УДК 681.518

[1]МАСШТАБИРУЕМЫЙ ПАРАЛЛЕЛЬНЫЙ ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ПОСТРОЕНИЯ ИДЕНТИФИЦИРУЮЩИХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ ДЛЯ СОВРЕМЕННЫХ МНОГОЯДЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Д.Е. Иванов, Институт прикладной математики и механики НАН Украины

ivanov@iamm.ac.donetsk.ua

1. 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 тем фактом, 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 являет заменой 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 являются достаточно медленными алгоритмами, что является основным их недостатком.

Для решения данной проблемы предложено несколько подходов. В работе [6] разработан параллельный генетический алгоритм (ПГА) построения тестов, который основан на схеме

[1]Данная работа была поддержана компанией Intel, в частности её структурными подразделениями Intel Software network и Manycore Testing Lab, которые предоставили бесплатный доступ к 12-ядерной рабочей станции.

островов. В нём происходит развитие нескольких независимых популяций с немного изменёнными 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 создание протокола взаимодействия компонент, что само по себе является нетривиальной задачей. Возможен также подход, когда разрабатываются параллельные версии процедур моделирования. 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. Недостатком указанных подходов является тот факт, что 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-и. Это подхлестнуло 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.

Целью данной статьи является 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 нашем случае под увеличением аппаратных средств будем понимать увеличение числа вычислительных ядер в

инструментальной системе. Основной вопрос проведённого 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?

Данная статья имеет следующую структуру. В первом разделе указывается на актуальность исследований и связь с другими работами в этой области. Во втором разделе кратко описываются схемы ГА построения идентифицирующих последовательностей. В следующем разделе рассмотрено построение параллельной версии ГА верификации эквивалентности схем. Четвёртый раздел посвящён описанию проведённых машинных экспериментов и их краткому анализу. В заключении делаются выводы и указываются направления дальнейших исследований.

2. ГА построения идентифицирующих последовательностей

Кратко напомним схему генетического алгоритма. Структурно генетические алгоритмы представляют собой итеративный процесс построения новых популяций из текущей (рис.1), целью которого является поиск субоптимального решения некоторой оптимизационной задачи. Критерий качества потенциальных решений обычно задаётся 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. Мы не будем в данной статье останавливаться подробно на данных моментах, поскольку они неоднократно подробно описывались в статьях, 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.

Генетические алгоритмы построения входных идентифицирующих последовательностей условно делятся на два класса.

1) 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.

2) Двухуровневые ГА. В данных алгоритмах итеративно происходит выбор промежуточных целей, для достижения которых вызывается основной цикл ГА. К данному 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 нами ранее разработанных алгоритмов мы выбрали одноуровневый ГА построения последовательностей, которые проверяют эквивалентность двух заданных последовательностных схем [9]. На его основе в следующем разделе будет представлена параллельная версия алгоритма. Отметим также, что выбор данного алгоритма означает, что будут исследоваться свойства параллельности генетических алгоритмов, которые включают процедуры исправного моделирования цифровых схем. Изучение аналогичных свойств ПГА, которые включают процедуры моделирования схем с неисправностями, требует отдельного исследования.

3. Параллельные версии ГА

3.1 Схемы построения параллельных ГА

Схемы построения ПГА делятся на два больших класса: «островов» и «мастер-рабочий».

Схема «островов» предполагает, что происходит развитие нескольких относительно независимых популяций. Они производят 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, которая определяет, какие подпопуляции будут считаться соседними и, соответственно, между какими островами будет возможен обмен особями;
- 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 описания способа их взаимодействия. Точная и корректная реализация такого взаимодействия различных копий ГА в модели островов является сложной задачей [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, который осуществляется на процессоре, который называется «мастер». На «рабочие» процессоры передаётся вычислительная нагрузка, которая связана 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 будет заметно в том случае, если оценка вычисляется по сложным формулам, либо, как в нашем случае, на основании моделирования. Данная схема является более простой и выбрана нами для реализации. В следующем разделе будет представлен механизм построения по данной схеме параллельной версии генетического алгоритма из исходной.

3.2 Построение параллельного ГА по схеме «мастер-рабочий»

Из описания структуры ГА в разделе 2 видно, что он носит итеративный характер. При этом работа на текущей итерации существенно зависит от результатов, которые получены в результате работы предыдущей итерации. Такая последовательностная схема работы не позволяет напрямую построить параллельную версию алгоритма. Однако в ГА построения идентифицирующих последовательностей всё-таки можно выделить достаточно крупные фрагменты, которые являются независимыми и позволяют параллельную реализацию. Это процедуры вычисления оценочных функций. В рассматриваемых алгоритмах эти процедуры являются процедурами моделирования работы цифровых схем, что предполагает достаточно высокую вычислительную нагрузку. Благодаря этому, при работе алгоритма большая часть вычислительной нагрузки приходится именно на данные процедуры моделирования, и лишь небольшая часть приходится 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, который применяется нами в ГА данного типа, рассмотрим подробно построение процедуры вычисления оценок особей. Вначале рассмотрим последовательную реализацию данной процедуры. Её псевдокод приведён ниже.

```
ОценитьПопуляцию(Популяция, ЧислоОсобей)
{
  for( i=0 ; i<ЧислоОсобей ; i++ )
```

```
  {
    ОценитьОсобь(Популяция[i]);
  }
}
```

Процедура *«ОценитьОсобь»*, которая вложена во внешний цикл, в непараллельной модели вычислений для особи *i* будет выполняться только после того, как завершена предыдущая итерация *i-1*. Если условно принять за один модельный такт время, которое необходимо для вычисления оценки одной особи в популяции, то схематично порядок вычисления при такой организации процесса можно изобразить как на рис.2а. Очевидно, что для последовательной модели данный порядок данный порядок вычислений является естественным. 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 являются 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 приведённый выше фрагмент будет выглядеть следующим образом.

```
ОценитьПопуляцию(Популяция, ЧислоОсобей)
{
  for( i=0 ; i<ЧислоОсобей ; i++ )
  {
```

```
    ОценитьОсобь->СоздатьПоток(Популяция[i]);
    ОценитьОсобь->Выполнить();
    ОценитьОсобь->ЖдатьЗавершения();
    ОценитьОсобь->УдалитьПоток();
  }
}
```

Схематически параллельное вычисление оценок особей в популяции изображено на рис.2б. Организация такого процесса будет состоять 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 мы предполагаем, что один вычислительный поток будет загружать один процессор (одно ядро) в системе, то необходимо одновременно выполнять такое число потоков, которое рано числу процессоров. Таким образом, в коде выше появится дополнительный вложенный цикл. Псевдокод параллельной реализации в терминах потоков приведён ниже.

```
ОценитьПопуляцию(Популяция, ЧислоОсобей)
{
  for( int i=0 ; i<ЧислоОсобей/ЧислоПотоков ; i++ )
  {
    j=i*ЧислоПотоков;
    выполнять_параллельно_для j, j+1, … , j+ЧислоПотоков-1
    {
```

```
        ОценитьОсобь->СоздатьПоток(Популяция[j]);
        ОценитьОсобь->Выполнить();
      }
      ЖдатьОкончанияПотоков j, j+1, … , j+ЧислоПотоков-1;
      УдалитьПотоки();
    }
}
```

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

4. Эксперименты и результаты

Благодаря компании Intel авторам был предоставлен доступ к лаборатории MTL (Manycore Testing Lab). Дистанционно была организована работа с рабочей станцией под управлением операционной системы MS Windows Server 2008 R2. Многоядерная ВС содержала два процессора Intel(R) Xeon CPU X5650 с частотой 2.67Ггц (каждый по 6 вычислительных ядер), технология HyperThreading – выключена, объем оперативной памяти 16 Гб. Функция GetSystemInfo() также показывала наличие 12 вычислительных ядер в системе.

В таких условиях были проведены машинные эксперименты, которые должны дать 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 является актуальным.

Первые же эксперименты показали, что нет никакой 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). Мы условно разделили эксперименты на две серии. В первой серии число потоков изменялось от 1 до 12, где верхняя граница определялась числом физических вычислительных ядер. Во второй серии экспериментов число потоков изменялось от 1 до 128. Данная серия экспериментов проводилась с целью выяснить максимальное возможное ускорение ПГА. В качестве последовательной версии алгоритма был 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 1-4%, что связано с накладными расходами на организацию параллельных вычислений. Однако такую погрешность вполне можно считать приемлемой, поскольку авторы не имели никакой

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

Объём числовых результатов, полученных в результате проведения экспериментов очень велик. Поэтому мы будем, в основном, приводить 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. Данные числовые значения являются очень высокими и превосходят данные, приведенные в [7]. Однако, следует 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 в лучшем, худшем и среднем случаях для первой серии экспериментов приведены на рис.3.

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 заставило сделать предположение, что дальнейшее наращивание числа потоков позволит и далее расти параметру ускорения работы.

С этой целью мы провели вторую серию экспериментов, в которой число потоков изменялось от 1 до 128. Большой объём полученных цифровых данных не позволяет привести их в данной статье. На рис.4 приведён график ускорения работы ПГА в лучшем, худшем и среднем случаях в данной серии экспериментов. Сравнение с графиком на рис.3 показывает, что максимальное достигнутое ускорение существенно выросло. Табл.2 содержит некоторые числовые данные для сравнения двух серий экспериментов. Видно, что максимальное достигнутое ускорение увеличилось с 9 до 13.5 раз. Причём для трёх схем из одиннадцати (s3271, s4863, s6669) это ускорение было выше, чем число вычислительных ядер в системе. Для одной из схем (s3384) дальнейшее увеличение числа потоков позволило увеличить полученное

в серии 1 значение ускорения ещё в 1.77 раза: с 6.22 до 11 раз. Только для одной из схем (s6669) 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 (s13207, s15850) лучшие значения ускорения относятся к первой серии экспериментов.

Таким образом, приведённые 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 всё ещё не представляется возможным.

Выводы

В статье изучаются свойства распараллеливания генетических алгоритмов построения входных идентифицирующих последовательностей. Для предложенной параллельной версии алгоритма верификации эквивалентности схем были проведены машинные эксперименты на параллельной рабочей станции с общей памятью, которая содержала 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. Также для достижения максимального ускорения необходимо выбирать число параллельных потоков существенно больше, чем число вычислительных ядер в системе.

Благодарности

Авторы благодарят компанию Intel®, а также её подразделение Intel® Software Network за

предоставленную возможность доступа к 12-ядерной рабочей станции лаборатории Manycore Testing Lab. Мы выражаем личную благодарность Майку Пирсу (Mike Pearce) а также Питеру Гинсбику (Peter Hinsbeeck) за оказанную техническую поддержу во время сессии доступа и после неё.

Литература

1. Goldberg, D.E. Genetic Algorithm in Search, Optimization, and Machine Learning / D.E. Goldberg.- Addison-Wesley, 1989.- 432p.
2. Скобцов, Ю.А. Основы эволюционных вычислений / Ю.А. Скобцов.– Донецк: ДонНТУ, 2008.- 326с.
3. Wu, X. Low power sequential circuit design by using priority encoding and clock gating / X. Wu, M. Pedram // Proceedings of the 2000 international symposium on Low power electronics and design, Rapallo, Italy.- 2000.- P.143-148.
4. Corno, F. GATTO: a Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits / F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda // IEEE Transactions on Computer-Aided Design, August 1996.- Vol. 15, №8.- P.943-951.
5. Skobtsov, Yu.A. Evolutionary approach to the functional test generation for digital circuits / Y. A. Skobtsov, D. E. Ivanov, V. Y. Skobtsov, R. Ubar // In Proc. of 9th Biennial Baltic Electronics Conf., BEC 2004, Tallinn, October 2004 / Tallinn Univ. of Techn.- 2004.- P.229-232.
6. Corno, F. A Parallel Genetic Algorithm for Automatic Generation of Test Sequences for Digital Circuits / F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda // International Conference on High-Performance Computing and Networking, Brussels (Belgium), April, Lecture Notes In Computer Science.- Vol.1067.- 1996.- P.454-459.
7. Parker, S. A parallel algorithm for fault simulation based on PROOFS / S. Parker, P. Banerjee, J. Patel // Proc. IEEE Int. Conf. Computer Design.- 1995.- P.616-621.

8. Intel® Software Insight. Multi-core Capability / R. Wirt.- USA: Intel Corporation, 2005, July.- 11p.
9. Иванов, Д.Е. Генетический подход проверки эквивалентности последовательностных схем / Д.Е. Иванов // «Радіоелектроніка. Інформатика. Управління».- Запоріжжя, ЗНТУ.- 2009.- №1(20).- С.118-123.
10. Иванов, Д.Е Алгоритм параллельного вычисления оценок особей при верификации эквивалентности последовательностных схем / Д.Е. Иванов // Проблемы информационных технологий, 2009.- №1(005).- С.105-112. (0.8 д.а.)
11. 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. - Випуск 14(129).- 2008.- С.97-106.
12. 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 .- Випуск 16(147).- 2009.- С.121-127.
13. Gillespie, M. Масштабирование программных архитектур для многоядерных вычислительных систем будущего [Электронный ресурс] / Matt Gillespie // Intel® Software Network.- Режим доступа: http://software.intel.com/ru-ru/articles/scaling-software-architectures-for-the-future-of-multi-core-computing/.- Загл. с экрана.- (1.06.2009)
14. 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.
15. Копысов, С.П.Методы привязки параллельных процессов и потоков к многоядерным узлам вычислительных систем / С.П. Копысов, А.К. Новиков, Л.Е. Тонков, Д.Е. Береснев // Вестник Удмуртского университета.- 2010.- №1.- С.123-132.

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