МЕТОДЫ И СРЕДСТВА АНАЛИЗА ФУНКЦИОНИРОВАНИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Балашов В.В., Бахмуров А.Г., Волканов Д.Ю.,
Смелянский Р.Л., Чистолинов М.В., Ющенко Н.В.

СТЕНД ПОЛУНАТУРНОГО МОДЕЛИРОВАНИЯ ДЛЯ РАЗРАБОТКИ ВСТРОЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Фак-т. ВМК МГУ имени М.В.Ломоносова, Москва,
e-mail: {hbd,bahmurov,dimawolf,smel,mike,youwhile}@cs.msu.su

1. Введение

Современные бортовые встроенные системы реального времени (ВС РВ) представляют собой многомашинные вычислительные комплексы. Количество каналов связи между приборами в составе ВС РВ достигает нескольких десятков. На каждой из ВС РВ работает специфичный набор приложений и системных задач. На ранних этапах разработки аппаратные и программные компоненты ВС РВ разрабатываются параллельно. На этих этапах для разработчиков программного обеспечения доступна лишь часть прототипов аппаратных устройств. Это создаёт множество проблем разработчикам ВС РВ.

Во-первых, недоступность аппаратных устройств или их прототипов ограничивает возможности оценки временных характеристик операций программного обеспечения над аппаратными устройствами, поэтому возникает потребность для оценки этих характеристик без доступа к прототипам аппаратных устройств.

Во-вторых, стандартная ОС РВ (операционная система реального времени) включает некоторые задачи, которые реализуют логику обменов по аппаратным бортовым каналам связи, включая обработку ошибок обмена. Для активизации и тестирования этих задач без привлечения прототипов других взаимодействующих устройств необходимо моделировать работу этих задач и моделировать обмен по бортовым каналам.

Также необходимо отметить, что разработка приборов, входящих в состав ВС РВ, на практике происходит распределённо и выполняется различными организациями. Готовность различных приборов к комплексированию наступает в разные моменты времени. Для соблюдения сроков комплексирования ВС РВ необходимо начинать работы по комплексирования с неполным комплектом доступных приборов.

На этапе комплексирования ВС РВ возникает ряд задач, требующих инструментальной поддержки, в том числе:

• проверка соответствия приборов ВС РВ требованиям технического задания, в том числе в части приёма и передачи данных по внешним интерфейсам;

• отработка взаимодействия между приборами ВС РВ по бортовым каналам передачи данных;

• комплексное тестирование и отладка ПО ВС РВ, в том числе ПО, выполняемого распределённо на различных приборах;

• оценка надёжности архитектуры ВС РВ, в том числе наличия резерва пропускной способности каналов передачи данных и устойчивость аппаратно-программных средств ВС РВ к сбоям при передаче данных;

• построение расписаний обмена данными по бортовым каналам, а также проверка правильности отработки этого расписания приборами в составе ВС РВ.

В данной работе описаны программно-аппаратные средства (стенд) полунатурного моделирования (ПНМ) бортовых ВС РВ, позволяющие осуществлять комплексирование ВС РВ и решать перечисленные выше задачи поэтапно, расширяя состав стыкуемых приборов по мере их готовности в виде натурных образцов.

Основными методами исследования ВС РВ в стенде ПНМ являются имитационное и полунатурное моделирование. В зависимости от степени готовности приборов ВС РВ, в составе стенда могут использоваться полностью программные имитационные модели всех приборов, либо комплекс из натурных образцов приборов и программных моделей приборов, собранных в единый стенд и сопряжённых через аппаратные каналы бортовых интерфейсов.

Также может варьироваться уровень детальности моделирования - от так называемых "интервальных" моделей, которые отрабатывают заданные циклограммы обменов, не выполняя вычисление передаваемых прибором данных, до полных функциональных моделей, эквивалентных реальным приборам по составу и значениям выдаваемых в бортовые каналы данных и включающих в свой состав реальное ПО приборов.

Стенд разработан в лаборатории вычислительных комплексов (ЛВК) факультета ВМК МГУ[1]. Особенностью стенда является возможность сравнительной оценки архитектур ВС РВ при помощи моделирования на ранних этапах разработки ВС РВ, что отличает стенд от существующих решений, обзор которых приведён в следующем разделе.

2. Задачи стенда полунатурного моделирования

Основными задачами стенда ПНМ являются:

- проведение исследований и оценка технических решений в области структуры ВС РВ, характеристик приборов и их программного обеспечения;
- комплексная отработка оборудования при решении задач информационного взаимодействия и прикладных задач ВС РВ;
- проверка работоспособности аппаратуры, в том числе проверка соответствия функционирования приборов и каналов передачи данных требованиям технического задания.

Основными методами исследования ВС РВ в стенде ПНМ являются имитационное и полунатурное моделирование. В зависимости от степени готовности приборов ВС РВ, в составе стенда могут использоваться полностью программные модели всех приборов, либо комплекс из натурных образцов приборов и программных моделей приборов, собранных в единый стенд и сопряжённых через аппаратные каналы бортовых интерфейсов.

Также может варьироваться уровень детальности моделирования - от так называемых "интервальных" моделей, которые отрабатывают заданные циклограммы обменов, не выполняя вычисление передаваемых прибором данных, до полных функциональных моделей, эквивалентных реальным приборам по составу и значениям выдаваемых в бортовые каналы данных и включающих в свой состав реальное ПО приборов.

Средства разработки моделей, входящие в стенд ПНМ, позволяют быстро создавать модели, имитирующие "окружение" отлаживаемого прибора и взаимодействующие с ним через аппаратные каналы передачи данных.

4. Структура программных и аппаратных средств стенда

ПО стенда ПНМ включает в себя набор программных инструментов, обеспечивающих решение следующих задач, связанных с моделированием:

- создание имитационных моделей приборов ВС РВ, а также вспомогательных моделей (например, модели внешней среды);
- организация взаимодействия моделей, выполнения набора моделей, их взаимодействия с аппаратурой в модельном и в реальном времени по бортовым каналам с возможностью внесения отказов в каналы;
- управление процессом моделирования в диалоговом режиме, либо выполнение автономного эксперимента без участия оператора;
- оперативное отображение результатов моделирования в графическом и табличном виде;

• регистрация и обработка результатов моделирования, в том числе взаимодействие с аппаратными мониторами каналов передачи данных.

ПО стенда функционирует под управлением ОС Debian GNU/Linux. Оборудование стенда ПНМ включает в себя следующие компоненты (Рис. 1):

• инструментальные машины частных моделей (ИМ ЧМ);

• инструментальные машины-регистраторы обмена по бортовым каналам передачи данных (ИМ-регистраторы);

• машины АРМ инженера-экспериментатора;

• натурные приборы из состава ВС РВ;

• натурные каналы информационного обмена, к которым подключены приборы, ИМ ЧМ и ИМ-регистраторы.

Инструментальные машины и АРМ представляют собой персональные компьютеры, объединённые сетью Ethernet. На ИМ ЧМ осуществляется имитационное моделирование приборов с помощью разработанных исследователем моделей. При моделировании соблюдается соответствие модельного и реального (астрономического) времени с погрешностью порядка 0,0001 секунды. Указанная точность обеспечивается применением на ИМ ЧМ специализированных расширений реального времени для ядра ОС Linux, входящих в среду выполнения моделей [1]. В состав ИМ ЧМ входят адаптеры бортовых каналов передачи данных (поддерживаются стандарты МКИО (ГОСТ Р 52070-2003) [3], ARINC-429 (ГОСТ 18977-79, РТМ 1495-75), частично поддерживается стандарт Fibre Channel. При помощи этих адаптеров модели устройств ВС РВ осуществляют обмен по каналам в реальном времени, полностью или в заданной части имитируя работу реальных устройств. Необходимо отметить, что на канале МКИО модель устройства может выполнять роль контроллера канала, что обеспечивает возможность комплексирования подмножества устройств ВС РВ, не включающего устройство-контроллер канала.

Реализованная в стенде функциональность взаимодействия моделей устройств ВС РВ и натурных устройств в реальном времени позволяет обеспечить пошаговое комплексирование ВС РВ с поэтапной заменой программных моделей устройств на натурные устройства. Таким образом. комплексное тестирование и отработка взаимодействия могут выполняться на произвольном сочетании натурных и моделируемых устройств.

ИМ-регистраторы осуществляют мониторинг информационного обмена по бортовым каналам и запись протоколов регистрации на жёсткий диск для последующего анализа. При выполнении эксперимента, на всех инструментальных машинах поддерживается

единый отсчёт времени, расхождение счётчиков времени на разных машинах не превышает 0,00001 секунды. Для генерации строго одновременных событий в целях синхронизации времени применяется специализированный протокол, использующий соединение LPT-портов ИМ ЧМ и ИМ-регистраторов. Периодически одна из ИМ ЧМ (мастер синхронизации времени) посылает сигнал через

свой порт LPT. Остальные машины принимают сигнал и фиксируют одновременное событие. Регистрация событий в моделях (на ИМ ЧМ) и регистрация обмена по каналам (на ИМ-регистраторах) выполняется в едином времени, что позволяет с высокой точностью отображать на единой оси времени события на разных каналах и в разных моделях.

Средства АРМ инженера-экспериментатора обеспечивают решение следующих задач:

- разработка моделей с поддержкой управления версиями моделей;
- настройка эксперимента, в т.ч. распределение моделей по инструментальным машинам и настройка регистрации по каналам;
- управление экспериментом с поддержкой оперативного изменения значений параметров моделей с консоли оператора;
- оперативное наблюдение за ходом эксперимента, включая отображение значений параметров моделей и отображение результатов регистрации;
- анализ результатов эксперимента, в том числе трасс значений параметров моделей, трасс событий в моделях и каналах.

Для разработки моделей используется специализированный язык описания моделей, представляющий собой подмножество языка Си, расширенное средствами привязки работы модели к реальному времени и средствами организации межмодельного взаимодействия.

6. Технология комплексирования ВС РВ с применением стенда

При комплексировании ВС РВ с применением стенда ПНМ, средства стенда применяются совместно со средствами инструментальной системы планирования обменов по каналу МКИО (САПР циклограмм) [4, 5], также разработанное в ЛВК. В составе САПР циклограмм следует выделить следующие компоненты, существенные для совместного применения САПР и стенда:

- база данных (БД) проекта ВС РВ, содержащая информацию о структуре ВС РВ (состав приборов и каналов передачи данных, структура бортовой сети) и о порядке обмена между приборами (входные и выходные данные устройств, структура передаваемых сообщений, расписание информационного обмена);

• средства автоматического построения расписания информационного обмена для выполнения на устройстве-контроллере канала MIL STD-1553B.

В рамках разработанной технологии, процедура комплексирования ВС РВ в условиях доступности заданного (как правило, неполного) набора аппаратных устройств, включает следующие шаги:

1. Подготовка аппаратного обеспечения стенда, в том числе подключение ИМ ЧМ, ИМ-регистраторов и натурных приборов к каналам передачи данных.

2. Ввод в БД проекта информации о структуре ВС РВ и порядке информационного обмена между приборами.

3. Автоматическое формирование интерфейсной части моделей приборов ВС РВ (входные и выходные параметры моделей, описание интерфейсов моделей с бортовыми каналами, описание связей между моделями).

4. Реализация логики функционирования моделей тех устройств, которые отсутствуют в аппаратном исполнении.

5. Автоматическое формирование (средствами САПР циклограмм) расписаний обмена по каналам MIL STD-1553B.

6. Автоматическое формирование кода задания расписания: для аппаратных устройств-контроллеров канала в составе ВС РВ и для моделей устройств-контроллеров канала.

7. Автоматическое формирование кода упаковки и распаковки передаваемых по каналам сообщений для аппаратных устройств в составе ВС РВ и для моделей устройств.

8. Настройка эксперимента, включающее распределение моделей по ИМ ЧМ и задание настроек регистрации обмена по бортовым каналам и событий в моделях.

9. Выполнение эксперимента, при котором в составе стенда одновременно работают натурные приборы ВС РВ и модели отсутствующих приборов.

10.Анализ результатов эксперимента, в том числе автоматизированная проверка зарегистрированных протоколов обмена по бортовым каналам на соответствие эталонным расписаниям обменов, заложенным в БД проекта.

При пополнении состава доступных натурных устройств ВС РВ, комплексирование ВС РВ в расширенном составе требует выполнения только тех шагов, которые связаны с подключением новых устройств, проведением экспериментов с их участием и анализа результатов этих экспериментов. При комплексировании последующих версий ВС РВ (например, в рамках создания линейки ВС РВ), разработанные модели,

наполнение БД и настройки экспериментов могут быть повторно использованы.

Стенд ПНМ может быть также применятся при отработке функционального ПО (ФПО) и системного ПО (СПО) приборов ВС РВ с использованием моделей этих приборов. Для этого в составе программных средств стенда должны быть реализованы модели низкоуровневых сервисов ОС прибора, в частности - модели сервисов межпроцессного взаимодействия и модели драйверов, работающих с адаптерами бортовых интерфейсов. После реализации моделей указанных сервисов, в состав моделей устройств ВС РВ могут быть включены исходные тексты разрабатываемого ПО. Выполнение моделирования с задействованием исходных текстов ПО позволяет осуществлять тестирование и отработку ПО без задействования натурного прибора, который может быть недоступен в аппаратном исполнении и выполнять детальное наблюдение за изменением выбранных переменных ПО с задействованием механизма оперативного отображения параметров моделей.

По завершении отработки ПО на модели, выполняется включение этого ПО в состав реального прибора (по факту готовности прибора) и продолжается отработка ПО уже в составе прибора.

В случае если прибор доступен в аппаратном исполнении с самого начала процесса комплексирования ВС РВ, использование стенда позволяет расширить состав ПО, задействуемого при отработке прибора, за счёт подачи на интерфейсы прибора данных по каналам бортовых интерфейсов и активизации ПО, отвечающего за приём и передачу данных по этим каналам. Частным случаем такого подхода к отработке ПО на натурном приборе является моделирование всего окружения прибора и анализ ответов прибора на тестовые наборы данных, формируемые моделями и подаваемые на вход прибора.

Стенд ПНМ имеет открытую архитектуру, позволяющую выполнять интеграцию со стендами других разработчиков. Подобная интеграция целесообразна в случаях, когда для выполнения некоторых задач отработки ВС РВ необходимо использование спецвычислителей в составе стенда. Примерами таких задач могут служить динамическое формирование и регистрация видеоданных в реальном времени, а также получение управляющих воздействий от пилота через органы управления в стенде-имитаторе кабины пилота. Существующая реализация стенда ПНМ задействует для интеграции с внешними стендами специализированный протокол обмена, основанный на технологии Ethernet. В перспективе рассматривается возможность применения технологии HLA[6].

7. Заключение

В данной работе представлен стенд полунатурного моделирования, разработанный в Лаборатории вычислительных комплексов факультета ВМК МГУ. Средствами стенда решаются задачи пошагового комплексирования бортовых ВС РВ с применением технологии полунатурного моделирования, а также задачи сравнительной оценки архитектур ВС РВ на ранних этапах разработки.

Стенд применяется в промышленности при разработке бортовых ВС РВ. В данной работе подробно рассматриваются изменения, внесенные в стенд для его успешного применения при моделировании ВС РВ морских НК.

К перспективным задачам по развитию стенда следует отнести:

- полную реализацию поддержки высокоскоростных каналов Fibre Channel, используемых в новейших ВС РВ;
- применение технологии HLA для реализации взаимодействия со сторонними стендами;
- интеграцию средств оперативного отображения результатов эксперимента с инструментальными средствами отладки в составе ПО приборов ВС РВ, что позволит выполнять оперативное наблюдение за изменением значений переменных в составе ПО приборов ВС РВ.
- реализацию автоматического сравнения результатов регистрации обмена по МКИО с эталонной циклограммой из БД циклограмм;
- создание автономного мобильного варианта стенда для мониторинга и анализа функционирования комплекса на объекте ссылка[2]

1. Грибов Д.И., Смелянский Р.Л. Комплексное моделирование бортового оборудования летательного аппарата // Методы и средства обработки информации. Труды второй Всероссийской научной конференции. - М.: Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 2005. - С.59-74
2. M.P.A.M. Brouwer, et al., Developments in Test and Verification Equipment for Spacecraft. Technical report NLR-TP-2000-658,National Aerospace Laboratory, Netherlands, 2000.
3. ГОСТ Р 52070-2003. Интерфейс магистральный последовательный системы электронных модулей.
4. V.V. Balashov, V.A. Kostenko, R.L. Smeliansky, S.V. Vavinov. A Tool System for Automatic Scheduling of Data Exchange in Real-Time Distributed Embedded Systems. Proc. of the 7th IEEE International Symposium on Computer Networks (ISCN'06), Istanbul, Turkey, 2006.
5. V.V. Balashov, V.A. Kostenko, R.L. Smeliansky, A Tool System for Automatic Scheduling of Data Exchange in Real-Time Distributed Avionics

Systems. Proc. of the 2nd EUCASS European Conference for Aerospace Sciences, Brussels, Belgium, 2007.

6. IEEE 1516 Standard Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules, IEEE, 2000.
7. Волканов Д.Ю. Использование имитационного моделирования для оценки надежности распределенных вычислительных систем // Труды Всероссийской научной конференции "Методы и средства обработки информации" (1 октября - 3 октября 2003 г., г. Москва) -М.: Издательский отдел факультета ВМиК МГУ, 2003. - С. 343-348.

Бахмуров А.Г., Волканов Д.Ю., Смелянский Р.Л.

МЕТОДИКА ПОИСКАОПТИМАЛЬНОГО НАБОРА МЕХАНИЗМОВ ОТКАЗОУСТОЙЧИВОСТИ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Фак-т. ВМК МГУ имени М.В.Ломоносова, Москва,
e-mail: {bahmurov,dimawolf,smel}@cs.msu.su

1. Введение

Под вычислительной системой реального времени обычно понимают такую вычислительную систему, работоспособность которой зависит не только от логических результатов вычислений, но и от промежутка времени, за который эти результаты были получены[1]. Бортовая вычислительная система (БВС) - это частный случай системы реального времени, на которую, помимо ограничений на время выполнения вычислений, могут накладываться ограничения на габариты, массу и потребляемую мощность.

При разработке БВС необходимо уделять должное внимание средствам отказоустойчивости и повышения надёжности. Надёжность - это способность системы безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с вероятностью не ниже заданной. Важными понятиями при исследовании надёжности вычислительной системы являются понятия неисправности, ошибки и отказа[2].

Под неисправностью в системе будем понимать некий дефект в программе или аппаратуре, который в определенных случаях может помешать эксплуатировать систему согласно ее спецификации. Говорят, что в какой-то момент времени в системе произошла ошибка, если в этот момент времени состояние системы отличается от ожидаемого. Говорят, что в системе произошёл отказ, если она не обеспечивает того поведения и функциональности, которого можно было ожидать от неё заранее, то есть, результат работы системы на каких-то входных данных отличается от ожидаемого по спецификации. При наличии неисправности в системе, система может перейти в ошибочное состояние, а ошибочное состояние привести к отказу системы. Одним из основных свойств системы, составляющих надёжность, является отказоустойчивость. Отказоустойчивость – это свойство системы или компонента системы, заключающееся в способности обнаружить и ликвидировать ошибочные состояния системы без влияния на её работоспособность.

В статье рассматривается задача выбора оптимального набора механизмов отказоустойчивости для БВС, а в качестве решения этой задачи была предложена методика, работающая с использованием генетических алгоритмов[3]. Первые работы, посвященные применению генетических алгоритмов для решения различных задач обеспечения надёжности вычислительных систем, принадлежат Томпсону[4]. Его идеи использовал и развил в своих работах Хартманн[5]. В этих работах решаются задачи проектирования отказоустойчивых интегральных схем, состоящих из различных логических элементов, вентилей и прочих низкоуровневых компонентов вычислительных систем. В данной статье рассматривается более высокий уровень представления вычислительной системы, а именно системный.

2. Задача выбора оптимального набора механизмов отказоустойчивости

Рассматривается БВС, описанная при помощи модели на основе инварианта программы[6]. В описании БВС выделяются аппаратные и программные компоненты. Для данной БВС известен набор неисправностей, которые могут возникать в компонентах. Также известен набор доступных механизмов отказоустойчивости[7]: резервирование компонентов, переконфигурирование системы, метод кодирования, метод с использованием контрольных точек, многоверсионное программирование, которые могут применяться к каждому из компонентов.

Необходимо выбрать такой набор механизмов отказоустойчивости, который бы предотвращал возможность возникновения отказа системы для максимального количества неисправностей при заданных ограничениях на систему.

В качестве ограничений на систему выступают: ограничение на применение данного механизма отказоустойчивости для конкретного компонента и сохранение директивных сроков выполняемых в системе задач.

3. Описание методики

Методика включает в себя следующие шаги:

1. Создание имитационной модели с нужным уровнем детальности.
2. Внесение неисправностей различных типов в модель.
3. Поиск при помощи генетического алгоритма набора механизмов отказоустойчивости.
4. Построение новой модели с полученным на шаге 3 набором механизмов отказоустойчивости и проверка соблюдения директивных сроков задач.

5. Если не выполнен критерий останова на шаге 3 или не выполнены ограничения, проверенные на шаге 4, то переход к шагу 3, иначе текущее решение принимается в качестве решения задачи.

Ниже приводится более подробное описание каждого из шагов методики.

Ввиду невозможности использования натурных компонентов, вычислительная система (её программные и аппаратные компоненты) на первом шаге применения методики представляется в виде имитационной модели.

На втором шаге с помощью средства внесения неисправностей в эту модель системы вносятся неисправности различных типов и распределяются между компонентами системы. Для моделирования состояния системы, в котором возникает отказ, надо каким-либо образом учитывать неисправности, возникающие в натурной системе. Существуют алгоритмы [8], позволяющие имитировать появление отказов в системе путём внесения неисправностей в модель системы.

На третьем шаге при помощи генетического алгоритма ищется набор механизмов отказоустойчивости. Для механизмов отказоустойчивости, которые мы хотим применять, известно какие неисправности они могут устранять, и известны наборы параметров этих средств. Разумно предположить, что если средство, позволяющее устранить неисправность, применяется к неисправному компоненту, то неисправность не приводит к отказу системы и таким образом ликвидируется. Из-за применения средств отказоустойчивости возрастает время выполнения задач в системе и они могут не укладываться в директивный срок.

Решение задачи выбора механизмов отказоустойчивости представляет собою закодированную битовую строку. Битовая строка состоит из блоков, которые соответствуют компонентам вычислительной системы. Число битов в блоке соответствует числу заданных средств отказоустойчивости, и каждый бит определяет, применяется ли то или иное средство к текущему компоненту (1 – применяется, 0 – нет).

В качестве целевой функции выступает количество неустранённых неисправностей в системе после применения средств отказоустойчивости, указанных в решении. Эта функция минимизируется.

Этот шаг рассматриваемой методики можно разделить на следующие этапы:

I. Генерация начальной популяции решений, на основе информации о моделируемой системе и внесённых в неё неисправностях.

II. Вычисление значений целевой функции для членов популяции и фиксирование наилучшего на текущий момент решения. Для каждого заданного в популяции решения целевая функция определяет, качество данного решения.

III.Проверяется критерий останова, если он достигнут, то вычисления прекращаются. В качестве критерия останова используется условие выполнения алгоритмом заданного количества итераций без улучшения целевой функции.

IV. Выбор решения для следующей популяции. На данный момент алгоритм использует схему пропорциональной селекции для выбора элементов популяции.

V. Над элементами новой популяции производится операция скрещивания. Элементы для скрещивания выбираются произвольным образом. Используется схема универсального (вероятностного) скрещивания двух решений, в результате чего появляются два новых решения. При скрещивании отбрасываются решения, которые не удовлетворяют ограничениям на использование механизмов отказоустойчивости. Например, механизм контрольных точек не применяется для аппаратных компонентов, поэтому в битовом представлении решения в соответствующих позициях для аппаратных будет стоять ‘0’ и операции скрещивания выбраны так, что не будут его модифицировать.

VI. Над элементами новой популяции производится операция мутации. Уровень мутации элемента популяции зависит от параметра, задаваемого пользователем. При мутации также как и при скрещивании отбрасываются решения, которые не удовлетворяют ограничениям на использование механизмов отказоустойчивости.

VII. Переход к этапу II.

На четвёртом шаге методики по полученному на предыдущем шаге решению строится имитационная модель, в которой, если для конкретного компонента в полученном решении стоит '1' в качестве признака использования данного механизма отказоустойчивости, то данный механизм применяется к компоненту в имитационной модели. После запуска этой модели и проверки директивных сроков выполнения задач данное решение принимается как текущее, либо не принимается.

Затем возвращаемся к предыдущему шагу и повторяем процесс получения нового решения и его проверки, пока решение, удовлетворяющее ограничениям, не будет улучшаться при заданном числе итераций генетического алгоритма.

Данная методика без автоматической генерации модели с механизмами отказоустойчивости была реализована в виде программного средства.

4. Апробация методики

Проверка корректности методики проводилась путём сравнения результатов, полученных с использованием генетического алгоритма с результатами, полученными с помощью алгоритма полного перебора.

В качестве среды моделирования была выбрана система ДИАНА [9]. Для внесения неисправностей в модель вычислительной системы используется Средство автоматического внесения неисправностей [8]. Реализация генетического алгоритма, осуществлялась при поддержке Среды конструирования специализированных алгоритмов оптимизации[10].

Среда специализированных алгоритмов оптимизации предоставляет библиотеку классов на языке C++, реализующих проблемно-независимую часть алгоритмов. В рамках исследования методики были реализованы классы на C++, унаследованные от стандартной библиотеки классов средства конструирования алгоритмов. Они описывали решение задачи, реализовывали целевую функцию и операции скрещивания и мутации.

В качестве исследуемой системы была рассмотрена часть реальной бортовой вычислительной системы. Исследуемая система состоит из процессора, на котором выполняются несколько функциональных задач и планировщика, распределяющего процессорное время между задачами, задачи могут обмениваться данными. Эта система моделировалась в среде ДИАНА. Функциональные задачи моделировались процессом FunctionalTask, а планировщик моделировался процессом Scheduler, наконец процесс Timer моделировал функции системных часов.

В качестве механизмов отказоустойчивости применялись: резервирование (для таймера), многоверсионное программирование и метод контрольных точек (для функциональных задач и планировщика). В планировщик и в набор функциональных задач вносились неисправности, приводящие к отказу компонентов системы. В таймер и в набор функциональных задач вносились неисправности, приводящие к искажению данных внутри компонентов системы. Количество функциональных задач варьировалось в диапазоне от 3 до 10. Количество итераций без улучшения целевой функции было равно 25.

Как генетический алгоритм, так и переборный используют одну и ту же целевую функцию, так что сложность каждой итерации у них одинакова, если пренебречь временем на мутацию и скрещивание.

В результате экспериментов с данной моделью было обнаружено, что количество итераций, необходимое для получения приемлемого по качеству решения при помощи предложенной методики существенно меньше, чем при полном переборе всех вариантов. И при полном

переборе и при использовании методики, достигался глобальный минимум, равный 0. Это соответствует исправлению всех неисправностей в системе. Результаты, полученные при экспериментах с различными наборами неисправностей, показаны в таблице 1. В первом столбце таблицы приведено количество функциональных задач в модели. Во втором столбце приведено среднее количество итераций генетического алгоритма для достижения минимума при различных наборах неисправностей. В третьем столбце приведена оценка количества итераций алгоритма полного перебора, которая не зависит от наборов неисправностей.

Таблица 1

Сравнение количества итераций алгоритмов для нахождения оптимального решения

Количество функциональных задач	Количество итераций генетического алгоритма	Оценка количества итераций алгоритма полного перебора
3	70	8192
5	360	524288
10	920	17179869184

5. Заключение

Эксперименты показали, что предложенная в статье методика работает корректно. В дальнейшем предполагается:

- Реализовать часть методики, относящуюся к автоматической генерации отказоустойчивой модели и её прогона, для автоматизации проверки соблюдения директивных сроков задач. Это позволит проводить гораздо большее число экспериментов за одно и тоже время.
- Исследовать поддержку в популяции большего количества различных решений. Это ухудшит скорость сходимости генетического алгоритма. Зато позволит в случае нахождения серии решений, в которых задачи не удовлетворяют директивным срокам, быстрее получить решения, в которых задачи удовлетворяют директивным срокам.
- Провести эксперименты не только с генетическими алгоритмами, но и с алгоритмами имитации отжига. Здесь пока неясно будет ли преимущество в скорости сходимости в случае использования алгоритмов имитации отжига.
- Оценить применимость методов многокритериальной оптимизации для нахождения компромисса между надёжностью и производительностью. На данный момент все характеристики системы, на которые не влияют механизмы отказоустойчивости, считаются фиксированными.

1. Stankovic J. A. Real-time Computing. // Byte Magazine. 1992. 17. N 8. P. 155-160.
2. Ammar H., Cukic B., Fuhrman C., Mili A. A Comparative Analysis of Hardware and Software Reliability Engineering. // Annals of Software Engineering. 2000. 10. N 1-4. P. 103-150.
3. Гладков Л.А., Курейчик В.В., Курейчик В.М. Генетические алгоритмы. / Под ред. В.М. Курейчика. – 2е изд., испр. и доп. – М.: ФИЗМАТЛИТ, 2006., 320 с.
4. Thompson, A. (1995). Evolving fault tolerant systems. In Proc. 1st IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95), pages 524-529. IEE Conf. Publication No. 414.
5. Hartmann,M. and Haddow, P. C. (2004). Evolution of fault-tolerant and noise-robust digital designs. IEE Proc. -Comput. Digit. Tech., 151(4):287-294.
6. Смелянский Р.Л. Анализ производительности распределенных микропроцессорных вычислительных систем на основе инварианта поведения программ. / Дисс. на соискание ученой степени доктора физико-математических наук. М.:МГУ, 1990
7. Shooman M.L. Reliability of computer systems and networks. John Willey & Sons Inc. 2002. 528 p.
8. Волканов Д.Ю., Шаров А.А. Программное средство автоматического внесения неисправностей для оценки надежности вычислительных систем реального времени с использованием имитационного моделирования // Методы и средства обработки информации. Труды второй Всероссийской научной конференции. - М.: Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 2005. - С.457-464.
9. A.Bakhmurov, A.Kapitonova, R.Smeliansky DYANA: An Environment for Embedded System Design and Analysis, in Proc. of 5-th International Conference TACAS'99, Amsterdam, The Netherlands, March 22-28, 1999. Springer (LNCS Vol.1579), pp.390-404
10. Калашников А.В., Костенко В.А., Маркин М.И. Средства конструирования итерационных алгоритмов для решения задач комбинаторной оптимизации // Искусственный интеллект, 2004., No 2, С.91-95.

Ломазова И.А.

АДАПТИВНОЕ И ДИНАМИЧНОЕ МОДЕЛИРОВАНИЕ ПОТОКОВ РАБОТ НА ОСНОВЕ ВЗАИМОДЕЙСТВУЮЩИХ СЕТЕЙ ПЕТРИ[1]

РГСУ, Москва, i_lomazova@mail.ru

Управление потоками работ (англ. – *workflow*) – это процесс распределения задач и документов между исполнителями – сотрудниками и/или прикладными компьютерными программами – с целью выполнения некоторого конкретного задания. В системах управления потоками работ четко прописаны бизнес-процессы организации, ее функциональные бизнес-правила, способы и сроки передачи документов и предоставления отчетности.

В настоящее время задаче моделирования потоков работ уделяется большое внимание в литературе по информационным технологиям. Создание и анализ модели системы является первым и важным шагом в разработке системы управления потоками работ – это позволяет обнаружить узкие места и ошибки на ранних стадиях разработки.

Можно выделить два основных направления в моделировании бизнес-процессов: модели, основанные на правилах (в частности, правилах нечеткой логики), и процессно-ориентированные модели, основанные на сетях Петри.

Преимуществами процессно-ориентированного подхода является сочетание наглядности графического представления с математически строгим определением модели и наличие развитых методов и средств анализа и верификации сетей Петри, в частности, методов и программных систем на основе методологии проверки модели (model checking).

Обыкновенная сеть Петри[2]представляет собой конечный ориентированный граф с вершинами двух типов, называемых соответственно *позициями* (изображаются кружками) и *переходами* (изображаются прямоугольниками). Дуга в сети Петри направлена или от позиции к переходу или от перехода к позиции. Вершина p называется *входной* для вершины t, если некоторая дуга направлена от p к t, и *выходной*, если дуга направлена от t к p. Через $\bullet p$ будем обозначать множество всех входных вершин для вершины p, а через $p\bullet$ – множество выходных для p вершин. *Разметка* M сети Петри определяется как отображение множества позиций P в множество

[1] Работа поддержана РФФИ, проекты 07-01-00702 и 09-01-00277

неотрицательных целых чисел. Говорят, что позиция p содержит k фишек при разметке M, если $M(p)=k$.

Поведение сети Петри определяется следующим образом. Переход t называется *активным* при разметке M, если любая входная позиция перехода t содержит хотя бы одну фишку при разметке M. Активный переход t может *сработать*, забирая по одной фишке из каждой своей входной позиции и добавляя по одной фишке в каждую свою выходную позицию. *Исполнение* для сети Петри определяется как последовательность срабатываний переходов, начиная с заданной *входной разметки*.

Для моделирования бизнес-процессов выделен специальный класс сетей Петри – WF-сети [1]. Сеть Петри называется WF-сетью, если выполняются следующие три условия:

1. Имеется одна позиция-источник $i \in P$, такая что $\bullet i = \varnothing$;
2. Имеется одна позиция-сток $o \in P$, такая что $o \bullet = \varnothing$;
3. Каждая вершина $x \in P \cup T$ находится на некотором пути от позиции i к позиции o.

WF-сеть описывает обработку отдельных экземпляров работы (это может быть заказ, изделие, документ и т.п.). WF-сеть имеет одну входную позицию (i) и одну выходную позицию (o), так как каждый экземпляр работы в WF-сети создается в момент появления в системе управления потоками работ и удаляется после его полной обработки. Таким образом, WF-сеть описывает жизненный цикл экземпляра работы. Третье условие в определении добавлено для того, чтобы исключить “лишние” задачи и/или условия, т.е. задачи и условия, не задействованные в обработке экземпляров.

Для WF-сетей разработаны специальные методы анализа. Одним из основных критерием корректности WF-сети является свойство бездефектности (правильной завершаемости).

WF-сеть PN называется *бездефектной*, если

1. Для любого состояния M, достижимого из начального состояния i, существует последовательность срабатываний, переводящая состояние M в заключительное состояние o.
2. Состояние o является единственным состоянием, которое достижимо из состояния i и содержит хотя бы одну фишку в позиции o.
3. В сети PN с начальной разметкой i нет мертвых (т.е. никогда не срабатывающих) переходов.

Свойство бездефектности разрешимо для WF-сетей. Более того, в классе всех WF-сетей выделен специальный подкласс

структурированных WF-сетей, которые являются бездефектными по построению.

Однако в области моделирования потоков работ сетями Петри остается еще много нерешенных проблем. Недостатком графической модели является жесткая изначально заданная структура сети. Между тем на практике в ходе эксплуатации и использования в систему управления бизнес-процессами, как правило, приходится вносить изменения. Перенастройка системы вручную в таких ситуациях может быть сделана только специалистом, и является очень трудоемкой и затратной процедурой. К тому же внесение изменений в систему может приводить к появлению ошибок в ее функционировании.

Можно выделить следующие важные требования к моделям систем управления бизнес-процессами ([8]):

- *Гибкость*: возможность исполнения нечеткой или не полностью определенной модели.
- *Адаптивность*: обеспечение перестройки системы в ходе работы в зависимости от некоторых условий и реакции на исключительные ситуации (включая не прогностические).
- *Динамичность*: возможность гибкой перестройки модели в соответствии с новыми внешними спецификациями (реинжиниринг процессов).

Гибкость модели потока работ может обеспечиваться за счет рассмотрения иерархических сетей Петри, когда более высокий уровень соответствует большему уровню абстракции. При этом любой уровень сети может быть выполнен [4].

Одним из способов адаптивного моделирования потоков работ является использование вложенных WF-сетей позволяющие динамически менять структуру системы, причем выполнять эти изменения автоматически [3,5-6]. Вложенные сети обеспечивают возможность подключения тех или иных подпроцессов на этапе выполнения, а добавление операций над сетевыми фишками позволяет динамически строить новые подпроцессы из имеющихся. В [5] доказана разрешимость свойства бездефектности для вложенных WF-сетей.

Однако вложенные WF-сети не решают проблему динамичности системы. Так, добавление новых операций в уже существующий бизнес-процесс (таких как, например, выполнение дополнительной проверки или заполнение еще одной формы отчета) требует, вообще говоря, перестройки сети.

Для решения этой проблемы мы предлагаем представлять «добавки» в уже существующий процесс в виде отдельной WF-сети, взаимодействующей с прежним бизнес-процессом. Похожий подход

использован в [7] для моделирования межорганизационных потоков работ. Поясним это на примере добавления подпроцесса *WF1* с начальной позицией-источником *i1* и позицией-стоком *o1* к уже существующей сети *WF*. Пусть в соответствии со спецификацией требуется, чтобы подпроцесс *WF1* запускался не ранее того, как в процессе выполнения *WF* сработает переход *t1*, и подпроцесс *WF1* должен быть завершен до срабатывания перехода *t2* в сети *WF*.

Перестроим сеть *WF1*, добавив к ней переходы *ti* и *to* такие, что $\bullet ti = \bullet to = \varnothing$; $ti\bullet = \{i1\}$ и $to\bullet = \{o1\}$. Срабатывание переходов *ti* и *to* в *WF1* будет соответствовать запуску и завершению соответствующего сети *WF1* подпроцесса. Пометим эти переходы метками *labi* и *labo* соответственно. В свою очередь переходы *t1* и *t2* пометим метками *lab1* и *lab2.*

Для организации асинхронного взаимодействия сетей *WF* и *WF1* построим управляющую сеть *C*. Сеть *C* будет состоять из двух фрагментов. Фрагмент, обеспечивающий взаимодействие перехода *t1* (в сети *WF*) и перехода *ti* (в сети *WF1*) состоит из двух переходов *t1'* (помечен меткой *lab1*) и *ti'* (помечен меткой *labi*) и одной позиции *s*. Полагаем $\bullet t1' = ti'\bullet = \varnothing$; $t1'\bullet = \bullet ti' = \{s\}$. Второй фрагмент строится аналогично. Выполнение взаимодействующих сетей выполняется по правилам горизонтальной синхронизации для вложенных сетей Петри, когда два перехода помеченные дополнительными метками *lab* и *lab* должны срабатывать одновременно (см. [3]).

Таким образом, добавление нового подпроцесса не меняет структуру исходной сети, не добавляет новых ее выполнений, но может, вообще говоря, сделать некоторые прежние выполнения невозможными.

В общем случае система взаимодействующих WF-сетей состоит из одной главной WF-сети, конечного числа WF-сетей подпроцессов и управляющей сети. Главная WF-сеть – это WF-сеть, в которой некоторые переходы могут быть помечены метками синхронизации. WF-сети подпроцессов – суть WF-сети, дополненные позициями запуска и завершения процесса. Кроме того, мы дополняем каждую такую сеть новой специальной позицией *init*. Позиция *init* является входной для запускающего перехода и выходной для завершающего. Она нужна для того, чтобы обеспечить возможность нескольких выполнений одного и того же подпроцесса в ходе одного выполнения главной сети. WF-сеть подпроцесса с добавленными запускающим и завершающим переходами и позицией *init* будем называть расширенной WF-сетью подпроцесса. Некоторые позиции в WF-сетях подпроцессов

также помечены метками синхронизации. Управляющая сеть – это однобезопасная обыкновенная сеть Петри.

В начальной разметке главная WF-сеть содержит одну фишку в позиции-источнике, а каждая из расширенных WF-сетей подпроцессов – по одной фишке в своей позиции *init*. Заключительным состоянием системы назовем состояние, в котором главная WF-сеть содержит одну фишку в позиции-стоке, а каждая из WF-сетей подпроцессов – по одной фишке в своей позиции *init*. Управляющая сеть в начальной разметке фишек не содержит. Заключительная разметка управляющей сети может быть произвольной.

Будем говорить, что система WF-сетей *бездефектна*, если

1. Главная WF-сеть и все нерасширенные WF-сети подпроцессов, входящие в систему, являются бездефектными.
2. Для любого состояния M, достижимого из начальной разметки системы, существует последовательность срабатываний, переводящая состояние M в заключительную разметку системы.
3. Если некоторая разметка системы достижима из начальной разметки и содержит фишку в позиции-стоке главной WF-сети, то эта разметка является заключительной.

В этом определении мы не требуем отсутствия мертвых переходов, как в случае определения бездефектности «обычных» WF-сетей. «Лишние» переходы – это плата за возможность не перестраивать систему заново.

Свойство бездефектности для систем взаимодействующих WF-сетей разрешимо. Темой дальнейших исследований будет выявление структурных характеристик, обеспечивающих бездефектность системы взаимодействующих потоков работ.

1. ван дер Аалст В., ван Хей К. Управление потоками работ: модели, методы и системы. М. : Физматлит, 2007. - 315 с.
2. Котов В.Е. Сети Петри. М.: Наука, 1984.
3. Ломазова И.А. Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой. М.: Научный мир, 2004. 208 с.
4. Flores-Badillo M., López-Mellado E., Padilla-Duarte M. Modelling and simulation of workflow processes using multi-level Petri nets. Proc. of the 4th International Workshop on Enterprise & Organizational Modeling and Simulation (EOMAS-2008), Montpellier, France, June 16-17, 2008, CEUR Workshop Proceedings.
5. van Hee K., Lomazova I. A., Oanea O., Serebrenik A., Sidorova N., Voorhoeve M. Nested nets for adaptive systems. 27th Int. Conf. on

Application and Theory of Petri Nets and Other Models of Concurrency, LNCS 4024, 241--260. 2006,
6. Lomazova I. A.. Nested Petri nets for adaptive process modeling. Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday, LNCS 4800, 413-426. 2008.
7. Prisecaru O., Jucan T. Interorganizational workflow nets; a Petri net based approach for modeling and analyzing interorganizational workflows. Proc. of the 4th International Workshop on Enterprise & Organizational Modeling and Simulation (EOMAS-2008), Montpellier, France, June 16-17, 2008, CEUR Workshop Proceedings.
8. Rajabi B.A., Lee S.P. Change management in buisiness process modeling based on object oriented Petri nets. International Journal of Buisiness, Economics, Finance and Management Sciences, 1:1:72-77. 2009.

Миков А.И., Замятина Е.Б.

ПРОБЛЕМЫ РЕАЛИЗАЦИИ СИСТЕМЫ РАСПРЕДЕЛЕННОГО МОДЕЛИРОВАНИЯ С УДАЛЕННЫМ ДОСТУПОМ

Кубанский государственный университет, Краснодар,
alexander_mikov@mail.ru
Пермский государственный университет, Пермь,
e_zamyatina@mail.ru

Введение

Рост сложности задач, которые требуют компьютерной обработки, развитие новой высокопроизводительной вычислительной техники (ЭВМ кластерной архитектуры, многоядерные процессоры) и появление новых технологий (Grid-технологий, Cloud-технологий) являются фактором, способствующим развитию распределенного имитационного моделирования[1]. Известно, что метод имитационного моделирования является широко применяемым, а иногда и единственным, методом исследования при решении задач производственной сферы (САПР), бизнеса, здравоохранения, транспорта и т.д.

Тем не менее, распределенное моделирование еще не получило широкого распространения, а причина этому отсутствие удобного графического интерфейса, программных средств визуализации, удаленного доступа[1]. Кроме того, разработка программного обеспечения, поддерживающего распределенное имитационное моделирование и удаленный доступ, требует решения ряда проблем, которые освещаются ниже. Речь пойдет о системе распределенного имитационного моделирования Triad.Net[2], разработка и реализация компонентов которой ведется группой преподавателей и студентов Пермского и Кубанского госуниверситетов.

Архитектура распределенной системы Triad.Net

Распределенная система имитации Triad.Net включает следующие компоненты: компилятор, ядро, графический редактор, подсистему отладки, подсистему валидации, подсистему синхронизации распределенных объектов модели, подсистему балансировки[3,4], подсистему организации удаленного доступа[5], подсистему защиты от внешних и внутренних угроз[6], подсистему автоматического доопределение модели. Компоненты распределенной системы имитации представлены на рис.1.

Расскажем коротко о назначении каждого из компонентов: TriadCompile (компилятор языка Triad, переводит описание имитационной модели с языка Triad во внутреннее представление); TriadDebugger (отладчик, использует механизм информационных процедур алгоритма

исследования, локализует ошибки и вырабатывает рекомендации для их устранения на основании правил из базы данных); TriadCore (ядро системы, включает библиотеки классов основных элементов модели), TriadEditor (редактор моделей, предназначен для работы с моделью как в удаленном, так и локальном режимах, локальный режим предполагает работу с системой в том случае, если нет удаленного доступа), TriadBalance (подсистема балансировки), TriadSecurity (подсистема безопасности), TriadBuilder (подсистема автоматического доопределения частично описанной модели), база данных, где хранятся экземпляры элементов модели.

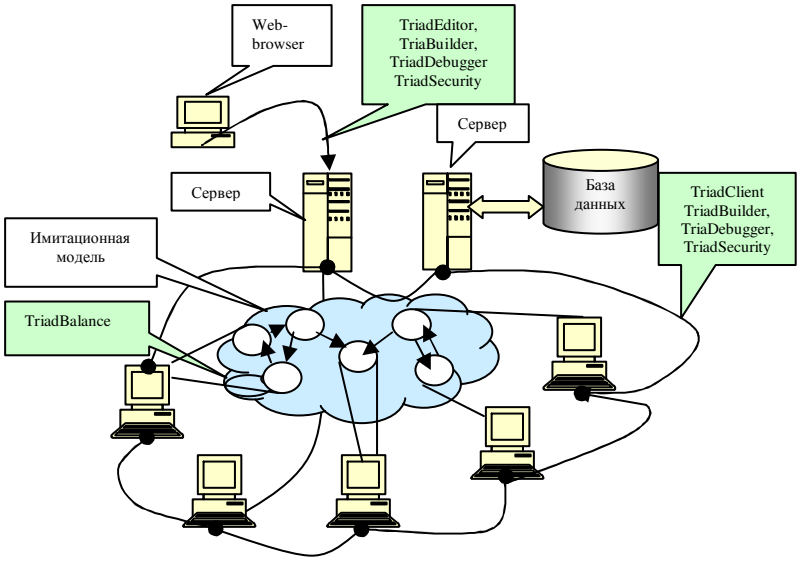


Рис.1. Отображение распределенной имитационной модели на архитектуру кластерной ВС и компоненты системы моделирования

TriadEditor-компонент подготовки имитационной модели

Имитационная модель может быть написана на алгоритмическом языке Triad или подготовлена с помощью графического редактора. Подготовка модели может быть выполнена в удаленном (в этом случае взаимодействие с моделью осуществляется через Интернет) и в локальном режимах. Для создания, редактирования и запуска имитационных моделей используется компонент TriadEditor. Этот компонент может встраиваться как в Web-страницу (удаленный режим), так и в обычное приложение Windows (локальный режим).

Работа в удаленном режиме выполняется через Интернет-портал, основанный на метаданных. Для организации Интернет-портала, для его администрирования и настройки интерфейса для конкретного пользователя разработано специальное инструментальное средство IMPortal, которое может существовать как самостоятельное приложение. Изменяя метаданные, можно изменить структуру портала, наполнить портал содержимым из любой другой области знаний. Отдельные компоненты имитационной модели могут быть выполнены на различных серверах. Количество серверов для выполнения имитационных моделей не ограничено и зависит от загрузки системы.

Представление имитационной модели на языке Triad

Расскажем коротко о представлении имитационной модели на языке моделирования Triad[7]: модель имеет иерархическое представление, на каждом уровне иерархии модель определяют слоем структур (описание компонентов моделируемого объекта и связей между ними), слоем рутин (описание поведения компонентов объекта), слоем сообщений (описание структуры сложных сообщений, которыми обмениваются различные компоненты модели). Рутина представляет собой совокупность событий, которые планируют друг друга. Компоненты модели одного уровня иерархии взаимодействуют друг с другом посылкой сообщений. Среди событий рутины выделено входное событие, обрабатывающее входные сообщения. Ниже приведено описание слоя структур модели, которая представляет собой фрагмент компьютерной сети, состоящей из рабочих станций, передающих сообщения друг другу и маршрутизаторов, отвечающих за нахождение пути передачи данных.

```
Type Router,Host; integer i;
    M:=dStar(Rout[5]<Pol[4]>);  M:=M+node Hst[8]<Pol>;
M.Rout[0]=>Router;
    for i:=1 by 1 to 4 do
        M.Rout[i]=>Router;
    M:=M+edge(Rout[i].Pol[1]—      Hst[2*i-2]);
        M:=M+edge(Rout[i].Pol[2]—      Hst[2*i-1]);
    endf;
    for i:=0 by 1 to 7 do
        M.Hst[i]=>Host;
    endf;
```

Рис.2. Слой структур модели с указанием семантических типов

Информационные процедуры и особая лингвистическая и программная единица *условия моделирования* реализуют алгоритм исследования имитационной модели. Алгоритм исследования осуществляет

сбор информации о функционировании модели, выполняет обработку данных по окончании моделирования, определяет, следует ли завершить сеанс моделирования или продолжить его. Информационные процедуры следят за изменением переменных, за событиями и полюсами рутин и являются единственным программным средством, имеющим одновременный доступ к нескольким рутинам.

Для взаимодействия с удаленной моделью используют программные механизмы информационных процедур.

TriadBuilder – компонент автоматического доопределения имитационной модели

Имитационная модель в Triad может быть описана не полностью, на ранних стадиях проектирования исследователю предоставляется возможность описать модель с существенной степенью приближения к оригиналу. Так, например, исследователь имеет возможность не включать в описание модели поведения ряда элементов модели, а указать их принадлежность к некоторому семантическому типу (например, *процессор, маршрутизатор, счетчик, триггер* и т.д., семантические типы определяют смысловую нагрузку того или иного объекта имитационной модели). На рис.2. можно видеть включенные в описание модели семантические типы Router и Host.

На компонент TriadBuilder возлагается ответственность анализа внутреннего представления имитационной модели. Если анализ показывает, что поведение какого-либо элемента модели пользователем не описано, TriadBuilder добавляет его к описанию модели (автоматическое доопределение модели). Для поиска экземпляра рутины используют базу знания, построенную в виде онтологий. В этих онтологиях описывают семантические типы, отношения наследования между ними, а так же множества соответствующих этим типам экземпляров рутин, и семантической информации, необходимой для проверки условий доопределения. При доопределении модели используют критерии специализации (семантические типы вершины в слое структур и типа экземпляра рутины совпадают), конфигурации (проверка количества входов и выходов вершины и экземпляра рутины), и декомпозиции (совпадение графа окружения).

TriadDebugger – верификация и валидация модели

После того, как модель построена, следует проверить, не содержит ли она ошибок и соответствует ли ее поведение тому, как это представляет себе исследователь, создавший модель. Для верификации и валидации модели в Triad.Net разработан специальный компонент TriadDebugger. Для обнаружения ошибок и их локализации TriadDebugger использует механизм информационных процедур. Информационные процедуры дают исследователю возможность определить, соответствует ли

значение некоторой переменной конкретному значению, произошло ли некоторое событие в заданный промежуток времени, пришло ли на входной полюс конкретное сообщение и т.д. Локализацию ошибок компонент TriadDebugger выполняет по правилам из базы знаний. База знаний представлена в виде онтологий, соответствующим различным типам ошибок.

Синхронизация распределенной модели

Синхронизация объектов распределенных имитационных моделей выполняется с помощью специальных алгоритмов: консервативного и оптимистического.

Консервативный алгоритм придерживается безопасной политики продвижения времени (время продвигается от события к событию, события сохраняют порядок причинности). Согласно этому классу алгоритмов рутина объекта модели выполняется только в том случае, если соблюдены все условия для её безопасного выполнения, т.е. существует уверенность в том, что объект не получит сообщения em_j с временем $t_j < t_i$ (t_i – время уже выполненного события рутины объекта).

Оптимистический алгоритм позволяет выполнять рутины до тех пор, пока не произойдет ошибка, то есть пока рутиной не будет получено сообщение em_j с меньшим временем, чем время выполненных событий $e_i(t_j<t_i)$. В этом случае реализуется протокол, который выполняет откат и исправляет ошибку. В случае применения консервативного и оптимистического алгоритмов в Triad.Net существуют механизмы, которые «расширяют горизонт времени», т.е. определяют «безопасные» события. В случае оптимистических алгоритмов сокращается количество откатов, в случае консервативных – происходит более «оптимистичное» продвижение рутины. «Безопасные» события определятся в Triad.Net в результате предварительного анализа топологических характеристик имитационной модели, исследования графа последовательности выполнения событий или в результате использования правил, которые задает пользователь на основании знаний о поведении конкретной имитационной модели.

TriadBalance – равномерное распределение нагрузки на вычислительные узлы

Во время функционирования распределенной имитационной модели возможно возникновения дисбаланса нагрузки. Это явление объясняется гетерогенностью имитационной модели, гетерогенностью вычислительной системы, на которой выполняется распределенный имитационный эксперимент, изменением нагрузки стороннего приложения, которое функционирует на тех же вычислительных узлах, что и распределенная модель. Для предотвращения негативного действия дисбаланса (снижение скорости выполнения имитационного

эксперимента) используют специальное программное обеспечение – компонент TriadBalance. Компонент является мультиагентным приложением. Известно, что мультиагентный подход дает возможность создавать гибкие, масштабируемые и эффективные приложения (возможность изменения функциональности приложения за счет добавления или замены агентов).

Перед выполнением имитационного эксперимента отдельные компоненты имитационной модели распределяют по вычислительным узлам (на одном узле могут располагаться один или несколько компонентов). Статическая балансировка выполняется на основании топологических характеристик модели. Динамическая балансировка (во время выполнения имитационного эксперимента) выполняется агентами анализа, распределения, миграции на основании данных, полученных от агентов датчиков. Агенты-датчики наблюдают за состоянием вычислительной системы (нагрузка на вычислительный узел, объем доступной оперативной памяти, пропускная способность линий связи) и компонентов имитационной модели (частота выполнения событий в рутине, частота прихода сообщений на входные полюса рутины и т.д.). Агент анализа определяет, возникла ли ситуация, требующая перераспределения ресурсов, агент распределения выявляет на критическом узле компонент-аутсайдер, агент миграции выполняет перенос компонента-аутсайдера на другой узел. Агенты анализа и распределения являются когнитивными. Правила, на основании которых они действуют, учитывают знания о топологии вычислительной системы и имитационной модели, знания о поведении конкретной имитационной модели (например, известно, что в определенный момент времени два компонента могут интенсивно обмениваться информацией). Агент-датчик имитационной модели использует для сбора статистики механизм информационных процедур. На основании данных агентов-датчиков вычислительной системы подсистема визуализации предоставляет пользователю информацию о функционировании вычислительной системы и карту отображения модели на вычислительные узлы в том числе.

TriadSecurity – подсистема безопасности распределенной имитационной модели с удаленным доступом

Удаленный доступ к модели предполагает, что имитационная модель должна быть защищена от внешних атак злоумышленников. Поскольку компоненты модели распределены по отдельным вычислительным узлам и обмениваются сообщениями во время выполнения, злоумышленник может вмешаться в работу имитационной модели, заменив нужное сообщение, а результаты проведения имитационного эксперимента в этом случае станут недостоверными. Защита модели от атак злоумышленников возлагается на подсистему безопасности

TriadSecurity. В модуле TriadSecurity выделяют 2 уровня: а)внешний уровень модуля защиты системы Triad.NET (TriadOutSecurity); б)внутренний (TriadInSecurity). Внешний уровень модуля защиты находится на сервере для публикации Web-приложений, взаимодействует с модулем IMPortal и предназначен для защиты системы от несанкционированного вмешательства через удаленный доступ. Внутренний уровень находится на сервере БД и предназначен для защиты системы от попыток взлома изнутри, то есть от сообщений злоумышленника, направленных на разрушение таких компонентов системы, как TriadCore и др. (этот уровень находится в стадии проектирования). При реализации компонента TriadSecurity реализован мультиагентный подход. Ключевым агентом внешнего уровня TriadSecurity является агент выявления атак, взаимодействующий со специальной базой сигнатур атак (под сигнатурой понимаем признак трафика известных атак: Fuzzy; UDP-Bomb, WinNuke, Land и т.д.).

Заключение

В статье описаны проблемы, с которыми разработчики распределенных систем имитации сталкиваются при их реализации, и решения, которые были приняты для преодоления этих проблем. При реализации распределенной системы Triad.Net и ее подсистем активно использованы методы искусственного интеллекта.

1. Perumalla K., and Fujimoto R. Interactive Parallel Simulations with the JANE Framework//Workshop on Parallel and Distributed Simulation(http://www.cc.gatech.edu/ computing/ pads/papers.html)
2. Миков А.И., Замятина Е.Б., Фатыхов А.Х. Система оперирования распределёнными имитационными моделями сетей телекоммуникаций. Труды Второй Всероссийской научной конференции «Методы и средства обработки информации». М.: Изд-во МГУ, 2003 г.
3. Миков А.И., Замятина Е.Б., Осмехин К.А. Динамическое распределение объектов имитационной модели, основанное на знаниях. //Proceedings of XIII International Conference “Knowedge-Dialogue-Solution” KDS, ITHEA, Sofia, 2007.Vol.2.,pp.618-624
4. Миков А.И., Замятина Е.Б., Козлов А.А. Оптимизация параллельных вычислений с применением мультиагентной балансировки.//Труды конференции ПАВТ-2009. стр. 599-604
5. Mikov A., Zamyatina E., Firsov A. Software for Remote Parallel Simulation.// International Journal «Information Theories & Applications», Vol.14, № 4, 2007, pp. 389-395
6. Миков А.И., Замятина Е.Б., Панов М.П. Мультиагентная система защиты имитационной модели с удаленным доступом.//Труды Международных научно-технических конференций «Интеллектуальные системы» (AIS’07) и «Интеллектуальные САПР» (CAD-2007),-М.: Физматлит, 2007, Т1. сс.323-330
7. Mikov A.I. Simulation and Design of Hardware and Software with Triad// Proc.2nd Intl.Conf. on Electronic Hardware Description Languages, Las Vegas, USA, 1995. pp. 15-20.

Никольская Ю.Н., Леницкий Д.С., Левченко Н.Н., Окунев А.С.

ВОЗМОЖНОСТЬ ПРИМЕНЕНИЯ ПАРАЛЛЕЛЬНОЙ ПОТОКОВОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ ДЛЯ СОЗДАНИЯ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Институт проблем проектирования в микроэлектронике РАН, г. Москва, unn@burcom.ru, lenitsky@burcom.ru, nick@ippm.ru, oku@ippm.ru

В настоящей статье рассматриваются варианты применения параллельной потоковой вычислительной системы (ППВС) в качестве вычислительной системы реального времени (ВСРВ).

При проектировании вычислительной системы реального времени необходимым условием является полная информация о событиях, которые могут произойти на объекте, а также о критических сроках обслуживания каждого из этих событий. ВСРВ - это аппаратно-программный комплекс, реагирующий в предсказуемое время на непредсказуемый поток внешних событий.

К ВСРВ предъявляются следующие требования:

- система должна успеть отреагировать на событие, произошедшее на объекте в течение времени, критического для этого события. Величина критического времени для каждого события определяется объектом и самим событием, и, естественно, может быть разной, но время реакции системы должно быть определено при создании системы для каждого вида события;
- система должна успевать реагировать на одновременно происходящие события. Даже если два или более внешних события происходят одновременно, система должна успеть среагировать на каждое из них в течение интервалов времени, критического для этих событий;
- отсутствие реакции на событие в предсказанное время считается ошибкой ВСРВ.

Различают системы реального времени двух типов – системы «жесткого» реального времени и системы «мягкого» реального времени. В системах «жесткого» реального времени, таких, например, как бортовые системы управления, системы аварийной защиты, регистраторы аварийных событий, нельзя допускать никаких задержек реакции системы ни при каких условиях, иначе результаты могут оказаться бесполезны в случае опоздания или может произойти катастрофа в случае задержки реакции.

Системы «мягкого» реального времени характеризуются тем, что задержка реакции не критична, хотя и может привести к увеличению стоимости получения результатов и снижению производительности системы в целом.

Основными параметрами ВСРВ являются время реакции системы и время переключения контекста.

Время реакции системы – это интервал времени от события на объекте (сигнал прерывания) до выполнения первой инструкции в программе обработки этого события. Обычно время реакции систем на прерывание составляет порядка 4-10 мкс.

Время переключения контекста – это время, необходимое для корректной остановки выполняемой задачи с сохранением данных, необходимых для возобновления последующего выполнения задачи, а также время подготовки, загрузки и запуска новой задачи (в частности, если задача была прервана, сохраненные данные записываются из памяти в процессор). Эта задержка должна быть мала и детерминирована. Это время является одной из важнейших характеристик ВСРВ. Обычно время переключения контекста в ВСРВ составляет величину порядка 10-20 мкс.

Кроме того в ВСРВ должны присутствовать:

- система приоритетов и алгоритмы диспетчеризации;
- механизмы межзадачного взаимодействия;
- средства для работы с таймерами.

В разрабатываемой ППВС вычислительный процесс управляется потоком данных. После трансляции, программа, написанная для выполнения на ППВС, представляет собой направленный граф потока данных. В узлах этого графа располагаются операторы, выполняющие определенные операции над поступающими на их входы данными. Оператор запускается на исполнение (активируется) в том случае, если на его входах присутствует пакет данных. Операторы могут быть одновходовыми, двухвходовыми, многоходовыми. Оператор представляет собой последовательную программу, результаты которой передаются по дугам графа к следующим операторам. Данные в вычислительной среде передаются от оператора к оператору в виде токенов. Токен представляет собой специальную структуру данных. В ней кроме самого данного указывается контекст, в котором используется это данное, адрес оператора и вход, на который направляется это данное.

Контекст и адрес оператора формируют *ключ оператора*, который уникально характеризует его расположение в графе потока данных виртуального пространства задачи.

ППВС представляет собой многоядерную масштабируемую вычислительную систему, в которой между ядрами в системе передаются токены (рис. 1). Кроме ключа оператора токен включает в себя кратность, маску и набор служебных полей. Коммутация между ядрами осуществляется на основе значения номера ядра, вырабатываемого блоком хеширования на основе функции распределения из полей ключа и маски токена.

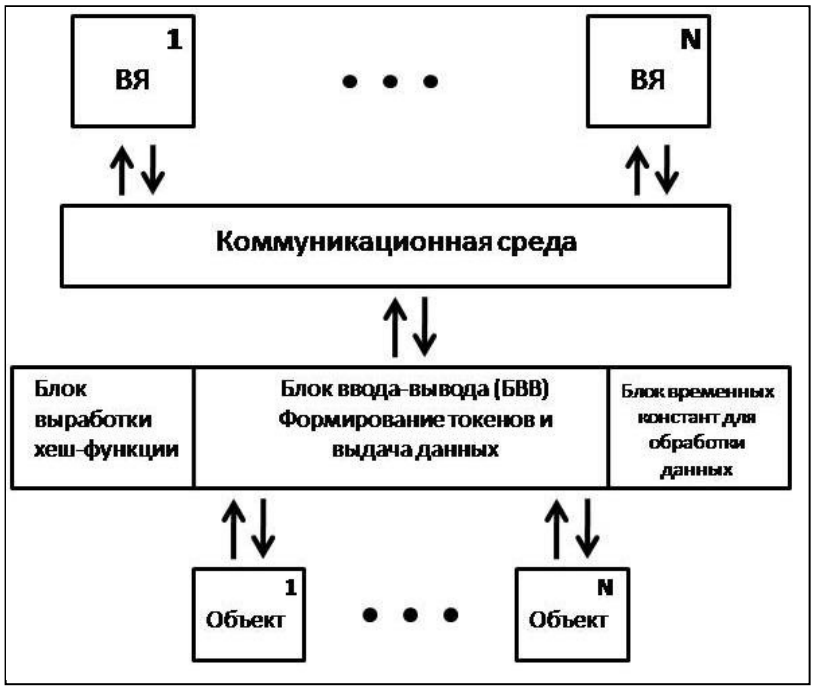


Рис. 1. Структура ППВС для систем реального времени

Ядро вычислительной системы конструктивно состоит из (рис. 2):

- модуля ассоциативной памяти (МАП) в котором происходит сопоставление токенов по определенным правилам и формирование пакетов (структуры данных, которая содержит операнды, необходимые для активации программы узла);
- исполнительного устройства (ИУ), в котором в соответствии с программой узла происходит обработка данных пакета и генерация новых токенов;

- блока хеширования, в котором осуществляется определение номера ядра для передачи токена, созданного в исполнительном устройстве;
- внутреннего коммутатора, который передает токен либо во внешнюю сеть, либо в «свой» МАП.

Кроме того в представленную схему ППВС вошли средства для работы с таймерами, которые позволяют использовать ее при работе в системах реального времени. Параллельная потоковая вычислительная система строится как совокупность вычислительных ядер (ВЯ), соединенных коммуникационной средой.

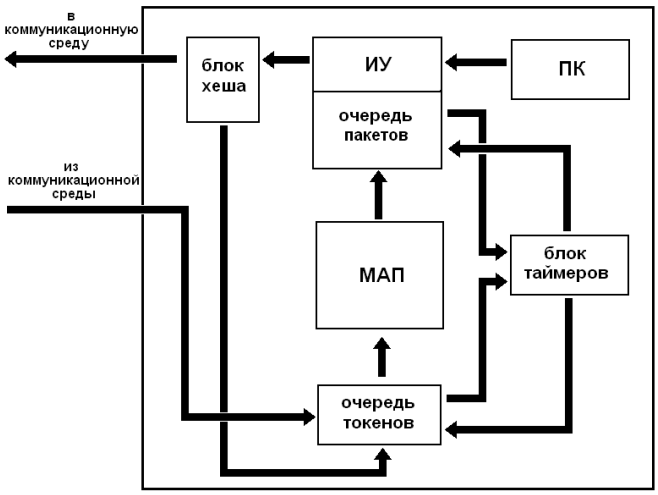


Рис. 2. Структурная схема вычислительного ядра (ВЯ)
ИУ - исполнительное устройство;
ПК – память команд;
МАП – модуль ассоциативной памяти.

Организация вычислительного процесса в ППВС осуществляется аппаратно реализованной ассоциативной памятью. Основными функциями АП ППВС являются:

- выявление готовых к исполнению операторов, АП в динамике выявляет параллелизм программы и автоматически распараллеливает ее выполнение на уровне операторов;
- быстрая аппаратная синхронизация операторов по данным.

Система приоритетов

В предлагаемой архитектуре система приоритетов может быть реализована следующим способом. Приоритет определяется на входе системы в зависимости от номера объекта реального времени. В этом случае при инициализации системы в блок ввода-вывода, в котором формируется входной токен, номеру объекта должен быть поставлен в соответствие номер приоритета и записан в специальное поле токена. В дальнейшем, в течение времени решения задачи, этот приоритет будет сохраняться при выдаче токенов из всех выполняемых операторов. Таким образом, все токены, относящиеся к данной задаче, имеют одинаковый приоритет. Однако есть возможность при определенных условиях изменять приоритеты для подзадач или даже отдельных узлов.

Необходимо отметить также, что при равенстве приоритетов преимущество имеет токен, у которого временной интервал до критической точки является наименьшим.

В ППВС имеются два блока, через которые возможно осуществлять прием на обработку токенов и пакетов в соответствии с установленными приоритетами - это буфер токенов на входе АП и буфер пакетов на входе ИУ. Всякий пришедший токен или пакет записываются в очередь в соответствии с приоритетами и временными интервалами.

Диспетчеризация

ППВС является асинхронной недетерминированной системой, поэтому достаточно сложно определить на каком этапе решения находится задача и, в частности, время окончания задачи. Существует несколько аппаратных способов решения этой задачи, которые в настоящее время исследуются на модели.

Механизмы взаимодействия между процессами

В ППВС разработан механизм передачи данных между процессами. Существует особый тип токена - токен «Косвенность», с помощью которого возможно осуществлять передачу данных между процессами.

Токен «Косвенность» предназначен для изменения значения ключа у совпавших с ним токенов. В поле данных этого токена хранится ключ того процесса или задачи, куда следует передать данное. При совпадении в АП токена, данное которого следует передать, с токеном «Косвенность» формируется новый токен, адрес оператора которого берется из поля данных токена «Косвенность», а данное - из совпавшего с ним токена. При помощи маскирования полей ключа и использования механизма кратности таким способом можно передавать множество данных с помощью токена «Косвенность».

Средства для работы с таймерами

Такие инструменты, как средства работы с таймерами, необходимы для систем с «жестким» временным регламентом. Эти средства, являющиеся базовыми механизмами, как правило, позволяют:

- измерять и задавать различные промежутки времени;
- генерировать прерывания по истечении временных интервалов или наступления какого-либо события;
- создавать разовые и циклические «будильники».

Кроме того, почти в каждой ВСРВ можно найти целый набор дополнительных, специфических только для нее механизмов, касающихся системы ввода-вывода, управления прерываниями и т.д.

В ППВС есть специальный токен «Счетчик», который можно использовать для определения общего времени наступления события. Кроме того в разрабатываемом блоке таймеров, входящего в состав ВЯ, аппаратно поддерживается работа с временными интервалами.

В заключение следует отметить, что эффективность использования ППВС в системах реального времени обеспечивается за счет следующих свойств:

1. Благодаря модели вычислений и предложенной ее аппаратной реализации экстрагируется практически весь параллелизм, заложенный в задаче.

2. Существует аппаратная поддержка распределения ресурсов вычислительной системы и синхронизации выполнения процессов во время вычислений.

3. При работе в системе реального времени синхронизация вычислительного процесса с внешними событиями осуществляется без потери производительности и с наименьшим временем реакции.

4. Обеспечена структурная надежность вычислительной системы.

5. В системе реализован широкий диапазон масштабируемости.

Работа поддержана грантом РФФИ 08-07-12100–офи.

1. Бурцев В.С. Информационно-вычислительные системы с автоматическим распараллеливанием вычислительных процессов // Труды II Международная конференция "Параллельные вычисления и задачи управления" - PACO ' 2004, пленарный доклад.
2. Никольская Ю.Н., Змеев Д.Н., Левченко Н.Н., Окунев А.С., Петрищев Д.В. Инструментальный испытательный комплекс для вычислительной системы новой архитектуры с автоматическим распределением ресурсов// Информационные технологии в науке, образовании, телекоммуникации и бизнесе. (IT + S&E'06), Гурзуф, 2006.

Рябых Н.Г.

СИНТЕЗ УСТОЙЧИВЫХ РАССИНХРОНИЗОВАННЫХ ИТЕРАЦИОННЫХ ПРОЦЕССОВ МЕТОДОМ ПРЕ- И ПОСТ-КОДИРОВАНИЯ

Московский физико-технический институт, Долгопрудный,
e-mail: nikolai.ryabykh@gmail.com

1. Введение. В последние годы все более «очевидной» для специалистов в области теории управления, передачи информации и др. становится идея применения асинхронных систем вместо традиционных «синхронных» систем. Асинхронный подход может найти широкое применение в задачах, связанных с многокомпонентными (многопроцессорными) вычислениями – например, в задаче позиционирования и определения параметров полета космических кораблей, когда данные снимаются и обрабатываются одновременно с нескольких датчиков.

Применение идеологии асинхронных процессов часто «упирается» в вопрос том, может ли асинхронная система, отвечающая своему синхронному прообразу, быть устойчивой при произвольном срабатывании ее подсистем. Ответ на поставленный вопрос в общем случае отрицателен, но преодоление данной, казалось бы, неразрешимой ситуации возможно, если обратиться к идее кодирования и декодирования информации.

2. Постановка задачи. Пусть имеется некоторая система W, состоящая из подсистем $W_1, \ldots, W_N$ (рис. 1).

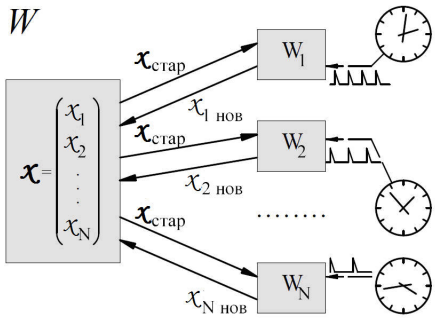


Рис. 1. Рассинхронизованная система

Тогда в общем виде состояние синхронизованной системы (все компоненты переключаются одновременно) записывается в виде

$$x_{n+1} = Ax_n + f_n,$$

(1)

где x_n, x_{n+1} – векторы состояния системы в моменты времени T_n и T_{n+1} соответственно, *A* – матрица перехода системы, f_n – вектор внешних воздействий.

Если одновременно переключаются не все компоненты, то динамика системы описывается уравнением

$$x_{n+1} = A_{\omega_n} x_n + f_{\omega_n},$$

(2)

где ω_n – множество номеров переключаемых в момент времени T_n компонент, A_{ω_n} – матрица, строки которой с номерами $i \in \omega_n$, совпадают с соответствующими строками матрицы $A = (a_{ij})$, а строки с номерами $i \notin \omega_n$ совпадают со строками единичной матрицы соответствующего размера.

Применение идеологии асинхронных процессов упирается в вопрос о том, может ли асинхронная вычислительная схема (2), отвечающая своему синхронному прообразу (1), сходиться к решению линейного уравнения

$$x = Ax + f$$

при произвольном выборе индексных последовательностей $\{\omega_n\}$.

Ответ на поставленный вопрос в общем случае отрицателен. Вместе с тем существуют классы матриц *A*, для которых сходимость синхронной процедуры влечет сходимость и ее асинхронного аналога. Так, такие классы образуют симметричные матрицы и матрицы с неотрицательными элементами [1], спектральный радиус которых строго меньше единицы ($\rho(A) < 1$). Таким образом возникает идея некоторым преобразованием привести исходную матрицу *A* к «хорошему» виду. Это становится теоретически возможным, если воспользоваться предложением Даймонда-Опойцева [2]: *если спектральный радиус* $\rho(A)$ *матрицы A размерности* $n \times n$ *строго меньше единицы, то для некоторого натурального* N ($N > n$) *найдется такая* $N \times n$ *матрица L и* $n \times N$ *матрица P, а также* $N \times N$ *матрица с неотрицательными элементами B, что справедливы следующие соотношения*

$$LA = BL,\ AP = PB,\ \rho(B) < 1.$$

3. Результат. Разработан алгоритм работы с синхронным итерационным процессом (1), позволяющий применить асинхронную схему вычислений. Разработана методика получения матриц B, P, L из матрицы *A* (кодирование), а также обратный переход (декодирование). Подробное описание алгоритма можно найти в [3].

Составлена программа на языке *C#*, иллюстрирующая работу алгоритма на примере матрицы поворота со сжатием (размерность 2×2).

4. Алгоритм работы с итерационным процессом (1).

1. Подготовка процедуры. Имея матрицу *A*, спектральный радиус которой строго меньше 1, находятся число *N*, а также матрицы L, P, B.

2. Пре-кодирование. Для нахождения решения уравнения

$$x = Ax + f$$

производится замена переменных $y = Lx$, $\tilde{f} = Lf$ (здесь векторы *y* и $\tilde{f}$ оказываются принадлежащими пространству достаточно большой размерности R^N).

3. Асинхронные вычисления с кодированными данными. Для построения последовательных приближений рассматривается асинхронная процедура

$$y_{n+1} = B_{\omega_n} y_n + \tilde{f}_{\omega_n}$$

Эта итерационная процедура в силу построения матрицы *B* будет сходящейся при любом выборе индексных последовательностей $\{\omega_n\}$ (см. [1]).

4. Пост-кодирование (декодирование). Для нахождения приближений к решению исходного уравнения достаточно произвести замену переменных $x_n = Py_n$.

5. Результаты работы программы. Рассмотрим итерационный процесс:

$$x_{n+1} = Ax_n + f_n$$

где

$$A = \lambda \cdot \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix},$$

$$\lambda = 0.99\,,\ \alpha = \frac{2\pi}{3},\ x_0 = f = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}.$$

Синхронный процесс (одновременный пересчет всех координат) сходится к точке:

$$x_* = \begin{pmatrix} 0.10734 \\ 0.39601 \end{pmatrix}.$$

Асинхронный процесс (последовательный пересчет каждой координаты) без кодирования:

$$x_{30} = \begin{pmatrix} -229.41761 \\ 183.64115 \end{pmatrix},\ x_* = \begin{pmatrix} \infty \\ \infty \end{pmatrix}$$

Процесс расходится, как и ожидалось.

Асинхронный процесс с кодированием:

$$N = 23$$

B – матрица размерности 23×23.

$$\rho(B) \le \|B\| = \max_j \sum_i \left| b_{ij} \right| = 0.9999 < 1$$

Результат после декодирования (количество итераций = 100):

$$x_* = \begin{pmatrix} 0.10733 \\ 0.39614 \end{pmatrix}.$$

Скорость и ход сходимости для синхронного и кодированного процессов можно наглядно сравнить на следующих графиках:

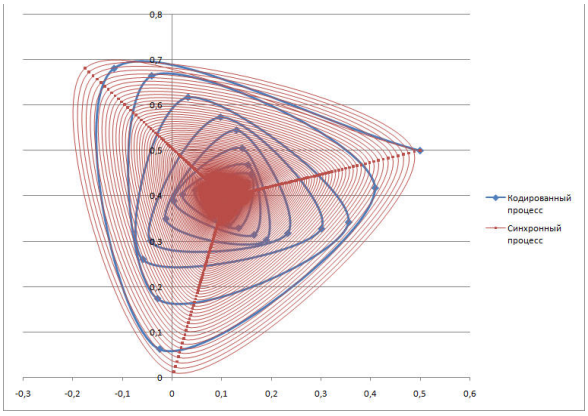


Рис.2. Ход приближений к решению для синхронного процесса и асинхронного процесса с кодированием

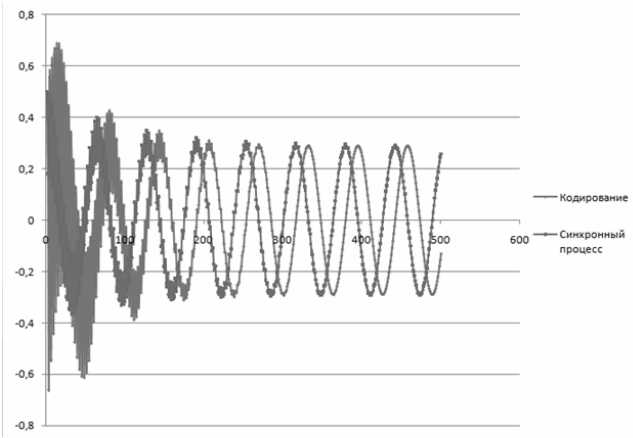


Рис.3. Сходимость процессов с периодическим внешним возмущением

6. Заключение. Главный результат работы состоит в следующем: доказана принципиальная возможность для произвольной матрицы со спектральным радиусом, не превосходящим 1, применить асинхронную процедуру вычисления решения уравнения $x = Ax + f$. Возможным достоинством предлагаемой процедуры является тот факт, что клиенты вычислительной процедуры – то есть процессоры, осуществляющие обработку векторов y_n – будут работать с «кодированными» данными, а их начальное кодирование и последующее декодирование будут

осуществляться «постановщиком» задачи. В ряде случаев такая «скрытность» для клиентов вычислительного процесса реальных данных может оказаться существенной.

Основным недостатком предлагаемой схемы является факт существенного роста размерности N матрицы B по сравнению с размерностью n матрицы A. Грубая оценка показывает, что

$$N \approx \frac{1}{(1-\rho(A))^{\frac{n-1}{2}}}.$$

В то же время следует отметить, что матрица B оказывается так называемой «разреженной» матрицей – каждая ее строка и столбец содержат не более $n+1$ ненулевых элементов. Как известно, это существенно упрощает работу с такими матрицами. Кроме того, есть основания полагать, что более «интеллектуальные» процедуры построения матрицы B могут существенно понизить значение размерности N.

Дальнейшие исследования должны проводиться в следующих направлениях:

- Варьирование свободными параметрами алгоритма кодирования с целью наилучшего приближения закодированного процесса к поведению системы, работающей по синхронной схеме;
- Попытка модифицировать алгоритм с целью уменьшения размерности матрицы B.

Описанный метод показывает, что, по крайней мере, теоретически построение гарантированно сходящихся асинхронных процедур возможно в достаточно широких ситуациях. Это означает, что необходимы интенсивные исследования в данном направлении, так как не исключена ситуация, что возможны и более простые методы синтеза гарантированно сходящихся асинхронных процедур.

1. Асарин Е.А., Козякин В.С., Красносельский М.А., Кузнецов Н.А. Анализ устойчивости рассинхронизованных дискретных систем. — М.: Наука, 1992. — 408 с.
2. Даймонд Ф., Опойцев В.И. Устойчивость линейных и разностных дифференциальных включений // Автоматика и телемеханика. — 2001. — № 5. — С.22–30.
3. Рябых Н.Г. Синтез устойчивых рассинхронизованных итерационных процессов методом пре- и пост-кодирования: Диссертация на соискание степени магистра. — МФТИ, 2008.

Смелянский Р.Л., Шалимов А.В.

К ВОПРОСУ О ЧАСТОТЕ ВЫПОЛНЕНИЯ ФРАГМЕНТОВ КОДА ПОСЛЕДОВАТЕЛЬНОЙ ПРОГРАММЫ

Фак-т. ВМК МГУ имени М.В.Ломоносова, Москва,
e-mail: ashalimov@lvk.cs.msu.su

1. Введение

Необходимость определения частотных характеристик фрагментов программы возникает во многих прикладных задачах:

- при оптимизации и распараллеливании программ, для того чтобы сконцентрировать усилия именно на активных участках программ.
- при разработке для операционных систем более эффективных стратегий управления и распределения ресурсов.
- при выборе резидентной части в системах реального времени.
- при построении оптимизирующих компиляторов. Затраты на компиляцию можно будет сократить, если компилятор будет оптимизировать не всю программу, а лишь активные ее части.
- распределении нагрузки в распределенных и многопроцессорных системах.
- для выделения наиболее интенсивно используемых фрагментов кода программы (везде далее просто фрагментов, если не оговорено противное), которые целесообразно подвергать наиболее тщательной проверке при тестировании.

Задача оценки частот выполнения фрагментов программы состоит из разметки исходной программы на фрагменты (здесь под фрагментами мы будем иметь ввиду линейные участки программы) и непосредственного определение частот выполнения этих фрагментов.

2. Постановка задачи

Дан текст последовательной программы.

Для каждого входного параметра известен диапазон значений и функция распределения значений на этом диапазоне.

Необходимо получить значения частот выполнения (*execution frequency*) фрагментов кода с заданной точностью [1,2].

3. Изменение исходной программы

Программа представляется в виде множества функций и множества входных параметров. Функция – это множество базовых блоков. Базовый блок – это пара (линейный участок, функция перехода). Линейный участок – это последовательность операторов, не содержащая операторов управления. Функция перехода определяет

условие перехода от одного базового блока к другому. Входной параметр программы - это переменная исследуемой программы, через которую в программу передают данные извне. Для подсчета частот исполнения преобразуем исходную программу без изменения её основной функциональности (Рисунок 1).

Программа изменяется следующим образом:

1. В программу добавляются функции инициализации, которые присваивают входным параметрам программы значения, сгенерированные по закону распределения их значений [3,4].
2. Команды чтения из внешних источников (с клавиатуры и т.п.) входных параметров программы заменяются на обращение к функциям инициализации.
3. В начало каждого базового блока добавлены счетчики. Счетчики увеличиваются на единицу каждый раз, когда происходит переход управления к базовым блокам. После окончания работы программы счетчики базовых блоков будут содержать количество передач управления на каждый базовый блок.

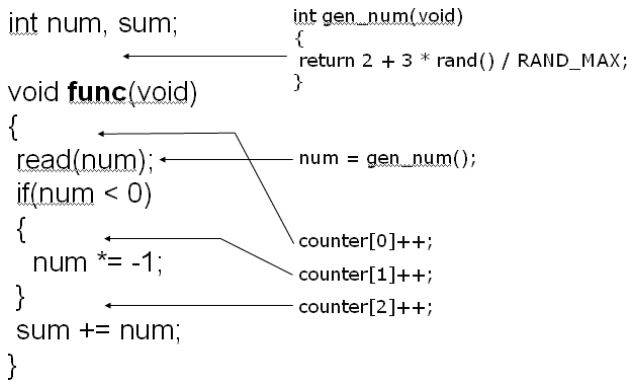


Рис.1. Изменение исходной программы

Не трудно видеть, что эти преобразования не затрагивают функциональность исходной программы, хотя несколько изменяют её временные характеристики.

4. Метод оценки частот выполнения фрагментов кода последовательной программы

Введем следующие обозначения:

- n – количество прогонов программы с различными значениями входных переменных.

- m – количество линейных участков (базовых блоков) программы.
- $П_i$ - i-тый прогон программы.
- n_{j_i} - число выполнений j-го линейного участка на i-ом прогоне программы (значение счетчика в конце прогона).

Произведем n прогонов программы ($П_1, П_2, \ldots, П_n$). Для каждого базового блока ($j = \overline{1,m}$) получим n значений каждого счетчика (Рисунок 2). Заметим, что результаты каждого прогона будут отличаться от результатов предыдущего, поскольку каждый раз при работе программы происходит генерация новых значений входных параметров.

	$П_1$	$П_2$	...	$П_n$	Средние частоты
1	n_{11}	n_{12}		n_{1n}	$N_1 = 1/n \cdot \sum_{i=1}^{n} n_{1i}$
2	n_{21}	n_{22}		n_{2n}	$N_2 = 1/n \cdot \sum_{i=1}^{n} n_{2i}$
...			...		...
m	n_{m1}	n_{m2}		n_{mn}	$N_m = 1/n \cdot \sum_{i=1}^{n} n_{mi}$

Рис.2. Результаты прогонов измененной программы

Таким образом для каждого j-го линейного участка есть ряд из n чисел $n_{j_1}, \ldots, n_{j_n}$, являющихся измерением частоты этого линейного участка. По этому ряду необходимо оценить частоту выполнения линейного участка, определить точность такой оценки.

Рассмотрим среднее арифметическое значений каждого счетчика линейного участка программы

$$N_j = 1/n \cdot \sum_{i=1}^{n} n_{ji} \qquad (1).$$

Покажем, что указанная величина является средней частотой линейного участка. На основании *закона больших чисел* и *центральной предельной теоремы* [4] можно сформулировать следующее утверждение.

Утверждение

1. Величина N_j (1) является средней частотой линейного участка и стремится к его частоте выполнения F_j при стремлении числа запусков программы n к бесконечности.

2. Среднюю частоту можно вычислить с заранее заданной точностью.

На практике применяется следующий алгоритм вычисления частоты линейного участка N_j с заданной вероятностью (надежностью) γ: $\mathrm{P}(\left|N_j - F_j\right| \le \varepsilon) = \gamma$.

1. Устанавливается счетчик числа прогонов $n = 0$.
2. Проводится один прогон измененной программы. По результату прогона запоминается число выполнений базового блока n_{ji}, $n = n + 1$.
3. Вычисляются среднее арифметическое частот и выборочная дисперсия:

$$N_j = 1/n \cdot \sum_{i=1}^{n} n_{ji} ,$$

$$S_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} (n_{ji} - N_j)^2 .$$

4. Проверяется выполнение условия

$$n > \left(\frac{u_{\frac{1+\gamma}{2}}}{\varepsilon} \right)^2 \times S_j^2 ,$$

где $u_{\frac{1+\gamma}{2}}$ - квантиль порядка $\frac{1+\gamma}{2}$ (см. таблицы в [3,4]).

5. Если условие выполнено, то задача определения средней частоты базового блока решена (т.е. N_j определена с заданной точностью). Иначе проводится очередной прогон программы (шаг 2).

Заметим, что аналогичным изменением программы можно определить средние значения различных характеристик программы. Например, количество переходов между базовыми блоками, среднее число шагов работы программы и т.п.

Важно отметить, что предлагаемая техника позволяет избежать исследования вопроса об условных и безусловных вероятностях переходов между базовыми блоками. Обычно в работах, где исследовались частотные характеристики программ, явно или не явно делалось предположение, что вероятность перехода к некоторому базовому блоку не зависит от того, по какому пути мы пришли к нему. Такое предположение является весьма спорным. Предложенная здесь техника учитывает зависимости в программе по управлению и их влияние на значения вычисляемых частот.

5. Практическое применение метода

На основе описанного выше метода была построена экспериментальная система для определения средних частот выполнения линейных участков программ, называемая FreqSys. Она используется в разрабатываемой авторами системе компактного представления программ для встроенных систем [7].

Входными данными системы FreqSys служат текст программы на языке Си, функции распределения входных параметров программы по диапазонам их значений и число прогонов программы. В результата работы системы имеем разбиение программы на базовые блоки и средние частоты их выполнения.

Покажем достоинства разработанной системы по сравнению со стандартными профилировщиками компиляторов llvm [5], gcc [6]. Для этого сначала опишем принцип работы профилировщиков этих компиляторов. Исследуемая программа модифицируется добавлением в начало каждого базового блока команды увеличения счетчика базового блока на 1. Измененная программа запускается и она начинает свое выполнение как обычно. В результате выполнения измененной программы генерируется файл профилировки. В этом файле находится информация о числе выполнений каждого линейного участка программы при данном запуске программы.

Преимуществами FreqSys по сравнению со стандартными профилировщиками являются:

1. Для получения средних частот выполнения линейных участков программы при использовании профилировщиков пользователю необходимо N ($N \to \infty$) раз запустить программу, ввести необходимые входные данные, получить результат. При использовании FreqSys пользователю необходимо лишь один раз задать распределения значений входных данных программы, и FreqSys в качестве результата

операции чтении входного параметра из внешнего источника будет выдавать значения, сгенерированные по заданным распределениям.

2. При использовании стандартных профилировщиков для получения средних частот выполнения линейных участков программы необходимо, чтобы программа была исполняемой, поэтому исходная программа не должна содержать неразрешенных ссылок на внешние переменные. Это ограничение отсутствует при использовании разработанной системы: наличие неразрешенных внешних переменных возможно, необходимо лишь задать распределения их значений.

3. При использовании стандартных профилировщиков для получения средних частот выполнения базовых блоков программы нельзя проанализировать отдельную функцию программы без запуска всей программы. Разработанная система позволяет выполнить анализ отдельно взятой функции. Для этого пользователю нужно лишь выбрать распределение значений для предлагаемых программой параметров: формальных аргументов анализируемой функции и используемых данной фукцией глобальных переменных.

Недостатками системы FrequencySystem по сравнению со стандартными профилировщиками являются:

1. Невозможность определения средних частот выполнения базовых блоков программ, которые имеют специфический формат входных данных, т.е. входные данные, для которых нельзя задать правильное распределение (например, jpeg-файлы, html-файлы).

2. Небольшая скорость работы, т.к. происходит выполнение программы в среде FreqSys, а не выполнение самой программы.

6. Заключение

В данной работе описан новый подход к определению средних частот выполнения фрагментов кода последовательной программы на основе функций распределения входных параметров. Важным достоинством предлагаемого подхода является то, что он позволяет избежать исследования вопроса об условных и безусловных вероятностях переходов между базовыми блоками программы. Обычно в работах, где исследовались частотные характеристики программ, явно или не явно использовалось предположение, что вероятности перехода к базовым блокам не зависят от того, по какому пути мы пришли к ним. Такое предположение является весьма спорным. Предложенная здесь техника учитывает зависимости в программе по управлению и их влияние на значения вычисляемых частот.

Показаны преимущества такого подхода по сравнению с существующими методами определения частот. Описанный метод применяется в разрабатываемой авторами системе компактного

представления программ для встроенных систем и показывает хорошие результаты работы [7].

1. Смелянский Р.Л., Гурьев Д.Е., Бахмуров А.Г. Об одной математической модели для расчета динамических характеристик программы. Программирование, N6, 1986
2. R.L. Smelianski, T. Alanko. On the calculation of control transition probabilities in a program Inform.Processing Letters N.3, 1986
3. Скрипкин В.А., Моисеенко Е.А. Математические методы исследования операций в военном деле. М.:Военное издательство министерства обороны СССР, 1979.
4. В.Ю. Королев Теория вероятностей и математическая статистика. М.: ТК Велби, Изд-во Проспект, 2008г.
5. Documentation for the LLVM System [HTML] (http://llvm.cs.uiuc.edu/docs/).
6. The GNU Compiler Collection [HTML] (http://gcc.gnu.org/)
7. Шалимов А.В. Метод компактного представления программ на основе частотных характеристик их поведения // Интеллектуализация обработки информации: Тезисы докладов Международной научной конференции / Крымский научный центр НАН Украины. - Симферополь, 2008

Харитонов В.Ю.

МОДЕЛИ СОГЛАСОВАННОСТИ ДЛЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ[2]

Московский энергетический институт (технический университет), Москва, kharitonovvy@gmail.com

1. Введение

Распределенные системы виртуальной реальности (РСВР) представляют собой один из наиболее динамично развивающихся классов распределенных систем (РС) [1,2]. Повышенный интерес к таким системам, в первую очередь, связан с качественно новыми возможностями, которые они предоставляют для человека. Прежде всего, пользователю предоставляется возможность «погружения» в некую *виртуальную среду*, достаточно точно имитирующую реальный мир. Пользователь может взаимодействовать с объектами виртуальной среды, как визуально, так и с помощью различных устройств ввода-вывода. Главным свойством РСВР является возможность взаимодействия множества географически удаленных пользователей в общей виртуальной среде, что может быть востребовано во многих прикладных областях.

Одной из важнейших задач при построении РСВР является создание максимально приближенной к реальным условиям виртуальной среды. Данное требование означает не только применение качественной трехмерной графики, но и обеспечение взаимодействия пользователей в виртуальном среде, максимально приближенного к взаимодействию в реальном мире при решении аналогичных задач. То есть речь идет о качественном характере взаимодействия пользователей. Однако возникает вопрос: что понимать под «качеством взаимодействия» и как можно его оценить.

В работе предлагается оценивать качество взаимодействия исходя из степени согласованности (*consistency*) состояний виртуальной среды, наблюдаемой различными пользователями. Рассматриваются различные подходы к определению понятия согласованности, а также предлагается модель согласованности для РСВР, основанная на принципе избирательной репликации данных (модель с «избирательной согласованностью»).

[2] Работа выполнена при поддержке Программы Рособразования «Развитие научного потенциала высшей школы» (проект 2.1.2/6718)

2. Согласованность в РСВР

2.1. Особенности РСВР

РСВР обладают всеми теми же свойствами, что и классические РС [3]:

- *отсутствие глобальных для всей системы часов* — все процессы системы взаимодействуют между собой асинхронно (но в системе может поддерживаться некоторая модель синхронизации);
- *отсутствие общей памяти* — все обмены данными ведутся только посредством передачи сообщений между процессами системы (хотя может существовать абстракция общей памяти);
- *географическая удаленность вычислительных узлов* — вычислительные узлы системы могут находиться как в пределах локальной сети, так и в масштабах глобальной сети Интернет;
- *автономность и неоднородность вычислительных узлов* — вычислительные узлы являются слабосвязанными, т.е. могут обладать различной производительностью и работать под управлением различных ОС.

Кроме того, у РСВР есть и свои особенности:

- *вычисления производятся в реальном времени* — это предъявляет особые требования к вычислительным узлам, которые должны успевать обрабатывать большие объемы информации за малое время, и к каналам связи, которые должны обеспечивать своевременный доступ к наиболее актуальному состоянию виртуальной среды;
- *вычисления носят специфический и итеративный характер* — большая их часть сводится к расчету глобального состояния виртуальной среды, которое постоянно меняется, причем не только в ответ на действия пользователей, но и с течением времени;
- *необходимость временной синхронизации* — хотя в системе нет глобальных часов, каждый из процессов имеет свои локальные часы и эти часы должны быть синхронизированы между собой.

2.2. Причинно-следственная согласованность

Данная модель согласованности вытекает напрямую из событийной модели распределенных вычислений [3], которая является удобным и хорошо зарекомендовавшим себя средством описания вычислений в РС. В рамках событийной модели распределенные вычисления рассматриваются как выполнение распределенной программы множеством процессов, входящих в РС. Выполнение каждого процесса можно рассматривать как последовательность *событий* двух типов: внутренние события и события приема/отправки данных [4]. Обозначим через e_i^l l-ое событие на i-ом процессе.

Событиями в РСВР могут быть создание объекта виртуальной среды, изменение состояния объекта и т.п. *Локальная история вычислений процесса* p_i — это последовательность событий $h_i = e_i^1 \; e_i^2 \ldots$, отражающая порядок выполнения локальных событий. Глобальная история вычисления есть множество $H = h_1 \cup \ldots \cup h_n$, включающее произошедшие во всех n процессах события.

Под локальным состоянием s_i процесса p_i будем понимать значения внутренних переменных процесса, а также последовательности сообщений, находящихся в каналах связи, инцидентных процессу. Глобальное состояние РС представляет собой совокупность состояний всех процессов $S = \{s_1,\ldots,s_n\}$. С течением времени, при возникновении новых событий, система последовательно переходит из одного *глобального состояния* в другое. Модель причинно-следственной согласованности подразумевает, что система может находиться только в *согласованных состояниях*, т.е. в состояниях, не нарушающих каузальных взаимосвязей между действиями различных процессов. Это, в частности, означает, что следствие не может возникать без причины, например, сообщение не может быть принято, не будучи посланным.

Другим условием причинно-следственной согласованности является соблюдение порядка передачи/приема сообщений: сообщения, имеющие причинно-следственную связь и посылаемые в определенном порядке с процессов (как с одного, так и с разных процессов) должны приниматься в том же порядке на процессе-приемнике. Из данного условия следует еще одна модель согласованности — *наблюдательная согласованность*.

2.3. Наблюдательная согласованность

Наблюдательная согласованность основана на понятии наблюдения. Наблюдение O_i — это упорядоченная последовательность событий, которая поступает в ходе работы РС на вход некоторого процесса-монитора p_i — особого типа процесса, способного записывать историю вычислений. Заметим, что в РСВР клиентские процессы пользователей являются одновременно участниками распределенных вычислений и мониторами.

Для обеспечения наблюдательной согласованности необходимо, чтобы последовательности событий, соответствующие наблюдениям O_1, O_2, $\ldots O_n$, производимым на сторонах процессов-мониторов p_1, p_2, $\ldots$ p_n, были идентичны друг другу и согласованы с историей вычислений H. На практике, в силу асинхронного характера вычислений в РСВР, обеспечить абсолютную наблюдательную согласованность невозможно и можно говорит лишь о частичной соответствии множеств O_1, O_2, $\ldots$ O_n. Однако ситуация может быть улучшена при использовании надежных соединений (протоколов), гарантирующих порядок доставки

сообщений, а также при использовании механизмов временной синхронизации.

2.4. Пространственно-временная согласованность

Другая модель согласованности основана на оценке степени соответствия локальных копий состояния виртуальной среды друг другу на процессах РСВР. В отличие от рассмотренных видов согласованности данная модель не учитывает причинно-следственные взаимосвязи между глобальными состояниями РСВР, а используется для сравнения самих состояний, с учетом специфики конкретной РСВР и восприятия пользователя. Для характеристики состояния виртуальной среды вводится понятие *вид*. Вид представляет собой изображение виртуальной среды, наблюдаемое с текущей позиции пользователя в виртуальном пространстве.

Для того, чтобы у всех пользователей возникало ощущение присутствия в общем для них виртуальном мире, их виды должны быть согласованы, т.е. процессы всех пользователей должны иметь непротиворечивые данные о текущих состояниях объектов виртуальной среды. Объекты виртуальной среды могут быть как статическими (неподвижными), так и динамическими (двигающимися по определенным законам, в том числе, под управлением пользователей). При оценке согласованности видов виртуальной среды различных пользователей в первую очередь имеет смысл рассматривать динамические объекты.

Пространственно-временная согласованность подразумевает введение метрики, позволяющей непосредственно измерить ошибку в расчете состояний объектов. Так в [5] в качестве метрики используется мгновенное значение расстояния между локальным и удаленным положениями объекта. Более интересную метрику позже предложили Zhou и др. в [6], позволяющую оценить, как отсутствие согласованности может повлиять на восприятие пользователя. Их метрика, называемая *пространственно-временной несогласованностью*, основана на том, что пользователь, наблюдающий виртуальную среду, принимает решения, как исходя из положений объектов, так и из длительности, в течение которой они остаются в этих положениях. Если эта длительность меньше некоторого порога воспринимаемого человеком, то пользователи могут не заметить наличие несогласованности. Если же длительность достаточно велика, то пользователи заметят сложившуюся ситуацию и смогут попытаться на неё повлиять. Предложенная метрика выглядит следующим образом:

$$\Omega = \begin{cases} 0, \text{если } |\Delta(t)| < \varepsilon, \\ \int_{t_0}^{t_0+\tau} |\Delta(t)| dt, \text{если } |\Delta(t)| \geq \varepsilon, \end{cases} \quad (1)$$

где $\Delta(t)$ — расстояние между истинным положением объекта на процессе источнике данных и оценкой этого положения на удаленном процессе (ошибка в определении положения объекта); ε — минимальное расстояние между объектами виртуальной среды, различимое пользователем; t_0 — момент начала измерения, τ — длительность измерения.

Кроме того, существуют другие подходы к определению пространственно-временной согласованности. Так, автором данной статьи в [7] было показано, что большое значение при определении ошибок в определении положений движущихся объектов влияет не только положение самих объектов, но и относительное расположение объектов относительно друг друга в процессе их движения. Там же продемонстрировано, что при сложных видах движения, таких как движение с изменяющимся ускорением, имеет смысл использовать метрики, основанные на производных высших порядков, таких как скорость и ускорение.

3. Модель с «избирательной согласованностью»

В зависимости от типа, а также индивидуальных особенностей объектов виртуальной среды, их состояния могут включать множество различных атрибутов. Так, состояния статических объектов могут содержать координаты объекта, его размеры, форму, уровень детализации полигональной сетки и т. д. Состояния динамических объектов дополнительно могут включать скорость, ускорение, значения сил и моментов. Соответственно, для обеспечения полностью согласованного взаимодействия пользователей необходима согласованность (пространственно-временная) по всем этим параметрам.

Так как взаимодействие пользователей в РСВР происходит в реальном времени, то моделирование и, особенно, визуализация состояния виртуальной среды должно производиться с высокой частотой (по современным требованиям — не ниже 60Гц). Пересылка полных состояний объектов с такой частотой потребовала бы очень высокой пропускной способности каналов передачи данных, т.е. такой подход оказывается накладным. Кроме того, полное состояние объекта часто является избыточным и необходимо лишь на стороне процесса источника данных, непосредственно управляющего объектом, для точного моделирования его состояния. В большинстве случаев для остальных процессов нет необходимости полностью моделировать состояние

объекта, достаточно лишь поддерживать некую его упрощенную модель. Соответственно, для них не нужно передавать специфические характеристики объекта. Например, для динамических объектов можно не передавать значения сил и моментов, ограничившись лишь скоростью и ускорением. Основная идея принципа «избирательной согласованности» — выделить наиболее значимые для конкретной задачи аттрибуты состояния объекта и поддерживать согласованность только по ним. Данные атрибуты могут быть выбраны на основе различных соображений, среди которых наиболее важное — человеческое восприятие. Принцип допускает, что между локальными состояниями процессов пользователей могут быть различия, но они являются незаметными для восприятия. В результате появляется возможность найти компромисс между согласованностью и аппаратными ограничениями (такими как пропускная способность и латентность сети) [1,8].

Основные особенности модели «избирательной согласованности»:

- сокращение числа параметров состояния объектов виртуальной среды, подлежащих репликации;
- применение упрощенных моделей объектов на процессах-приемниках данных при одновременном использовании методов предсказания состояния объектов (см. [7,8]).

Другие аспекты предлагаемой модели:

- использование распределеннного графа сцены для хранения состояния виртуальной среды;
- применение гибридной стратегии репликации данных;
- применение механизмов временной синхронизации;
- управление совместным доступом к состоянию виртуальной среды;
- обработка взаимодействий объектов в виртуальном пространстве.

4. Заключение

Рассмотренные модели согласованности не являются взаимоисключающими, а, скорее, дополняют друг друга. В то время, как причинно-следственная и наблюдательная модели носят больше теоретический характер, вводя ограничения на порядок выполнения событий, пространственно-временная модель позволяет численно измерить согласованность с учетом специфики конкретной РСВР. При построении конечной РСВР следует принимать во внимание каждую из моделей.

Предложенная модель согласованности с избирательной репликацией данных положена в основу разрабатываемой в настоящее время автором программной библиотеки Terrain 3, которая позволит создавать РСВР для конкретных областей применения.

1. S. Singhal, M. Zyda. Networked virtual environments: design and implementation. ACM Press/Addison-Wesley Publishing Co., New York, NY, 1999.
2. Дзегеленок И.И., Харитонов В.Ю., Орлов Д.А. Вычислительные аспекты построения распределенных систем виртуальной реальности. Вестник Московского Энергетического института, №5, 2008 г. — М.: Изд. дом МЭИ, 2008. С. 27-32.
3. A. Kshemkalyani and M. Singhal. Distributed Computing: Principles, Algorithms, and Systems, Cambridge University Press, New York, NY, USA, 2008.
4. O. Babaoglu and K. Marzullo. Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. Distributed Systems, Addison-Wesley, Wokingham, 1993, pp. 55-96.
5. C. Diot and L. Gautier. A Distributed Architecture for Multiplayer Interactive Applications on the Internet. IEEE Networks magazine, Vol. 13, 1999, pp. 6-15.
6. S. Zhou, W. Cai, B. Lee, and S. J. Turner. Time-Space Consistency in Large-Scale Distributed Virtual Environments. ACM Trans. Model. Comput. Simul. Vol. 14, 1, 2004, pp. 31-47.
7. V.Y. Kharitonov. An Approach to Consistent Displaying of Virtual Reality Moving Objects. Proceedings of 3rd International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2008, IEEE Computer Society, Los Alamitos, CA, USA, 2008, pp. 390-397.
8. J. Smed, H. Hakonen. Algorithms and Networking for Computer Games. UK, Chichester: John Wiley & Sons, 2006.

Цветков В.В., Змеев Д.Н.

ПАРАЛЛЕЛЬНАЯ РАБОТА ИСПОЛНИТЕЛЬНЫХ УСТРОЙСТВ В ЯДРЕ ППВС

ИППМ РАН, г. Москва, vasily2002@mail.ru, zmejevdn@list.ru

Параллельная потоковая вычислительная система (ППВС) представляет собой гибридную архитектуру, совмещающую традиционный фон-неймановский принцип вычислений и принцип потока данных [1].

Программа ППВС – это совокупность фон-неймановских программ (узлов), между которыми организована передача данных с использованием специальных структур – токенов. В токенах, кроме данного представляющего операнд, указывается также адрес программы узла, на который направляется операнд и контекст.

Структурная схема ППВС представлена на рис.1

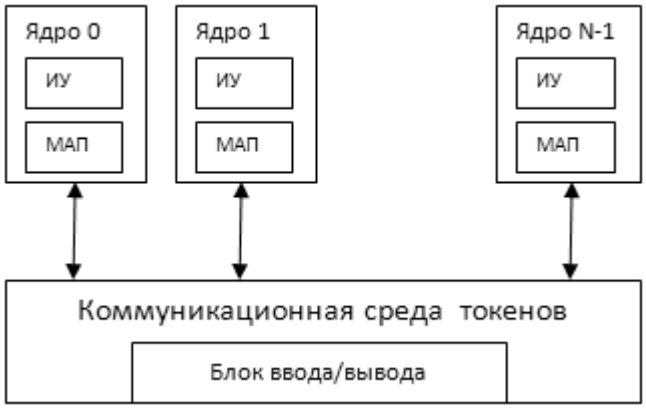


Рис. 1. Структурная схема ППВС

В ППВС реализована гибридная модель потока данных с динамически изменяемым контекстом.

Активация (запуск) программы узла происходит в режиме управления данными, то есть при наличии данных на всех входах узла. Аппаратная поддержка запуска узлов выполнена с помощью специальных блоков сопоставления операндов. В ППВС функцию блока сопоставления операндов выполняет ассоциативная память, поиск информации в которой осуществляется по ключу, содержащему адрес узла и контекст токена.

Базовым элементом ППВС является ядро, представляющее связанные между собой модуль ассоциативной памяти (МАП) и исполнительное устройство (ИУ). Ядра объединены через коммуникационную среду.

На рис.2 показана структурная схема ядра ППВС.

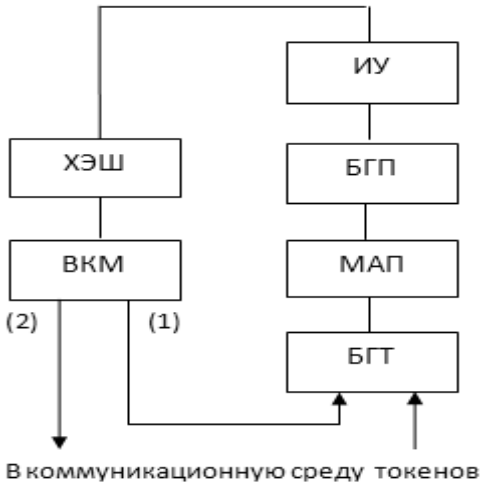


Рис. 2. Структурная схема ядра ППВС

Сформированные в модулях АП пакеты вместе с указателем на программу узла направляются для выполнения в исполнительное устройство «своего» ядра. В результате обработки пакета, на выходе исполнительного устройства генерируются токены, которые через внутренний коммутатор ядра (ВКМ) направляются либо в коммуникационную среду, либо в МАП «своего» ядра. По модулям АП токены распределяются через коммуникационную среду по номеру ядра, вычисляемому в блоке хэширования (ХЭШ) в соответствии с функцией распределения. Для выравнивания неравномерности потоков токенов и пакетов, в ядре перед ИУ и МАП установлены буфер готовых токенов (БГТ) и буфер готовых пакетов (БГП).

Конвейер ядра представляет собой циклическую структуру, в которой формируются независимые друг от друга пакеты данных. Выполнение операция на каждой ступени конвейера не требует какого-либо анализа результатов, полученных на предыдущих ступенях. При высоком уровне распараллеливаемости программы все ступени конвейера работают одновременно, и время выполнения программы определяется работой наиболее медленного звена конвейера.

В рассматриваемой гибридной системе потоковой модели вычислений на каждую операцию в АП, в результате которой формируется пакет, затрачивается более одной операции в ИУ для обработки данных этого пакета. Это приводит к тому, что ИУ в конвейере ядра обычно является наиболее медленным устройством, ограничивающим время выполнения программы. Преодолеть это ограничение можно, увеличивая количество ИУ в ядре, обеспечив при этом их параллельную работу. Принципиально, параллельную работу нескольких ИУ в ядре можно реализовать, поскольку формируемые в АП пакеты являются независимыми друг от друга и могут исполняться в произвольном порядке.

Целью работы является исследование функционирования ядра ППВС в конфигурации ядра, включающей несколько исполнительных устройств. В работе приводятся результаты моделирования работы одного ядра при различном количестве ИУ ($К_{ИУ}$) в ядре. Приводятся оценки коэффициента ускорения работы ядра при изменении количества ИУ. Коэффициент ускорения в ядре ($К_{У\_Я}$) определяется как отношение времени выполнения программы в ядре при одном ИУ ко времени выполнения программы при заданном количестве ИУ.

Принцип работы программной поведенческой модели основан на «событийном моделировании». Конфигурация системы состоит из связанных между собой объектов. Каждый объект представляет собой программную модель элементов системы, таких как модуль ассоциативной памяти (МАП), исполнительное устройство (ИУ), коммутаторы (КМ), и имеет определенные входные и выходные интерфейсы, что позволяет реализовывать и отлаживать его независимо от других. Объекты, получая на вход данные, выполняют их в соответствии с прописанным для этого «входа» алгоритмом. По завершении выполнения генерируется «событие» и передается на «выход», откуда оно попадает на «вход» к связанным с ним объектам. Этот цикл выполняется до тех пор, пока в системе не останется «событий» для исполнения. Структура «события» состоит из объекта, которому оно передается, времени начала его исполнения в объекте-получателе, а также передаваемых данных

Моделирование было проведено на пакете тестовых программ обработки изображений, включающем программы линейных фильтров, алгоритмов математической морфологии, скелетизации, программы преобразования цветовых форматов. Характеры зависимостей результатов моделирования для различных программ имеют похожий характер. В данной работе в качестве примера приводятся результаты моделирования работы ядра при изменении количества ИУ для программы преобразования цветового формата RGB в формат YCbCr.

На рис. 3 показана зависимость коэффициента ускорения работы ядра от количества ИУ.

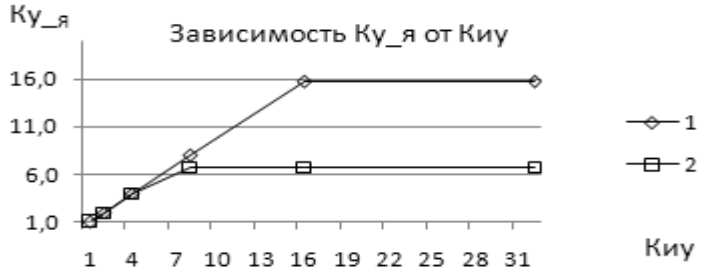


Рис. 3. Зависимость коэффициента ускорения от количества ИУ

На начальном этапе коэффициент ускорения растет по линейному закону. Этот участок зависимости соответствует случаю, когда время работы конвейера ограничено блоком исполнительных устройств.

Максимальный диапазон изменения количества ИУ, при которых наблюдается увеличение $К_{у\_я}$, определяется отношением времени обработки узла (включая время приема пакета в ИУ, предобработку и время выполнения программы узла) к интервалу времени между поступлениями пакетов в ИУ. На рис.3 этому случаю соответствует кривая (1). Максимальное количество активно работающих ИУ равно 16, которые обеспечивают 16-ти кратное ускорение работы ядра. Работа конвейера ограничивается в этом примере устройством МАП.

В некоторых случаях, в зависимости от параметров устройств конвейера, ограничение роста коэффициента ускорения может наблюдаться внутри указанного диапазона, если время работы конвейера начинает ограничиваться каким-либо другим устройством, у которого активное время работы превышает активное время работы МАП. В качестве примера, на рис.3 приведена кривая (2), полученная при моделировании работы ядра с увеличенным временем работы блока ХЭШ. Максимальное количество активно работающих ИУ в ядре и максимальный коэффициент ускорения в этом случае примерно равны 6, и определяются ограничением, которое накладывается блоком ХЭШ на время работы конвейера.

Устройство, ограничивающее время работы конвейера можно определить из диаграмм загруженности, которые приведены на рис.4.

Загруженность устройств определялась как отношение активного времени работы устройства к общему времени моделирования. Активное время оценивается как минимальное время, которое может

быть затрачено устройством для выполнения его доли работы в программе, при заданных установках параметров устройств конвейера.

На рис.4 приведены три диаграммы (a),(b) и (c). На диаграмме (a) показана загруженность устройств конвейера при одном ИУ в ядре. Эта диаграмма соответствует начальным точкам на кривых (1) и (2) на рис.3. На диаграмме (b) показана загруженность устройств конвейера при 16 ИУ. Эта диаграмма соответствует количеству ИУ, при котором прекращается рост коэффициента ускорения на кривой (1). На диаграмме (c) показана загруженность устройств при увеличенном времени работы блока ХЭШ. Диаграмма соответствует количеству ИУ, при котором прекращается рост коэффициента ускорения на кривой (2). Из рисунка видно, что при $К_{ИУ}$ равном 1, максимально загруженным устройством в конвейере является ИУ (рис.4a), а при $К_{ИУ}$ равном 16, максимально загруженным устройством, определяющим работу конвейера, становится АП (рис.4b). При увеличенном времени работы блока ХЭШ (рис.4c), полностью загруженным устройством в момент прекращения роста коэффициента ускорения становится ХЕШ.

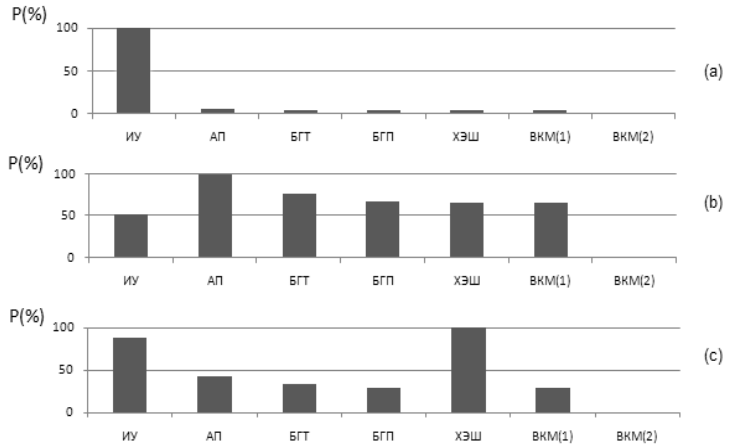


Рис. 4. Загруженность устройств в конвейере ядра

Принципиальным является случай, когда ограничения на время работы конвейера накладываются модулем АП. Максимальное количество активно работающих ИУ в ядре, а, следовательно, и значение максимального коэффициента ускорения, которое можно в этом случае достигнуть, определяется отношением среднего времени исполнения программы узла к темпу выдачи пакетов из МАП.

1. А.Л. Стемпковский, Н.Н. Левченко, А.С. Окунев, В.В. Цветков Параллельная потоковая вычислительная система – дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов, Информационные технологии №10, 2008г.

Чистолинов М.В.

ЯЗЫК ОПИСАНИЯ ШАБЛОНОВ ДЛЯ АНАЛИЗА ПОВЕДЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Фак-т. ВМК МГУ имени М.В.Ломоносова, Москва,
e-mail: mike@cs.msu.su

1.Введение

В работе предложен язык описания шаблонов для анализа поведения вычислительных систем. Поведение вычислительной системы представляется как совокупность всевозможных вычислений системы, заданных в форме трасс событий. Предложенный язык позволяет задавать спецификацию событий системы и спецификацию поведения системы в терминах расширенных регулярных выражений над заданными событиями. Особое внимание уделяется определению иерархии классов и атрибутов событий. Реализован механизм распознавания поведения системы по заданной спецификации.

Использование событий и регулярных выражений для описания поведения распределенных программ и вычислительных систем имеет давнюю историю. Так, ещё в конце 80-х годов был разработан ряд средств для отладки программ распределённых вычислительных систем (РВС), ориентированных на отслеживание последовательностей происходящих в системе событий, а не на отслеживание значений переменных в точке останова. Последовательности событий описывались с помощью различных расширений регулярных выражений (РРВ) [1,2,3].

Язык, предлагаемый в настоящей работе, может быть использован при анализе поведения вычислительных систем не только в процессе отладки системы, но и в процессе имитационного моделирования системы на этапе её разработки, и в процессе мониторинга за правильностью функционирования системы при её эксплуатации.

2. Основные определения

Событием называется любая ситуация в процессе функционирования системы, существенная с точки зрения наблюдателя за системой. Например, если наблюдение за системой происходит в терминах *состояний*, то событие – это смена наблюдаемых состояний. Событие – это единица наблюдения, фиксации и отображения хода вычислительного процесса [2]. У события, как правило, есть атрибуты, характеризующие его *тип*, *момент возникновения*, *точку возникновения* и т.д.

Трассой (историей) функционирования ВС (РВС) будем называть последовательность событий, которая формируется *наблюдателем* в процессе наблюдения за функционированием системы. Будем различать трассу событий, порождаемую компонентом системы или процессом, функционирующим в системе, и общую трассу, которая получается из трасс компонентов системы и процессов, путем их объединения в одну трассу.

Поведением будем называть совокупность трасс событий. Поведение может быть описано как частично упорядоченное множество событий. Частичный порядок в поведении возникает, например, из-за наличия нескольких независимых подсистем в РВС и отсутствия для них общих часов.

Спецификацией события будем называть описание состава и типов атрибутов события, а также расположения этих атрибутов в памяти.

Спецификацией поведения ВС будем называть описание *правильного* поведения ВС на некотором языке. Правильность поведения оценивается человеком, при задании спецификации. В данной работе под спецификацией поведения ВС понимается задание возможных трасс событий, которые будут задаваться при правильном функционировании ВС. Вид правильной трассы задается при помощи РРВ.

3. Особенности языка

Предлагаемый язык позволяет описывать классы событий, их атрибуты и спецификации потоков событий на основе расширенных регулярных выражений (РРВ). Существенными особенностями языка, по сравнению с другими языками спецификаций на основе РВ являются:

1. Явное описание и использование атрибутов событий.
2. Иерархия наследования классов событий.
3. Возможность задавать параметризованные (шаблонные) спецификации и выделять общие подвыражения спецификаций.
4. Использование срезов (выборок) событий в анализируемом потоке на основе диапазонов значений атрибутов событий.
5. Набор специальных метасимволов РВ, включающий параллельную композицию РВ, конъюнкцию (пересечение) РВ, операцию следования событий по времени, fifo и lifo обратные ссылки, и т.д.

Язык предназначен как для автономного использования, так и для использования в составе средств визуализации и анализа трасс событий средств моделирования [5].

4. Особенности потоков событий

Можно выделить следующие особенности потоков событий по сравнению с символьными строками, для работы с которыми традиционно используются регулярные выражения [4]:

1. Бинарное представление элементов потоков – событий – в том числе из-за сложной внутренней структуры событий.
2. Наличие атрибутов событий – тип, время, источник события и т.д. – и их существенное влияние на семантику событий.
3. Поток событий по сути является линеаризацией частично-упорядоченного множества событий от различных источников. Т.е. более правильно рассматривать поток событий не как линейную последовательность, а как частично-упорядоченное множество.
4. Ряд событий и атрибутов имеет специальную семантику, которая должна отражаться в языке спецификации для более естественного описания шаблонов:
    - события, определяющие *состояния* объектов-источников;
    - атрибуты, определяющие связи (ссылки) между событиями, например отображаемые стрелками в Визуализаторе;
    - атрибуты, определяющие ссылки на внешние трассы с дополнительной информацией о событии.

5. Основные элементы языка

Классы событий

Событие – с точки зрения языка - структура данных в памяти. Для событий в предметной области могут быть заданы атрибуты, определяющие их *тип* и *подтип*, а также любое количество других атрибутов.

Язык позволяет описать *классы событий*, *атрибуты событий*, *размещение событий* в памяти и *ограничения на значения атрибутов событий* для каждого класса событий. Это позволяет строить работу распознавателя независимо от конкретной семантики событий и от низкоуровневых средств работы с трассой событий.

Классы событий могут совпадать с типами/подтипами событий, а могут и расширять их на основе дополнительного учета значений других атрибутов. Для классов событий можно (и нужно) задавать иерархию наследования.

Распознавание классов событий строится так, чтобы каждое событие в потоке относилось в точности к одному классу (без учета

наследования классов событий и операции приведения события производного класса к базовому).

До этапа распознавания должен проводиться анализ "совместности" ограничений на значения атрибутов базовых и производных классов событий.

Описания классов событий должно (в том числе, с помощью специальных директив) однозначно определять размер события и расположение его атрибутов в памяти. Организация событий различных типов в форме union может быть однозначно представлена иерархией классов событий языка.

Регулярные выражения

Регулярное выражение (РВ) определяет основной элемент спецификации потока событий. РВ может строится из символов классов событий и из других (объявленных ранее) РВ. Таким образом существует возможность однократного описания общих подвыражений спецификации.

Спецификация потока событий – это специально выделенное РВ.

Описание регулярного выражения может быть параметризовано (в смысле шаблонов C++) следующими сущностями:

- конкретным классом (классами) событий;
- конкретными РВ;
- конкретными значениями атрибутов событий.

В РВ могут использоваться следующие стандартные метасимволы: '|', '*', '+', '?', '(', ')' (не сохраняющие скобки), '[:', ':]' (событийные классы или группы событий).

В РВ могут использоваться следующие специальные метасимволы (далее r1, r2 – некоторые РВ):

- r1][r2 - синхронная параллельная композиция (тасовка), которая предполагает распознавание всех "общих" символов потока как r1, так и r2;
- r1 & r2 - конъюнкция (пересечение языков) РВ r1, r2;
- r1 <time[, time']> r2 - операция "следования по времени" событий lastpos РВ r1 и firstpos РВ r2 (могут использоваться только если у событий имеется атрибут "время");
- (<имя>:r1) - именованные сохраняющие скобки;
- \<имя> - fifo обратные ссылки;
- /<имя> - lifo обратные ссылки;
- ^<класс события> - оператор явной квалификации класса события, по умолчанию <класс события> сопоставляется любому событию этого класса + (в соответствии с принципами полиморфизма) любому событию производного класса.

Также в РВ могут использоваться операторы выборки **slice**.

Пустые цепочки (ε) не могут использоваться в РВ. Вместо метасимвола произвольного события '.' может использоваться символ базового класса события без явной квалификации класса, например 'Event'.

Выборки

Выборка (или срез) выделяет "блок" (более точно - поддерево) в РВ и определяет ограничения на события, которые будут в этом блоке распознаваться. Выборка как бы определяет "фильтр" на рассматриваемые события. Фильтрация может производиться как по группам событий, так и по диапазонам значений атрибутов.

Типичная ситуация – выборка только событий на заданном интервале времени, или только событий одного класса (например, только событий изменения состояния).

Наличие в трассе событий, не попадающих в выборку, не рассматривается как ошибка (они "срезаются" до распознавания РВ).

В выборке может задаваться конкатенация условий по различным критериям/атрибутам. Выборки могут быть вложенными друг в друга на любом уровне РВ.

Семантически выборка для РВ эквивалентна параллельной композиции этого РВ и всех событий, "срезанных выборкой", появляющихся в любом количестве и в любом порядке.

До этапа распознавания должен проводиться анализ "совместности" условий вложенных выборок друг с другом и с ограничениями на значения атрибутов классов событий, использованных в РВ внутри выборок.

6. Синтаксис языка

Объявление класса событий:
event <имя> [: event <имя>];

Определение класса событий:
event <имя> [: event <имя>] {
(<описание атрибута> | <прагма event>);+
};

Описание атрибута:
<тип атрибута> <имя атрибута>;
<тип атрибута> ::= uint8 | int8 | ... | uint64 | int64 | float | double
<прагма event> ::= <прагма offset> | <прагма size> | <прагма restriction>

Задаем смещение между атрибутами в представлении события в памяти.

<прагма offset> ::= offset(<смещение в байтах>);

Задаем размер события в памяти.

<прагма size> ::= size(<размер в байтах>);

Задаем ограничения на значения атрибутов событий (прежде всего – type, subtype) для данного класса событий.

<прагма restriction> ::= restriction(<ограничение атрибута>[, <ограничение атрибута>]*);

<ограничение атрибута> ::= <имя атрибута>(<значение>[, <значение>])

Объявление РВ:

[template <<описание параметра>[, <описание параметра>]*>]
regexp <имя>;

Определение РВ:

[template <<описание параметра>[, <описание параметра>]*>]
regexp <имя> {
<описание РВ>
};

<описание параметра> ::=
event <имя параметра> |
regexp <имя параметра> |
<тип атрибута> <имя параметра>

Использование (подстановка) параметризованного РВ выглядит так:
<имя РВ><P1,P2,...,Pn> где Pi – фактические значения параметров.

В параметризованном РВ имена формальных параметров могут использоваться там же, где и соответствующие объекты.

Имена параметров, как и имена классов событий и regexp-ов должны начинаться с заглавной буквы.

Спецификация – выделенное РВ, которое выделяется ключевым словом spec, не имеет имени, параметров и не может иметь объявлений.

spec {
<описание РВ>
};

Описание выборок

Описание блока выборки может появиться для выделения любого подвыражения в описании РВ.

Описание выборки:

slice [(<ограничение выборки>[, <ограничение выборки>]*)] {
<описание РВ>
}

<ограничение выборки> ::= <ограничение атрибута> | <ограничение набора событий>

<ограничение набора событий> ::= events(<спецификация группы событий>)

где группа событий специфицируется как внутри скобок '[:', ':]'.

<ограничение атрибута> ::= <имя атрибута>(<значение>[, <значение>])

Ограничения выборки определяют тот набор событий, который будет рассматриваться внутри РВ в slice. Т.е. по сути задают фильтр на события.

Оператор slice без списка ограничений выполняет "автоматическую выборку" для РВ, т.е. обеспечивает "срезание" тех событий, которые не могут быть сопоставлены ни одному событию в РВ (с учетом вложенных slice-ов и ограничений на атрибуты).

Для разработанного языка создан транслятор и экспериментальная среда анализа трасс событий для комплексного стенда моделирования [5].

1. Бочков С.О. Смелянский Р.Л. Отладка программ в распределённых вычислительных системах // Программирование 1988. №4.
2. Bates P., Wileden J. High-level debugging of distributed Systems: The Behavioral Approach // J. Systems and Software.- 1983.- Vol.3, N.4-P.255-264
3. F. Baiardi N. De Francesco, S. Stefanini, G. Vaglini Development of a debugger for a concurrent language // IEEE Trans.on Soft. Eng. – 1986.- Vol.SE-12, N.4.- P.547-553
4. Дж. Фридл Регулярные выражения, изд. 1 Питер 2001
5. Баранов А.С., Грибов Д.И., Поляков В.Б., Смелянский Р.Л., Чистолинов М.В. Комплексный стенд математического моделирования КБО ЛА // Труды Всероссийской научной конференции "Методы и средства обработки информации" 2003 г., г. Москва) -М.: Издательский отдел факультета ВМиК МГУ, 2003. - С. 282-295

Чистолинов М.В.

АНАЛИЗ ЦИКЛОГРАММ ОБМЕНОВ ПО МКИО С ИСПОЛЬЗОВАНИЕМ АЛГОРИТМОВ НЕЧЁТКОГО СОПОСТАВЛЕНИЯ СТРОК

Фак-т. ВМК МГУ имени М.В.Ломоносова, Москва,
e-mail: mike@cs.msu.su

1.Введение

В отечественных авиационных и корабельных бортовых вычислительных комплексах широко используется мультиплексный канал информационного обмена (МКИО) [1]. Это канал с централизованным управлением, для которого характерно применение статических расписаний обменов, когда контроллер канала инициирует обмены между абонентами по заранее построенной циклограмме [2]. Как правило, циклограмма представляет собой так называемый большой цикл, разбитый на подциклы равной продолжительности, каждый из подциклов включает в себя цепочку обменов по каналу [2,3].
При большом количестве обменов задача ручного анализа результатов регистрации обменов становится слишком трудоёмкой, а результаты анализа – ненадёжными. Зарубежные и отечественные производители адаптеров МКИО [4,5,6] предлагают различные программные средства для сбора и визуализации информации об обменах на канале, однако эти средства не учитывают циклический характер обменов и не пытаются анализировать циклограмму обменов.

Мы предлагаем подход к анализу обменов по МКИО, ориентированный на «восстановление» циклической структуры расписания обменов и на сопоставление этой структуры с «эталонной». В качестве «эталонной» структуры может выступать как циклограмма обменов, построенная средствами САПР циклограмм [2], так и результаты анализа ранее зарегистрированной трассы обменов.
Особенность анализа и сравнения циклограмм обменов состоит в том, что невозможно «сопоставить» трассы обменов простым «наложением» из-за возможных допустимых различий с эталоном:

- сдвиг трасс обменов друг относительно друга по времени;
- «плавание» времени старта и длительности выполнения цепочек обменов;
- ошибки в обменах;
- пропуски и повторы обменов (или паузы вместо обменов);
- добавления новых обменов (в процессе сопровождения комплекса);

- относительное сжатие/растяжение циклограммы по времени.

- Указанные особенности создают сложности с выделением и идентификацией цепочек обменов, в частности они не позволяют непосредственно сравнивать цепочки обменов. Для преодоления этих сложностей нами была предложена схема кодирования цепочек обменов и использован аппарат нечёткого сопоставления строк [7].

2. Формальные определения и постановка задачи

Обмен M=<t,d,f,src,dst,sb1,sb2,sz,err> - атомарный завершенный обмен информацией между абонентами канала МКИО, в соответствии с протоколом обмена по ГОСТ Р 52070-2003 [1]. Каждый обмен характеризуется следующими атрибутами:

t - время начала обмена;

d - продолжительность обмена;

f - формат обмена по ГОСТ Р 52070-2003 - целое число в интервале [1,10];

src, dst - адреса абонента-отправителя и абонента-получателя данных в рамках обмена - целые числа в интервалах [0,31];

sb1, sb2 - подадреса абонента-отправителя и абонента-получателя данных в рамках обмена - целые числа в интервалах [1,30];

sz - количество слов данных, передаваемых в рамках обмена [1,32];

err - признак и тип ошибки, произошедшей при выполнении обмена.

Атрибуты сообщения M обозначаются как Mt, Md,…, Merr

Дополнительно используются обозначения для моментов времени начала и окончания обмена: Ms=Mt и Me=Ms+Md

При сравнении обменов друг с другом учитываются только значения атрибутов <f,src,dst,sb1,sb2,sz>. Эти атрибуты задают *тип обмена* Type(M).

- Трасса обменов – Tr = {M[1], M[2], …, M[n]}|Me[i] ≤ Ms[i+1] - конечная последовательность обменов. Между последовательными обменами в трассе обменов могут быть интервалы времени:
- I[n] = [Me[n], Ms[n+1]), I[0] = [0, Ms[1]).
- Цепочка обменов – C = {I[n-1],M[n],M[n+1],...,M[n+k],I[n+k]}|
- |I[n-1]|≥Tc, |I[n+k]|≥Tc, |I[j]|<Tc, для n≤j<n+k - последовательность обменов из трассы Tr, непосредственно следующих друг за другом, ограниченных справа и слева интервалами времени |I|≥Tc. В настоящей работе Tc=800мкс.
- Цепочка обменов может состоять из одного обмена.
- Большой цикл G={C[1],...,C[N]}|
- Tr={C[1],...,C[N],C[N+1],...,C[2N],...}, C[i]=C[kN+i], ∀ k≥0

- минимальная повторяющаяся последовательность цепочек обменов, образующая трассу Tr.
- Размер большого цикла N - количество цепочек обменов, образующих большой цикл.
- Период большого цикла P - продолжительность большого цикла по времени.
- Подцикл T[n]=[|T|*(n-1), |T|*n], |T| = P/N - интервал времени внутри большого цикла. Большой цикл разбивается на последовательность подциклов одинаковой продолжительности. Каждый подцикл содержит отдельную цепочку команд [2,3].

Циклограмма обменов Sc = {Is[0],C[1],...,C[N],Ie[N]}, Is[0]=[Ts[1],Cs[1]), Ie[N]=[Ce[N],Te[N]) - последовательность цепочек обменов формирующая большой цикл G и трассу обменов Tr, при их последовательном воспроизведении на канале МКИО. Is[0] - интервал времени от начала первого подцикла T[1] до старта первого обмена C[1]. Ie[N] - интервал времени между последним обменом цепочки C[N] и концом последнего подцикла T[n].

Циклограмма обменов – это большой цикл, для которого определена последовательность подциклов и границы подциклов по времени.

Сопоставление цепочек обменов Gamma(C1,C2) - последовательное сопоставление обменов, образующих цепочки C1, C2:

C1={M1[1],M1[2],...,M1[n1]}, C2={M2[1],M2[2],...,M2[n2]}.

Каждый обмен из одной цепочки может быть сопоставлен только одному обмену из другой цепочки. Некоторые обмены цепочек могут не участвовать в сопоставлении. Формально сопоставление цепочек это последовательность возрастающих пар индексов для сопоставленных обменов:

Gamma(C1,C2)={(i1,j1),...,(ik,jk)}, $i_l < i_{l+1}, j_l < j_{l+1}, l = 1..k$

Мера близости Omega(C1,C2) между цепочками обменов выражается через меру близости между обменами V(M1,M2) $\in$ [0,1] для сопоставленных обменов цепочек C1,C2:

Omega(C1,C2)=(1- $\sum_{l=1..k}$ (1-V(C1[i_l],C2[j_l]))/max(|C1|,|C2|)

Способ определения меры близости между обменами V(M1,M2) описан в разделе 4.

Решаемая задача:

Исходные данные: зарегистрированная трасса обменов на канале МКИО Tr={M[1],...,M[n]} .

Требуется: разработать набор алгоритмов и реализовать программные средства, обеспечивающие решение следующих задач:

• вычисление статистических характеристик трасс обменов Tr:

– состав обменов {M[i]} и типы обменов {Type(M[i])};
– состав цепочек обменов {C[i]};
– средняя продолжительность обменов $\sum$|M[i]|/n;
– загрузку каналов обменов $\sum$|M[i]|/P;
– состав и характеристики ошибок в обменах {Merr[i]};

• определение циклических характеристик обменов:
– состав большого цикла G={C[1],...,C[N]}, количество цепочек обменов в большом цикле N;
– продолжительность большого цикла по времени (период большого цикла) P;
– состав и расположение подциклов большого цикла {T[n]}, смещение по времени границы первого подцикла относительно начала трассы обменов T_start;

Имея большой цикл G, продолжительность подцикла |T| и T_start можно однозначно восстановить циклограмму обменов Sc.

• сравнение двух заданных циклограмм обменов для трасс Tr1, Tr2: G1(Tr1)={C1[1..k]} и G2(Tr2)={C2[1..k]} с использованием заданной меры близости между отдельными обменами V(M1,M2):
– мера близости между цепочками обменов Omega(C1,C2)=(1-$\sum$(1-V))/max(|C1|,|C2|);
– мера близости между фрагментами трасс Omega(Tr1,Tr2)= $\sum$(Omega(C1[i],C2[i]))/k.

Для сравнения циклограмм в трассах обменов Tr1, Tr2 предварительно должен быть определен состав цепочек и выделены большие циклы G1(Tr1), G2(Tr2). Считаем, что число цепочек обменов в сравниваемых циклограммах одинаковое, т.е. изменения в трассах не привели к изменению в количестве подциклов большого цикла.

3. Определение циклической структуры обменов

Общая идея определения циклической структуры большого цикла такова:

1. Выполняется анализ статистических характеристик обменов: общее количество обменов и типов обменов, количество ошибок, загрузка канала, адреса задействованных в обменах абонентов (с последующей детализацией по подадресам), задействованные в обменах форматы (с последующей детализацией по адресам) и т.д.

2. Последовательность обменов разбивается на цепочки, исходя из заданного Tc=800мкс. Для цепочек также определяется частотность их вхождения.

3. Выделяется самая редкая цепочка обменов и для неё выполняется последовательная проверка гипотез о периодичности

возрастающих интервалов между последовательными вхождениями этой цепочки в циклограмму.

Разработаны и программно реализованы следующие алгоритмы:

- Алгоритм разбиения последовательности обменов на цепочки;
- Алгоритм определения количества и последовательности цепочек в большом цикле (N);
- Алгоритм определения продолжительности большого цикла по времени (P);
- Алгоритм определения границ подциклов.

Все указанные алгоритмы, за исключением алгоритма определения последовательности цепочек в большом цикле, не более чем линейны по количеству анализируемых обменов в трассе.

Алгоритм определения последовательности цепочек в большом цикле квадратичен относительно просмотренного числа цепочек Nmax. Для определения периодически повторяющейся последовательности цепочек в общем случае, необходимо иметь границу сверху на число цепочек в большом цикле. Это позволяет избежать просмотра всей трассы и квадратичного роста сложности анализа.

Из практики известны следующие ограничения на анализируемые трассы:

- продолжительность большого цикла по времени (P): 160 мс - 2 с;
- продолжительность подциклов по времени |T|: 5 мс - 100 мс;
- количество подциклов в большом цикле P/|T|: 4 - 200;
- количество обменов в большом цикле: до 5000;
- количество обменов в подцикле (цепочке): до 100;

продолжительность одного обмена по времени: 25 мкс - 750 мкс.

4. Нечёткое сопоставление цепочек обменов

В случае, если количество подциклов двух циклограмм совпадает, и статистические характеристики близки, выполняется процедура нечёткого сопоставления цепочек обменов, образующих эти циклограммы.

Для этого нами была определена кодировка обменов M, мера близости между обменами V(M1,M2) и задача сравнения цепочек обменов была сведена к задаче нечеткого сравнения строк, для решения которой существует несколько известных алгоритмов [7]. Определим кодировку обменов 32-разрядными беззнаковыми целыми числами со следующим использованием бит:

- формат обмена (4 бит)
- адрес приемника (5 бит)
- адрес передатчика (5 бит)

- подадрес приемника (5 бит)
- подадрес передатчика (5 бит)
- количество слов данных/команда (5 бит)
- наличие ошибок (1 бит)

Указанные битовые поля не являются равнозначными при сравнении обменов и расположены в порядке убывания значимости. Так, изменение формата обмена или адресов, при сравнении обменов считается гораздо более существенным, чем изменение подадресов или количества слов данных. Для этого определим меру близости между обменами V(M1,M2) с учетом различных весов изменений обменов в цепочке и изменений их атрибутов сопоставленных обменов:

- вставка обмена в цепочку - 0,5
- удаление обмена из цепочки - 0,5
- изменение формата обмена - 1
- изменение адреса обмена - 1
- изменение подадреса обмена - 0,5
- изменение количества слов данных - 0,3
- изменение флага наличия ошибок - 0,1

В данном случае 0-ое значение меры близости V соответствует эквивалентности обменов, а близкое к 1 – существенным различиям в обменах.

Для определенных таким образом кодировки обменов и меры близости между обменами, задача нечеткого сравнения цепочек обменов сводится к задаче нахождения самой тяжелой общей подпоследовательности (heaviest common subsequence, hcs) для двух строк. hcs(x,y) - это последовательность, общая для строк x и y, имеющая при данной весовой функции максимальную сумму весов компонент. В общем случае, функция может зависеть как от самих символов, так и от их расположения в исходных строках.

Для решения задачи сравнения цепочек обменов определим весовую функцию hcs как W(i,j)=1-V(M1[i],M2[j]), где (M1[i],M2[j])-сопоставленная пара обменов из разных цепочек, найденная алгоритмом поиска hcs.

Для решения задачи нахождения hcs можно воспользоваться одним из известных алгоритмов: Вагнера-Фишера, Хиршберга, Накатсу, Укконена [7].

Так как цепочки редко имеют длину больше 50 обменов, и количество цепочек в большом цикле также невелико, различия в эффективности алгоритмов можно считать несущественными. Для реализации нами был выбран наиболее простой и наглядный алгоритм нечеткого сравнения - алгоритм Вагнера-Фишера, он имеет

квадратичную сложность по времени и по памяти от длины сопоставляемых строк [7].

Для сопоставления больших циклов двух трасс обменов в плохом случае требуется сравнить каждую цепочку из последовательности Sc1 с каждой цепочкой из Sc2 из-за возможного фазового сдвига по времени большого цикла. Таким образом, общая трудоемкость алгоритма - $O(N^2 \max_{i=1,N}(|Sc1[i]|)\max_{j=1,N}(|Sc2[j]|))$, где N – кол-во цепочек в циклограммах Sc1,Sc2, а максимумы определяют максимальную длину цепочки обменов в циклограммах Sc1 и Sc2.

Реализованные алгоритмы анализа были апробированы на трассах обменов по каналу МКИО бортовой вычислительной системы морского навигационного комплекса.

1. Государственный стандарт РФ «Интерфейс магистральный последовательный системы электронный модулей» ГОСТ Р 52070-2003.
2. Балашов В.В., Вавинов С.В., Гурьянов Е.С., Костенко В.А., Смелянский Р.Л. Система автоматического построения циклограммы обменов по шине с централизованным управлением // Методы и средства обработки информации. Труды второй Всероссийской научной конференции. - М.: Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 2005, с.516-521.V.V.
3. Костенко В.А., Гурьянов Е.С. Алгоритм построения расписаний обменов по шине с централизованным управлением и исследование его эффективности. // Программирование, 2005, No. 6, с. 340-346.
4. Электронная компания «Элкус», «ПО "ТЕСТЕР MIL-STD-1553B"» [HTML] http://www.elcus.ru/swtester1553.htm
5. Data Device Corporation (DDC), dataMARS/dataSIMS & Tester Simulator Menu software, [HTML] http://www.ddc-web.com/Products/MIL-STD-1553/Software.aspx
6. НТЦ «Модуль» Прикладное ПО для интерфейсных модулей МКО [HTML] http://www.module.ru/ruproducts/mil_std/soft
7. Graham A. Stephen. String Search // School of Electronic Engineering Science University College of North Wales TR-92-gas-01, 1992.

Чупилко М.М.

ИНТЕГРАЦИЯ ПОДХОДОВ К ТЕСТИРОВАНИЮ МОДЕЛЕЙ АППАРАТУРЫ НА ОСНОВЕ МЕТОДОЛОГИИ OVM

ИСП РАН, г.Москва, chupilko@ispras.ru

Многие методы и методики, разрабатываемые в академической среде, имеют серьезный теоретический фундамент, но не всегда применимы на промышленном уровне. Для индустрии нужны не методы, а технологии, поддержанные инструментальными средствами, обладающие достаточной универсальностью и интегрированные в имеющиеся процессы разработки и производства.

Одним из возможных путей для успешного выведения результата академических исследований на рынок является интеграция с широко используемой промышленной технологией.

В области тестирования моделей аппаратуры, на данный момент наиболее подходящей для интеграции является *открытая методология верификации* (*Open Verification Methodology*, *OVM*) [1], разработанная совместными усилиями фирм Cadence и Mentor Graphics. OVM занимает по подсчетам ее создателей до 60% рынка [2].

В данной работе рассмотрен пример интеграции OVM и подхода UniTESK к автоматизации тестирования моделей аппаратуры [3], разработанного и применяемого в ИСП РАН. Хотя в плане тестирования программного обеспечения UniTESK является уже устоявшейся, промышленной технологией, для верификации аппаратуры этот подход стал использоваться сравнительно недавно. Основным недостатком подхода, препятствующим его широкому применению, является отсутствие интеграции с общепринятыми процессами и инструментами разработки.

Рассмотрим основные возможности методологии OVM, которые делают ее промышленным подходом.

Все методики, определенные OVM, реализованы с использованием широко распространенных языков проектирования и верификации аппаратуры: SystemC, SystemVerilog, *e*. Это означает простую интеграцию с процессами разработки аппаратуры.

Особенностями подхода является использование *моделирования на уровне транзакций* (*Transaction-Level Modeling*, *TLM*), *генерации стимулов на основе ограничений* (*constraint-random generation*), а также *верификации, нацеленной на покрытие* (*Coverage-Driven Verification, CDV*). CDV наполняет практическим содержанием термин «цели тестирования», определяя их как создание всех интересных для

разработчика вариантов работы устройства, заданных в тестовом покрытии.

Использование объектно-ориентированных языков и TLM повышает уровень повторного использования кода тестовой системы, так как они оба стандартизуют компоненты тестовых систем – на уровне внутренней архитектуры и связей между ними.

OVM подразумевает четкое разделение тестовой системы на компоненты. Тестовая система состоит из 5 слоев: *реализации*, *транзакторов*, *операций*, *анализа*, *управления*, что представлено на рис. 1.

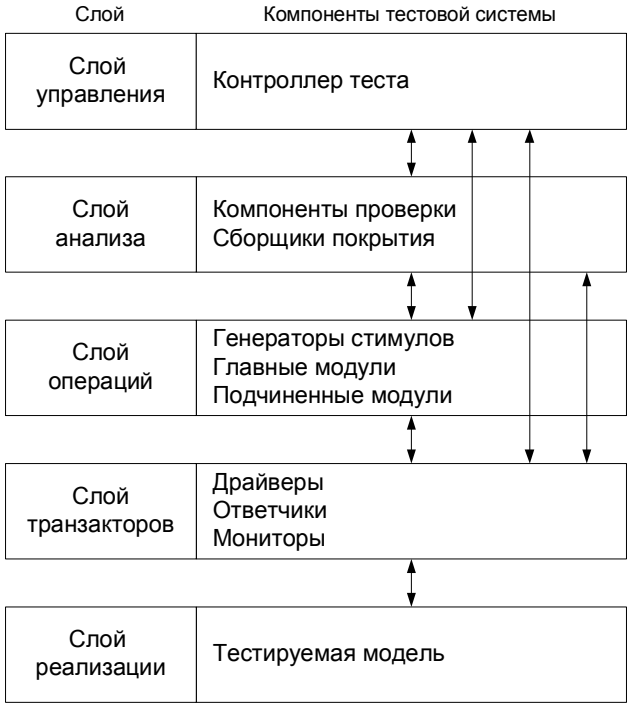


Рис.1. Структура тестовой системы согласно OVM

Слой транзакторов включает компоненты, которые имеют два интерфейса: используемый для связи с тестируемой моделью, и внутренний интерфейс тестовой системы – TLM. *Драйверы* (*drivers*), *ответчики* (*responders*), *мониторы* (*monitors*) объединены функцией преобразования данных между реализационным представлением и используемым внутри тестовой системы.

В *слой операций* включены компоненты, имитирующие окружение тестируемой модели. *Генератор стимулов* (*stimulus generator*) используется для создания потока тестовых воздействий, а *главные* (*master*) и *второстепенные* (*slave*) модули расширяют его возможности для поддержки операций с обратной связью.

Слой анализа объединяет компоненты, реализующие обработку информации, поступающей от нижележащих слоев. *Компоненты проверки* (*scoreboard*) сравнивают поведение тестируемой модели с эталонным, а *сборщики покрытия* (*coverage collector*) накапливают информацию о достигнутом уровне тестового покрытия для реализации CDV.

Слой управления включает в себя единственный компонент – *контроллер тестов*, который осуществляет запуск и останов тестирования в соответствии с информацией, поступающей от нижележащих слоев.

Следует заметить, что особую популярность OVM придала возможность построения сложных тестовых систем из блоков, называемых *OVC* (*Open Verification Component*), каждый из которых представляет собой тестовую систему для отдельного модуля тестируемой аппаратной системы. Совокупность OVC управляется единым *синхронизирующим контроллером* (*virtual sequencer*). Таким образом, тестовая система как бы составляется из «кирпичиков», которые могут поставляться вместе с модулями, разработанными сторонними компаниями.

Обобщая, OVM описывает общий подход к организации тестовых систем, описывает их архитектуру и предлагает способ построения тестовых систем из готовых блоков.

Рассмотрим интеграцию с OVM подхода UniTESK к тестированию моделей аппаратуры.

UniTESK также, как и OVM, четко определяет набор компонентов, их интерфейсы и функции, но еще использует *пред-* и *постусловия* выполнения операций или их отдельных стадий для автоматического создания *компонентов проверки* (*тестовых оракулов*), и *конечно-автоматные модели* для генерации последовательности тестовых воздействий. Архитектура тестовой системы согласно UniTESK представлена на рис. 2.

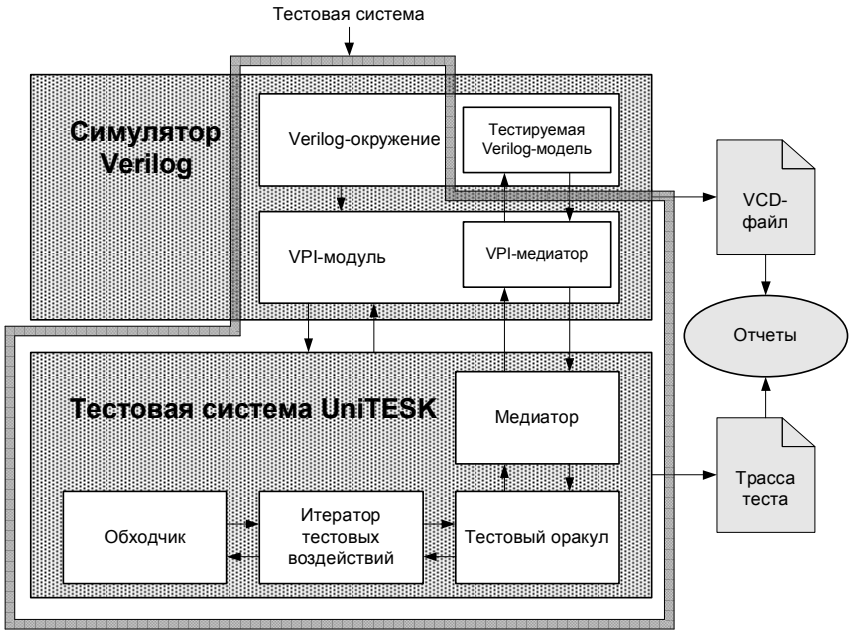


Рис.2. Архитектура тестовой системы UniTESK

Тестовая система состоит из следующих компонентов:

- *обходчик* – библиотечный компонент, предназначенный для генерации последовательностей тестовых воздействий на основе абстрактного представления тестируемой модели – графа состояний;
- *итератор тестовых воздействий*, предоставляющий обходчику набор допустимых тестовых воздействий в каждом достижимом состоянии;
- *тестовый оракул*, проверяющий корректность поведения тестируемой модели, ее соответствие эталонной модели, автоматически генерируемый на основе спецификации;
- *медиатор*, переводящий тестовые воздействия из внутреннего представления тестовой системы в формат тестируемой модели, также осуществляющий преобразование в обратном направлении;
- дополнительные компоненты для взаимодействия с Verilog-симулятором (Verilog-окружение, VPI-модуль, VPI-медиатор).

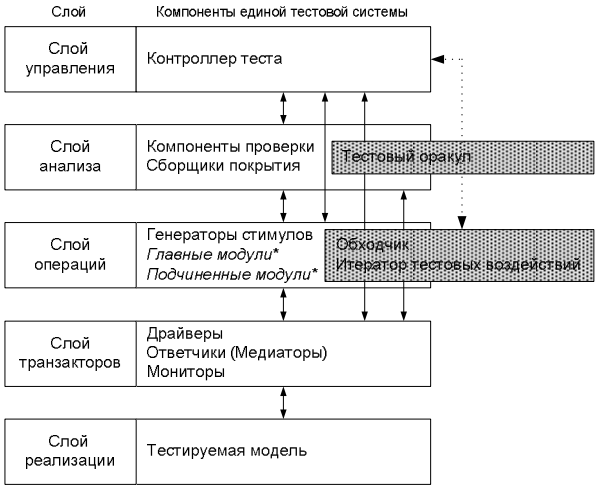
Необходимо отметить, что для тестирования моделей аппаратуры используется реализация UniTESK для языка C – CTESK [3], что препятствует использованию объектно-ориентированного программирования и интеграции с существующими процессами разработки аппаратуры.

Схема возможного способа интеграции подхода UniTESK в OVM представлена на рис. 3.

Согласно рисунку, необходимо включить совокупность обходчика и итератора тестовых воздействий UniTESK в слой операций, с подчинением их контроллеру теста, обеспечив связь между ними посредством интерфейса TLM. Спецификация тестируемой модели в виде пред- и постусловий следует поместить в тот же слой. Их можно использовать для создания объектов главных и подчиненных модулей, например, для контроля предусловий операций.

Тестовый оракул, который в целом повторяет функции компонентов проверки, в рамках UniTESK генерируется автоматически из спецификаций. Поэтому его добавление к слою анализа не носит принципиального характера, но позволяет автоматизировать создание компонентов проверки.

Медиаторы UniTESK являются полным аналогом транзакторов OVM.



* - модифицированы применением пред- и постусловий

Рис.3. Интегрированный подход

Поскольку OVM является распространенным подходом, такая интеграция позволяет создавать высококачественные тесты, используя

теоретически обоснованный подход UniTESK, для инженеров, занимающихся промышленным проектированием и верификацией аппаратуры. Тесты UniTESK приобретают новое качество — их можно отчуждать, повторно использовать при построении более крупных тестовых систем и т.д.

В настоящее время в ИСП РАН идут работы по реализации концепции UniTESK для языка SystemC. К настоящему моменту разработана библиотека спецификации аппаратуры, поддерживающая описание сложной аппаратуры с конвейерной организацией и возможностью параллельного выполнения операций, готовая для интеграции в OVM.

В работе приведен пример интеграции академического подхода в коммерчески успешную технологию. Этот пример можно обобщить и на другие научные разработки, что позволило бы результатам академического сообщества сделать свои результаты доступными для широкого круга пользователей.

1. *OVM User Guide*. Mentor Graphics, Cadence, 2008 (www.ovmworld.org)
2. *OVM Is The Safest Bet By 2:1*, (http://www.cadence.com/Community/blogs/fv/archive/2009/02/18/ovm-is-the-safest-bet-by-2-1.aspx?postID=14789)
3. В.П. Иванников, А.С. Камкин, В.В. Кулямин, А.К. Петренко. *Применение технологии UniTESK для функционального тестирования моделей аппаратного обеспечения*. Препринт ИСП РАН, 2005