы ПГА для первой серии экспериментов показаны на рис. 3.

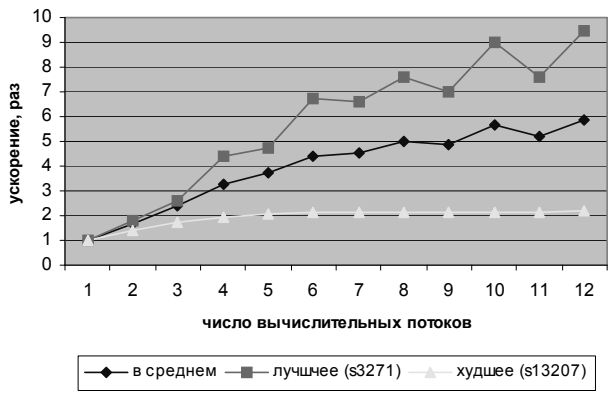

Рис. 3. Ускорение работы ПГА в зависимости от числа вычислительных потоков. Серия 1 экспериментов

Анализ результатов первой серии экспериментов (табл. 2, колонка «серия 1») показывает, что почти для всех схем максимальное ускорение работы ПГА достигнуто в правом конце диапазона изменений числа потоков. Это привело к предположению, что последующее наращивание числа потоков позволит и далее расти параметру ускорения работы.

С этой целью проведена вторая серия экспериментов, в которой число потоков изменялось от одного до 128. Большой объем полученных цифровых данных не позволяет привести их в данной статье. На рис. 4 показан график ускорения работы ПГА в трех случаях данной серии экспериментов. Сравнение с гра-

Таблица 2. Максимальное достигнутое ускорение для контрольных схем

Схема	Серия 1, 12 потоков максимум		Серия 2, 128 потоков максимум	
	Макс. достигнутое ускорение	Минимальное число потоков для достигнутого ускорения	Макс. достигнутое ускорение	Минимальное число потоков для достигнутого ускорения
*s*3271	9.00	12	13.50	105
*s*3330	5.64	12	9.54	115
*s*3384	6.22	12	11.00	100
*s*4863	9.50	12	13.15	119
*s*5378	3.96	12	4.12	18
*s*6669	7.81	12	12.30	128
*s*9234	2.27	7	2.33	17
*s*13207	2.18	12	2.18	12
*s*15850	3.23	10	3.23	12
*s*35932	8.39	12	10.51	125
*s*38584	6.53	12	7.11	25
средн.	5.88	11.36	8.09	80

фиком на рис. 3 показывает, что максимальное достигнутое ускорение существенно выросло. Таблица 2 содержит некоторые числовые данные для сравнения двух серий экспериментов, из которых видно, что максимально достигнутое ускорение увеличилось с девяти до 13,5 раз. Причем для трех схем из одиннадцати (*s*3271, *s*4863, *s*6669) это ускорение было выше, чем число вычислительных ядер в системе. Для одной из схем (*s*3384) дальнейшее увеличение числа потоков позволило увеличить полученное в серии 1 значение ускорения еще в 1,77 раза: с 6.22 до 11 раз. Только для одной из схем (*s*6669) во второй серии экспериментов максимальное ускорение достигнуто при максимальном числе потоков 128. Это говорит о том, что для набора схем в целом нет необходимости далее увеличивать число параллельных вычислительных потоков. Для схем с низким значением параметра ускорения в первой серии экспериментов дальнейший рост числа потоков практически не улучшает ситуацию. А для двух схем (*s*13207, *s*15850) лучшие значения ускорения относятся к первой серии экспериментов.

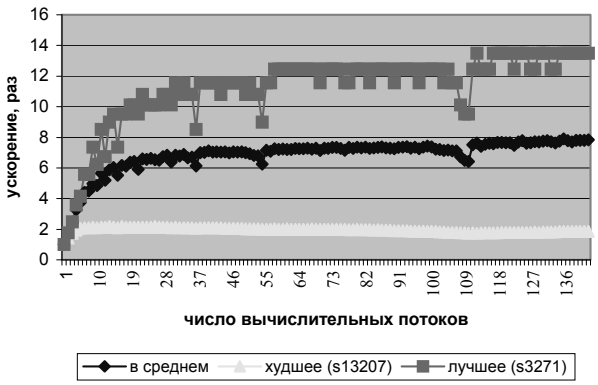

Рис. 4. Ускорение работы ПГА в зависимости от числа вычислительных потоков. Серия 2 экспериментов

Таким образом, приведенные числовые результаты показывают, что для максимальной утилизации вычислительных ядер в системе, число одновременно запущенных потоков должно быть существенно больше, чем число ядер. Однако точное определение данного числа все еще не представляется возможным.

Заключение. В статье изучаются свойства распараллеливания генетических алгоритмов построения входных идентифицирующих последовательностей. Для предложенной параллельной версии алгоритма верификации эквивалентности схем проведены машинные эксперименты на параллельной рабочей станции с общей памятью, содержащей 12 вычислительных ядер. Данные эксперименты показали отличную масштабируемость для некоторых контрольных схем, тогда как для других она оказалась крайне низкой, причины чего, очевидно, кроются во внутренней структуре данных схем. В среднем достигнуто ускорение работы алгоритма в 8.09 раза. Причины ограниченности этого показателя требуют дальнейшего изучения. Для достижения максимального ускорения необходимо выбирать число параллельных потоков существенно больше, чем число вычислительных ядер в системе.

Авторы благодарят компанию *Intel*®, а также ее подразделение *Intel® Software Network* за предоставленную возможность доступа к 12-ядерной рабочей станции лаборатории *Many-core Testing Lab*. Выражаем личную благодарность Майку Пирсу (*Mike Pearce*), а также Питеру Гинсбику (*Peter Hinsbeeck*) за оказанную техническую поддержку во время сессии доступа и после нее.

1. *Goldberg D.E*. Genetic Algorithm in Search, Optimization, and Machine Learning. – Addison-Wesley, 1989. – 432 p.
2. *Скобцов Ю.А*. Основы эволюционных вычислений. – Донецк: ДонНТУ, 2008. – 326 с.
3. *Wu X*., *Pedram M.* Low power sequential circuit design by using priority encoding and clock gating // Proc. of the 2000 Intern. Symp. on Low power electronics and design, Rapallo, Italy. – 2000. – P. 143–148.
4. *GATTO*: a Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits / F. Corno, P. Prinetto, M. Rebaudengo et al. // IEEE Transactions on Computer-Aided Design, August 1996. – **15**, № 8. – P. 943–951.
5. *Evolutionary* approach to the functional test generation for digital circuits / Yu.A. Skobtsov, D.E. Ivanov, V.Y. Skobtsov et al. // Proc. of 9th Biennial Baltic Electronics Conf., BEC 2004, Tallinn, October 2004. – Tallinn Univ. of Techn. – 2004. – P. 229–232.
6. *A Parallel* Genetic Algorithm for Automatic Generation of Test Sequences for Digital Circuits / F. Corno, P. Prinetto, M. Rebaudengo et al. // Intern. Conf. on High-Performance Computing and Networking, Brus-

sels (Belgium), April // Lecture Notes In Comp. Sci. – 1996. – **1067**. – P. 454–459.
7. *A parallel* algorithm for fault simulation based on PROOFS / S. Parker, P. Banerjee, J. Patel // Proc. IEEE Int. Conf. Comp. Design. – 1995. – P. 616–621.
8. *Wirt R*. Intel® Software Insight. Multi-core Capability / USA: Intel Corporation, 2005, July. – 11 p.
9. *Иванов Д.Е.* Генетический подход проверки эквивалентности последовательностных схем // Радіоелектроніка. Інформатика. Управління. – 2009. – № 1(20). – С. 118–123.
10. *Иванов Д.Е.* Алгоритм параллельного вычисления оценок особей при верификации эквивалентности последовательностных схем // Проблемы информационных технологий. – 2009.– № 1(005). – С. 105–112.
11. *Иванов Д.Е.* Генетические алгоритмы построения идентифицирующих последовательностей для цифровых схем с памятью // Наук. пр. Донецького нац. техн. ун-ту. Сер.: Обчислювальна техніка та автоматизація. – 2008. – **14**(129). – С. 97–106.
12. *Иванов Д.Е.* Взаимодействие компонент в распределенных генетических алгоритмах генерации тестов // Там же. – 2009. –**16**(147). – С. 121–127.
13. *Gillespie M*. Масштабирование программных архитектур для многоядерных вычислительных систем будущего. – http://software.intel.com/ru-ru/articles/scaling-software-architectures-for-the-future-of-multi-core-computing/
14. *Combinational* profiles of sequential benchmark circuits / F. Brgles, D. Bryan, K. Kozminski // Intern. Symp. of circuits and syst., ISCAS–89. – 1989. – P. 1929–1934.
15. *Методы* привязки параллельных процессов и потоков к многоядерным узлам вычислительных систем / С.П. Копысов, А.К. Новиков, Л.Е. Тонков и др. // Вест. Удмурт. ун-та. – 2010. – № 1. – С. 123–132.
16. *Гергель В.П.*, *Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебн. пособие. – Н. Новгород: Изд-во ННГУ им. Н.И. Лобачевского. – 2003. – 184 с.

Поступила 20.10.2010
Тел. для справок: (062) 311-6795 (Донецк)
E-mail: *ivanov@iamm.ac.donetsk.ua*,
http://*www.iamm.ac.donetsk.ua*/*ru*/*employees*/*e*15/