Информационные технологии

Вестник Нижегородского университета им. Н.И. Лобачевского, 2011, № 3 (2), с. 187–193

187

Параллельное тестирование больших автоматных моделей

УДК 004.415.53

ПАРАЛЛЕЛЬНОЕ ТЕСТИРОВАНИЕ БОЛЬШИХ АВТОМАТНЫХ МОДЕЛЕЙ\*

© 2011 г. И.Б. Бурдонов, С.Г. Грошев, А.В. Демаков, А.С. Камкин, А.С. Косачев, А.А. Сортов

Институт системного программирования РАН, Москва

[sortov@ispras.ru](mailto:sortov@ispras.ru)

Поступила в редакцию 20.01.2011

Технология тестирования UniTESK позволяет создавать качественные тесты сложных систем на соответствие формальным моделям требований к ним. В большинстве случаев выполнение таких тестов требует значительного времени. Представлено расширение технологии UniTESK, позволяющее эффективно распараллеливать выполнение теста на вычислительном кластере.

Ключевые слова: функциональное тестирование, формальные спецификации, модели аппаратуры, распределенные системы, распределенное тестирование, конечные автоматы.

Введение

Статья описывает расширение технологии тестирования UniTESK [1], использующее воз- можности вычислительных кластеров для со- кращения времени тестирования сложных си- стем на соответствие формальным моделям требований к ним.

В технологии UniTESK тестируемая система

(реализация) представляется в виде модельного конечного автомата (МКА), входные символы которого соответствуют тестовым стимулам, а выходные символы – выдаваемым тестируемой системой реакциям. Процесс тестирования за- ключается в прохождении маршрутов по графу состояний МКА (далее называемому «модель- ным графом» или просто «графом»). При этом на каждом переходе автоматически осуществ- ляются проверки на соответствие наблюдаемого поведения (реакции) тестируемой системы за- данным требованиям, которые описаны в виде спецификации. Спецификация определяет класс удовлетворяющих ей реализаций, модельные графы которых являются различными подгра- фами графа спецификации. Граф спецификации задается неявно пред- и постусловиями стиму- лов. Поэтому при тестировании граф реализа- ции строится по мере его обхода; обход графа завершается после прохождения всех его дуг (когда обнаружено, что ни из одной известной

\*

Статья рекомендована к печати программным ко-

митетом Международной суперкомпьютерной конфе- ренции «Научный сервис в сети Интернет: суперком- пьютерные центры и задачи».

вершины не выходит непройденная дуга) или обнаружения ошибки, не позволяющей продол- жать тестирование.

Подробнее о способе тестирования с помо- щью конечных автоматов, используемом в тех- нологии UniTESK, можно прочитать в [1, 2]. Такой способ даёт хорошее покрытие требова- ний к тестируемой системе. Однако размер мо- дельного графа может быть очень большим, даже после применения различных способов его сокращения [3]. В результате обход МКА на одной машине может длиться недопустимо долго.

Чтобы ускорить процесс тестирования, со- храняя при этом достоинства технологии UniTESK (в частности, автоматическое обеспе- чение полноты обхода модельного графа и ав- томатическую оценку достигнутого покрытия формальной модели тестируемой системы), предлагается использовать возможности, предо- ставляемые распределёнными вычислительны- ми системами.

Используемый метод тестирования имеет следующие особенности, которые усложняют решение задачи распараллеливания теста.

1. Тестовая последовательность строится в виде единого маршрута по графу, проходящего все его дуги.

2. Граф не задан заранее, а строится дина- мически по мере его обхода.

3. Как правило, единственным способом пе- ревода тестируемой системы в требуемое состо- яние является прохождение в графе маршрута, ведущего из текущего состояния в требуемое.

Эти особенности не позволяют распределять работу между узлами вычислительного класте- ра статически. Существуют методы (например, описанный в работе [4]), позволяющие записы- вать информацию о графе во время его обхода и использовать её в дальнейшем для статического построения кратких маршрутов, что частично решает указанную задачу. Однако для таких методов необходим первоначальный обход гра- фа, который также может занимать недопусти- мо долгое время; кроме того, статические мето- ды неустойчивы к неизбежным в процессе раз- работки изменениям как спецификации, так и реализации.

Для решения задачи распараллеливания те- стов был разработан метод параллельного об- хода графов и на его основе реализована тесто- вая система.

Требования к тестовой системе

Для возможности параллельной работы те- стовой системы необходимо, чтобы структуры МКА и порождённого им модельного графа удовлетворяли следующим ограничениям.

(A) Модельный граф конечен и сильно свя-

зан. Это требование для возможности обхода графа налагается технологией UniTESK и при последовательном режиме работы.

(B) На всех узлах вычислительного класте- ра граф одинаков.

(C) Для любой вершины графа можно вы- числить все стимулы, которыми помечены вы- ходящие из неё дуги.

(D) МКА детерминирован в том смысле,

что пресостояние и стимул любого перехода однозначно определяют его постсостояние. Это требование может быть ослаблено в зависимо- сти от используемых алгоритмов обхода [5].

Выполнение указанных выше требований необходимо для возможности применения предлагаемого метода параллельного тестиро- вания. Кроме того, для целесообразности его применения должно выполняться следующее требование.

(E) Обмены информацией о дугах с други-

ми узлами кластера должны требовать суще- ственно меньше времени, чем проходы дуг гра- фа (включающие в себя применение тестового воздействия, его обработку тестируемой систе- мой и оценку правильности полученных реак- ций) одним узлом кластера.

Архитектура тестовой системы

Тестовая система состоит из множества про- цессов, выполняющихся на узлах вычислитель- ного кластера. Процессы могут обмениваться сообщениями, но не имеют общих данных: каждый процесс имеет свою копию тестируе- мой системы и работает, вообще говоря, по сво- ему алгоритму обхода графа. Процессы связаны односторонними сетевыми соединениями типа точка-точка (двусторонние соединения рас- сматриваются как пары односторонних соеди- нений), образующими сильно связный ориенти- рованный граф; по этим соединениям они об- мениваются информацией об исследованной части модельного графа. Предполагается, что соединения сохраняют порядок сообщений, не теряют их и не производят лишних сообщений.

Каждый процесс осуществляет генерацию

своей тестовой последовательности и подачу тестовых воздействий на тестируемую систему. Процесс представляет собой систему, состоя- щую из нескольких взаимодействующих ком- понентов. Основные компоненты показаны на рис. 1.

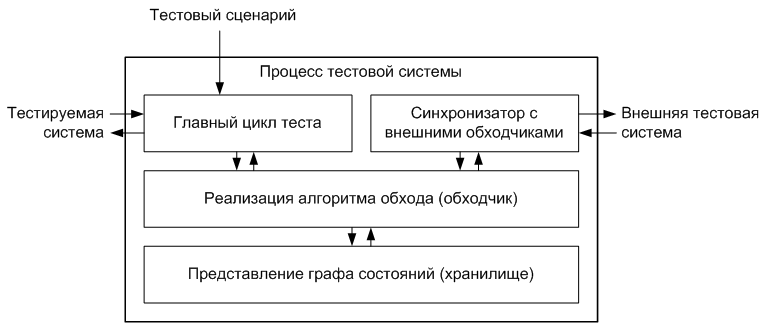


Рис. 1. Устройство процесса тестовой системы

Реализация алгоритма обхода или обходчик

– библиотечный компонент, реализующий не- который неизбыточный алгоритм обхода гра- фов [2]. Обходчик является пассивным компо- нентом, в том смысле, что он не подает воздей- ствия на тестируемую систему. Основной зада- чей обходчика является вычисление маршрута из заданной вершины графа в какую-либо вер- шину, из которой выходят еще не пройденные дуги.

Все обходчики реализуют единый интер- фейс, предоставляющий следующие функции:

• Инициализировать обходчик. Инициализи-

рует хранилище, устанавливает параметры об-

хода и передает обходчику начальную вершину графа и список допустимых в ней стимулов.

• Вычислить маршрут в графе и подавае-

мый стимул. Вычисляет маршрут из указанной

вершины графа в вершину, в которой есть еще не пройденные дуги, а также стимул одной из этих дуг. Если в графе отсутствуют непройден- ные дуги, функция сообщает о завершении те- стирования. Если обнаружено нарушение усло- вия сильной связности графа или других требо- ваний на граф, налагаемых обходчиком, функ- ция сообщает об ошибке.

• Добавить в граф пройденную дугу. Добав-

ляет дугу в хранилище (при этом указывается,

получена ли эта дуга синхронизатором от дру- гого процесса или пройдена локально). Вместе с дугой передается информация о числе стиму- лов, допустимых в конечной вершине дуги.

• Получить список дуг, пройденных локаль-

но. Возвращает список дуг, пройденных ло-

кальным обходчиком с момента последнего вы- зова этой функции (или начала работы процесса тестовой системы). Эта функция вызывается синхронизатором для получения дуг, которые нужно передать другим обходчикам.

• Завершить работу обходчика. Освобож-

дает ресурсы, занятые во время обхода.

Если обходчики, располагающие одинаковой информацией о пройденных дугах и находящи- еся в одном состоянии, выбирают очередную непройденную дугу одинаковым образом, то они будут двигаться по одному и тому же маршруту. Это особенно актуально в начале тестирования, поскольку все процессы стартуют из одной и той же начальной вершины. Для по- вышения эффективности параллельного обхода графа используется «развод» обходчиков по разным подграфам (идеальной является ситуа- ция, когда разные обходчики работают с попар- но непересекающимися подграфами). Развод может быть реализован разными способами. Один из наиболее простых подходов основан на

передаче каждому обходчику его номера – чис- ла I из множества {0, …, N–1}, где N – число процессов тестовой системы. Каждый раз, когда обходчик сталкивается с выбором стимула из списка еще не поданных стимулов L, он выби- рает тот из них, который расположен по индек-

су I mod |L|, где |⋅| – размер списка. Другим спо-

собом является случайный выбор стимула.

Представление графа состояний или храни- лище – часть обходчика или независимая подси- стема, отвечающая за хранение информации о пройденной части графа. Обновление хранили- ща осуществляется через интерфейс обходчика. Используемое представление данных может зависеть от выбранного алгоритма обхода и особенностей реализации, однако любое пред- ставление содержит следующую информацию:

• известные вершины графа;

• пройденные дуги, для которых известны

начальная вершина, стимул перехода и конеч-

ная вершина;

• стимулы, допустимые в известных вер-

шинах.

Главный цикл теста – активная часть тесто- вой системы, которая на основе тестового сце- нария (неизбыточного описания графа), исполь- зуя обходчик (обновляя модельный граф и вы- числяя пути в вершины с непройденными дуга- ми), подает воздействия на тестируемую систе- му и проверяет полученные от неё реакции.

Синхронизатор – осуществляет обмен дан- ными с аналогичными компонентами на других узлах и с обходчиком.

Протокол синхронизации

Каждый процесс регулярно выполняет про- цедуру синхронизации, которая инициируется появлением входящих сообщений или обновле- ниями в локальном хранилище, или таймером. Алгоритм синхронизации следующий:

I. Синхронизатор принимает все входящие сетевые сообщения и запрашивает у обходчика обновление – множество новых дуг, пройденных данным процессом с момента последней син- хронизации. Обозначим через R (received) мно- жество дуг, информация о которых содержится в принятых сообщениях, S (sent) множество дуг, которые данный процесс уже посылал в сооб- щениях по выходящим связям, N (new) обнов- ление.

II. В локальное хранилище (через обходчик)

добавляется множество новых переходов

R\(S∪N), помеченных как полученные от других

процессов.

III. По всем исходящим соединениям рассы- лается сообщение, содержащее множество пе-

реходов (R∪N)\S.

IV. Обновляется множество посланных пе- реходов: S := S∪R∪N.

Задав какой-либо способ упорядочения сти-

мулов, мы в силу требований к тестовой систе- ме (B) и (C) можем перенумеровать для каждой вершины Z все допустимые в ней стимулы чис- лами от 1 до deg(Z), где deg(Z) – количество та- ких стимулов, и далее рассматривать в качестве стимулов переходов только эти номера. Дуги передаются по сети в виде четвёрки (X, Y, Z, deg(Z)), где X – идентификатор начальной вер- шины дуги, Y – номер стимула, Z – идентифика- тор конечной вершины дуги. Таким образом, получателям сообщений вместе с информацией о новой вершине графа становится также изве- стен список допустимых в ней стимулов.

Несложно убедиться, что при таком прото- коле синхронизации информация о каждой ду- ге, пройденной хотя бы одним процессом, пере- даётся по каждому соединению ровно 1 раз. Оценим максимальное время T, через которое информация о новой дуге станет гарантирован- но известна всем процессам (избавляя их от необходимости проходить её самостоятельно). Пусть t – максимальное время передачи инфор- мации о дуге от одного процесса к соседнему процессу в сети, включая время обработки со- общения в процессе. Расстояние от процесса p1 до процесса p2 – минимальная длина пути из p1 в p2 по графу межпроцессных соединений. Пусть d – диаметр графа межпроцессных со- единений, то есть максимум из расстояний от процесса до процесса. Тогда T = dt.

Отметим также, что, используя этот факт, можно очищать множество S, удаляя из него дуги, которые уже были получены по каждой из входящих связей.

Координация

Процесс тестирования может управляться с использованием координатора. Координатор предоставляет пользователю возможность за- дать конфигурацию вычислительной среды: узлы, на которых запускаются процессы тесто- вой системы и связи между ними. В начале ра- боты тестовой системы координатор запускает все процессы и передает им информацию об их входящих и выходящих соединениях. Кроме того, координатор позволяет управлять процес- сом тестирования и централизованно собирать информацию о состоянии процессов, найден- ных ими ошибках, достигнутом тестовом по-

крытии, количестве пройденных дуг графа и т.д.

При обнаружении ошибки дальнейшее пове- дение тестовой системы определяется уровнем критичности этой ошибки. Выделяются следу- ющие уровни:

1) Локально устранимая ошибка – ошибка, которая не влияет на дальнейший обход графа. При обнаружении такой ошибки процесс опо- вещает координатора и продолжает тестирова- ние и обмен данными с другими процессами.

2) Локально неустранимая ошибка – ошиб-

ка, при которой дальнейший обход графа этим процессом невозможен. При обнаружении та- кой ошибки процесс передает информацию об этом координатору, прекращает тестирование, однако продолжает обмен данными с другими процессами, сохраняя таким образом сильную связность графа межпроцессных соединений.

3) Глобально неустранимая ошибка – ошиб- ка, при которой невозможен дальнейший обход графа всеми процессами. Процесс оповещает координатора, который передает всем осталь- ным процессам указание о прекращении тести- рования.

Завершение работы тестовой системы также

возможно при получении соответствующей ко- манды от пользователя. В этом случае коорди- натор передает всем узлам указание завершить работу. Процессы могут сохранить свое состоя- ние, чтобы была возможность в дальнейшем возобновить тестирование.

Сохранение и восстановление состояния

Несмотря на ускорение тестирования за счёт распараллеливания, полный обход модельного графа всё равно может занимать весьма про- должительное время. Сохранение результатов проделанной работы для последующего восста- новления и продолжения обхода графа позволя- ет прерывать тестирование, например, для про- ведения регламентных работ, а также продол- жать тестирование после сбоев, не теряя полу- ченных результатов.

Рассмотрим два протокола такого сохране-

ния и восстановления.

Протокол контрольных точек

Построение контрольной точки. Время от времени (по команде координатора или по не- кому общему соглашению) все процессы при- останавливают обход графа, продолжая при этом обмены информацией, и ждут, когда каж- дый процесс завершит прохождение очередной

дуги, а в сети перестанут циркулировать сооб- щения о пройденных дугах. Как показано выше, это произойдёт, как только информация о каж- дой пройденной дуге достигнет каждого про- цесса, после чего в хранилище каждого процес- са будет содержаться информация об одном и том же наборе вершин, пройденных и непрой- денных дуг. После этого каждый процесс со- храняет информацию из своего хранилища на локальный диск, а когда сохранение успешно завершено всеми процессами, продолжает те- стирование из той вершины модельного графа, в которой он находится в данный момент (или завершает работу, если поступила соответству- ющая команда от координатора).

Восстановление происходит следующим об-

разом: после запуска всех процессов и установ- ления между ними соединений каждый процесс загружает из локального файла состояние хра- нилища, сохранённое при последнем успешном построении контрольной точки, после чего начинает обход графа с его начальной вершины.

Протокол журнализации

Несколько процессов при старте назначают-

добавляются в локальное хранилище вместе с полученными от других процессов. Отметим, что выполнение требования (E) гарантирует, что загрузка и рассылка всем процессам сохра- нённых данных об известной части графа могут быть выполнены существенно быстрее, чем по- вторный её обход. После завершения загрузки из журнала этот процесс переходит к тестиро- ванию в обычном режиме, при этом он может дописать в конец своего журнала дуги, которые в нём отсутствовали, но были получены во вре- мя загрузки от других процессов, и в дальней- шем продолжать дописывать в конец журнала все новые дуги.

Опыт применения подхода

Описанный в статье подход был применен для распараллеливания функционального те- стирования моделей цифровой аппаратуры [5]. Получаемые в процессе графы состояний в за- висимости от сложности моделей и целей те- стирования содержали от тысячи до миллиона вершин и, соответственно, от нескольких тысяч до нескольких миллионов дуг. Для прогона те-

стов использовалось от 1 до 150 компьютеров

®

ся ведущими журнал (если не нужна защита от

(процессор: Intel

Core™2 Quad Q9400,

таких сбоев узла, при которых теряется содер- жимое его диска, то достаточно одного процес- са). Синхронизаторы этих процессов каждый раз при рассылке по сети сообщений с множе- ством дуг одновременно записывают эти дуги в том же порядке в файл на локальном диске (журнал).

Восстановление происходит следующим об-

разом: после запуска всех процессов и установ- ления соединений один из процессов начинает последовательную загрузку дуг из журнала, в то время как все остальные начинают работать обычным образом. Во время загрузки этот про- цесс не осуществляет обход графа, но принима- ет и ретранслирует сообщения о новых дугах, полученные от других процессов. Загружаемые из журнала дуги также рассылаются по сети и

2.66 GHz; оперативная память: 4 Gb), работаю- щих под управлением ОС Linux и объединен- ных в сеть Ethernet. В табл. 1 приводится услов- ная классификация сложности тестов в зависи- мости от числа дуг в графе состояний и указы- вается количество ресурсов, затрачиваемое на прогон тестов (число компьютеров и время).

Аналитическая оценка эффективности рас-

параллеливания представляется затруднитель- ной, поскольку в ней нужно учесть множество факторов, включая время прохода по дуге гра- фа, время, затрачиваемое на передачу данных между процессами, топологию межпроцессных связей и многие другие. При проведении экспе- риментов измерялся коэффициент эффективно- сти распараллеливания K(n) = T(1)/(nT(n)), где T(n) – время прогона теста на n процессах.

Классификация тестов по уровню сложности

Таблица 1

|  |  |  |  |
| --- | --- | --- | --- |
| Категория сложности тестов | Число дуг в графе  состояний | Используемое число  компьютеров | Время, затрачиваемое  на прогон тестов, мин |
| Простые тесты | < 10000 | 1 | < 30 |
| Тесты средней сложности | 10000 – 100000 | 1–10 | < 30 |
| Сложные тесты | 100000 – 1000000 | 10–100 | < 30 |
| Очень сложные тесты | > 1000000 | > 50 | > 60 |

Проведенные замеры показывают, что на за- дачах функционального тестирования моделей аппаратуры при правильно подобранной топо- логии значение коэффициента эффективности распараллеливания превосходит 0.8, однако здесь следует сделать несколько замечаний. Во-первых, эксперименты проводились на ком- пьютерах с двумя и более ядрами, что позволяет синхронизатору не отнимать вычислительные ресурсы у обходчика графов. Во-вторых, для

разного числа компьютеров использовались разные топологии межпроцессных связей: для 8 и менее компьютеров, как правило, использова- лась топология «кольцо», для 9 и более – «дву- мерный тор».

Для 100–150 компьютеров этих двух вариан- тов вполне достаточно, однако для эффективно- го распараллеливания на большем числе ком- пьютеров могут потребоваться другие тополо- гии («трехмерный тор», «гиперкуб»).

Результаты прогона теста средней сложности

Таблица 2

|  |  |  |  |
| --- | --- | --- | --- |
| Используемое число  компьютеров | Топология  межпроцессных связей | Время, затрачиваемое  на прогон теста, мин | Коэффициент эффективности  распараллеливания |
| 1 | — | 95.2 | 1 |
| 9 | Кольцо | 11.8 | 0.9 |
| 9 | Тор 3×3 | 10.9 | 0.97 |
| 16 | Кольцо | 6.7 | 0.89 |
| 16 | Тор 4×4 | 6.2 | 0.96 |
| 25 | Кольцо | 4.4 | 0.87 |
| 25 | Тор 5×5 | 4.0 | 0.95 |

Результаты прогона сложного теста

Таблица 3

|  |  |  |  |
| --- | --- | --- | --- |
| Используемое число  компьютеров | Топология  межпроцессных связей | Время, затрачиваемое  на прогон теста, мин | Коэффициент эффективности  распараллеливания |
| 1 | — | 803.3 | 1 |
| 81 | Кольцо | 12.2 | 0.81 |
| 81 | Тор 9×9 | 11.4 | 0.87 |
| 100 | Кольцо | 10.2 | 0.79 |
| 100 | Тор 10×10 | 9.5 | 0.85 |

В табл. 2 и 3 представлены типичные ре- зультаты прогона тестов. В них показано время, затрачиваемое на прогон теста средней сложно- сти (18227 состояний, 109362 дуги) и сложного теста (84561 состояние, 338244 дуги) на разном числе компьютеров для топологий «кольцо» и

«двумерный тор».

Перспективы

Следующие вопросы требуют дополнитель- ного изучения:

• Выбор оптимального графа обмена дан-

ными между узлами вычислительной систе-

мы. Узлы вычислительной системы должны быть соединены между собой таким образом, чтобы информация между ними распространя- лась как можно быстрее, но без чрезмерной

нагрузки на каналы связи. Полярными вариан- тами соединений являются схемы «каждый-с- каждым» и «кольцо». При выборе схемы связей следует учитывать возможности конкретной системы.

• Специализация вычислительных узлов.

Возможно, в некоторых ситуациях окажется

полезным разделение функциональности вы- числительных узлов. Решением ресурсоемких задач могут заниматься специально выделенные узлы, обслуживая запросы всех остальных. Например, обходчики могут самостоятельно принимать решение, если в текущей вершине модельного графа или в соседних с ней есть непройденные дуги. А в сложных случаях, ко- гда расстояние до непройденных дуг велико, запрашивать выбор пути у специально выде- ленного узла.

• Централизованное управление обходом

графа. Выделение узлов, которые управляют

поведением множества обходчиков, дает воз- можность координировать их работу. Таким образом можно повысить эффективность распа- раллеливания особенно для систем, граф состо- яний которых имеет определенный вид.

В качестве примера рассмотрим ситуацию,

когда из начальной вершины выходит две дуги, но одна из них ведет вершину с большим коли- чеством выходящих дуг, а другая – в длинное кольцо, из каждой вершины которого ведет только одна дуга.

Если каждый обходчик принимает решение самостоятельно, то примерно половина пойдет по кольцу и будет дублировать действия друг друга. Централизованное управление позволяет предварительно исследовать возможные пути на определенную глубину, используя неболь- шое количество обходчиков, а затем распреде- лить основную массу обходчиков по путям пропорционально количеству найденных непройденных дуг.

Заключение

В статье описано расширение технологии UniTESK средствами распараллеливания рабо- ты тестовых систем на распределенных вычис- лительных кластерах. Важной особенностью является то, что распараллеливание осуществ- ляется динамически, без использования инфор- мации о структуре модельного графа. С точки зрения инженеров, работать с распределенной тестовой системой не сложнее, чем с обычной тестовой системой, выполняемой на одном компьютере (дополнительные входные данные связаны со структурой межпроцессных соеди- нений, но они задаются один раз, при конфигу- рации). Предлагаемый подход обеспечивает существенное сокращение времени выполнения

тестов, и, как следствие, уменьшение времени, затрачиваемого на обнаружение ошибок, и ускорение процесса проектирования в целом.

Дальнейшее направление работы мы видим в реализации механизмов, нацеленных на повы- шение гибкости и эффективности распаралле- ливания тестов. К ним относятся динамическая реконфигурация топологии сети (для сложных тестов может потребоваться ускорение за счет добавления новых вычислительных узлов в граф межпроцессных соединений) и поддержка систем с общей памятью, в частности, много- ядерных процессоров (в этом случае возможна более эффективная реализация синхронизато- ров, а также совместное использование одного хранилища несколькими процессами).

Список литературы

1. Кулямин В.В., Петренко А.К., Косачев А.С., Бурдонов И.Б. Подход UniTesK к разработке тестов

// Программирование. 2003. № 29(6). С. 25–43.

2. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов: детерминированный случай // Программиро- вание. 2003. № 29(5). С. 59–69.

3. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Использование конечных автоматов для тестирова- ния программ // Программирование. 2000. № 26(2). С. 61–73.

4. Грошев С.Г. Локализация ошибок методом построения сокращенных трасс // Программирова- ние. 2009. № 35(3). С. 35–50.

5. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов: недетерминированный случай. // Програм- мирование. 2004. № 1(30). С. 2–17.

6. Иванников В.П., Камкин А.С., Косачев А.С., Кулямин В.В., Петренко А.К. Использование кон- трактных спецификаций для представления требо- ваний и функционального тестирования моделей аппаратуры // Программирование. 2007. № 33(5). C. 47–61.

PARALLEL TESTING OF LARGE AUTOMATA MODELS

I.B. Burdonov, S.G. Groshev, A.V. Demakov, A.S. Kamkin, A.S. Kosachev, A.A. Sortov

The UniTESK technology allows the development of high-quality tests to check complex systems for compli- ance with formal models of requirements. However, in most cases such tests take a significant amount of time. This article presents an extension of the UniTESK technology which makes it possible to speed up the testing by its paral- lel execution on a computer cluster.

Keywords: functional testing, formal specifications, hardware models, distributed systems, distributed testing, finite-state automata.