конструирования ТП. Первая модель – это детерминированная модель, которая позволяет задачам ТП выполняться или один раз, или вообще не выполняться в течение любой реализации процесса. Допустимыми решениями этой задачи являются «успешные» допустимые *w*-е реализации ТП.

Вторая из рассмотренных моделей соответствует вероятностному выполнению задач. В этом случае допустимые решения для соответствующей задачи оптимизации – это допустимые случайные акции π. И, наконец, последняя модель, рассмотренная здесь, имеет дело с последовательностями случайных акций, что соответствует многочисленным исполнениям ТП, вплоть до успешного завершения процесса. Результат решения задачи – направления ψ, которые соответствуют последовательности случайных акций и могут быть получены за конечное число шагов итеративного алгоритма.

Библиографические ссылки

1. Капулин Д. В., Ковалев И. В., Царев Р. Ю. Архитектурная надежность программного обеспечения информационно-управляющих систем : монография / Краснояр. гос. аграр. ун-т. Красноярск, 2011.

2. Антамошкин А. Н., Ковалев И. В., Царев Р. Ю. Математическое и программное обеспечение отказоустойчивых систем управления и обработки информации : монография / Краснояр. гос. аграр. ун-т. Красноярск, 2011.

3. Ковалев И. В., Зеленков П. В., Брезицкая В. В. Инструментальные средства формирования мультиверсионной архитектуры отказоустойчивых программных систем : монография / Краснояр. гос. аграр. ун-т. Красноярск, 2011.

4. Antamoshkin A. N., Saraev V. N., Semenkin E. S. Optimization of unimodal monotone pseudoboolean functions // Kibernetika. 1990. Vol. 26, № 5. P. 432–442.

E. A. Antamoshkina

TECHNOLOGICAL PROCESSES REALIZATION MODELING

The author suggests analytical optimization procedure for modeling of technical processes realization on computer nets of automation control systems.

Keywords: technological process, computer net, modeling, oriented graph.

УДК 519.688

К. В. Богданов, А. Н. Ловчиков

МОДЕЛИРОВАНИЕ СИСТЕМ С СУЩЕСТВЕННЫМИ НЕЛИНЕЙНОСТЯМИ С ПОМОЩЬЮ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

Предлагается новый подход к созданию EDA-систем, предназначенных для моделирования схем, содержащих элементы, функционирование которых связано с резким изменением параметров и при математическом описании приводит к существенному возрастанию производных фазовых переменных, что при традиционном моделировании ведет к срыву вычислительного процесса.

Ключевые слова: электронные устройства, вычислительные алгоритмы, САПР, EDA-системы.

Современное программное обеспечение, применяющееся для моделирования различных процессов, начиная от нагрузочных испытаний и заканчивая переходными процессами в электронных схемах, основывается на архитектуре, заложенной еще на заре развития вычислительной техники, в 60–70-х гг. прошлого века.

Развитие данного класса программного обеспечения идет, как правило, по пути улучшения пользовательских интерфейсов, расширения базы данных электронных компонентов и т. п., в то время как основные вычислительные алгоритмы остаются прежними. Чаще всего все сводится к решению получившейся системы линеаризованных дифференциальных уравнений, являющейся математической моделью устройства. Численные методы решения при этом позволяют получать весьма точные результаты, но проблемы возникают при моделировании систем с большим количеством компонентов, что приводит к усложнению математической модели, и при моделировании систем, в которых имеются существенные нелинейности, значительно огрубляющие результаты моделирования либо приводящие к тому, что вычислительный процесс расходится [1; 2].

В работах [3] и [4] авторами был предложен иной подход к решению этой задачи: рассмотрение моделируемой схемы как совокупности конкурирующих независимых параллельных процессов.

Основные проблемы, связанные с такой схемой работы, заключаются в том, что необходимо обеспечить высокую степень взаимной изолированности вычислительных процессов, сохраняя возможность синхронизированного обмена данными. Можно провести аналогию с современными вычислительными сетями, где каждая вычислительная машина максимально «самостоятельна», но имеет возможность обмениваться данными, разбитыми на пакеты, с любой другой машиной в сети в произвольный момент времени. Однако существенным отличием здесь будет являться то, что каждый процесс должен выдавать и принимать порцию информации строго по синхронизирующему сигналу. В том случае, если процесс не успевает по каким-либо причинам это сделать, возможны два подхода: ожидание и уничтожение. В первом случае ни одна порция данных от других процессов не будет принята и не будет передана, пока от всех процессов модели не будет получен ответ. Во втором случае процессы, данные от которых не получены по истечении установленного времени, будут уничтожены либо перезапущены, а недостающие значения данных заменены на нулевые либо заранее принятые. Возможна и гибридная стратегия, когда процесс уничтожается после некоторого ожидания.

Реализация асинхронного обмена сообщениями, как это сделано, например, в вычислительных сетях на основе технологии Ethernet, может повлечь за собой серьезную проблему: результаты моделирования будут зависеть от производительности системы и без предварительного профилирования реализовать модель не получится.

Как при асинхронном, так и при синхронном обмене необходим отдельный процесс-маршрутизатор. В его функции входит сбор данных от остальных процессов, уничтожение процессов, не выславших данные в течение отведенного времени, рассылка данных по процессам в соответствии с таблицей взаимосвязей (рис. 1).

Реализовать такую систему возможно с помощью различных средств разработки ПО, но в данном случае был выбран Erlang (эрла́нг) – функциональный язык программирования, позволяющий разрабатывать программное обеспечение для разного рода распределенных систем. Язык разработан и поддерживается компанией Ericsson, включает в себя средства порождения параллельных процессов и их коммуникации с помощью посылки асинхронных сообщений. Программа транслируется в байт-код, исполняемый виртуальной машиной, что обеспечивает возможность ее выполнения в различных операционных системах. Функциональная парадигма позволяет Erlang избежать таких традиционных для императивных языков проблем распределенных приложений, как необходимость синхронизации, опасность возникновения тупиков и гонок [5].

Запущенный экземпляр эмулятора Erlang называется узлом. Узел имеет имя и располагает информацией о существовании других узлов на данной машине или в сети. Создание и взаимодействие процессов разных узлов не отличается от взаимодействия процессов внутри узла. Для создания дочернего процесса необходимо знать лишь его идентификатор (имя). При этом нет необходимости в указании конкретного физического узла, на котором этот процесс будет выполняться. Этим обусловливается высокая масштабируемость и способность почти линейного повышения производительности с ростом мощности системы (кластера) [6].

Количество рассылаемых сообщений будет зависеть исключительно от топологии моделируемой системы.

Очевидно, что для функционирования системы необходимо иметь несколько различных типов элементов.

1. Обычный процесс (передаточная функция, пример реализации приведен выше). Обеспечивает преобразование входного потока данных в выходной. В каждый момент синхронизации обязан принять и передать одну порцию (кортеж) данных.

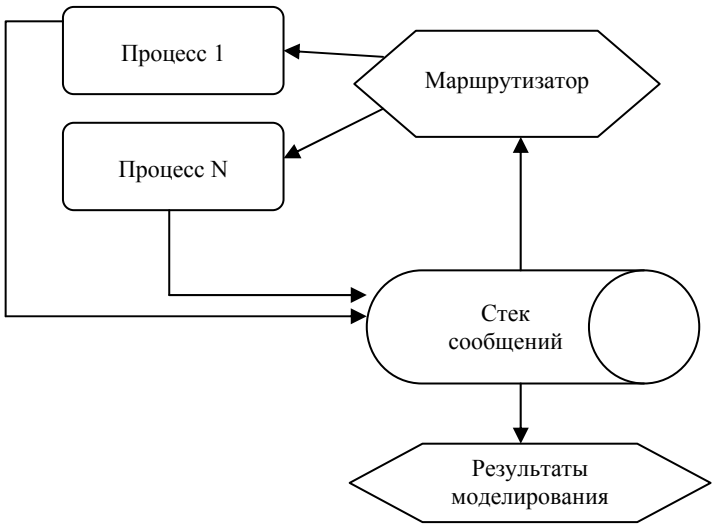

Рис. 1. Общая схема системы моделирования с параллельными процессами

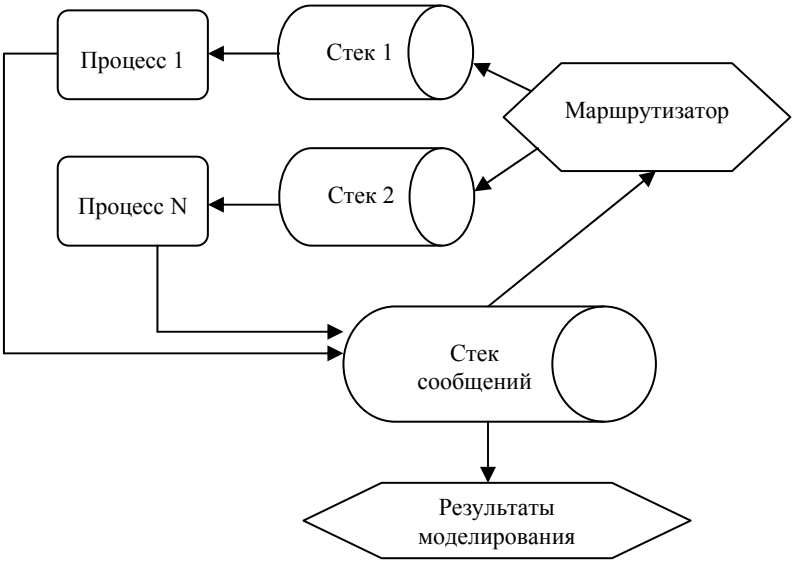

Рис. 2. Общая схема системы с индивидуальными стеками процессов

2. Маршрутизатор. Обеспечивает перераспределение сигналов по нескольким выходным каналам в зависимости от соотношения количества входов и выходов. Маршрутизатор по сути является совокупностью всех узлов цепи. Простейший пример симметричного (распределяющего входные сигналы равномерно) маршрутизатора размерности 2×3 будет выглядеть следующим образом:

```
commutator(Server_Node) ->
        receive
        stop ->
    exit(normal);
    {pin1, signal1_type, signal1},{pin2, signal2_type, signal2} ->
    signal1_type -> signal1_type, signal2_type, signal1_type
    (signal1+signal2)/3-> signal1, signal2, signal3
  {modeling, Server_Node}!{self(), output1, 1, signal1_type, signal1},{self(), output2, 2,
   signal2_type, signal2},{self(), output3, 3, signal3_type, signal3 }
end.
```

3. Источник. Обеспечивает выдачу сигналов. Входов не имеет. Простейший пример источника с одним выводом:

```
source(Server_Node) ->
        receive
        stop ->
          exit(normal);
          { } ->
{modeling, Server_Node}!{self(), output1, 1, type1, 100}
end.
```

4. Выход. Псевдоблок, необходимый для получения текущих значений параметров для анализа. Является адаптером между моделью и пользовательским интерфейсом. Имеет один вход, выходов нет.

Одной из важнейших процедур при создании подобной модели является построение таблицы маршрутизации по пользовательской модели (к примеру, по принципиальной электрической схеме). На основе этой информации также должны создаваться процессы-коммутаторы.

Однако при реализации такой системы выяснилось, что оказалось невозможным построение, например, переходных процессов. Происходит это из-за того, что обработка данных в подобной системе происходит без учета предыдущих значений, а, следовательно, интегрирование или дифференцирование величин, представляющих собой какой-либо рассматриваемый полезный сигнал, в отдельных узлах системы произвести не получится.

Для решения этой проблемы предлагается для каждого из блоков создать собственный стек, хранящий текущее и предыдущие значения сигналов. В этом случае общая схема системы будет выглядеть так, как представлено на рис. 2.

Важным вопросом здесь также является выбор величины (глубины) стека. С одной стороны, слишком малое количество хранящихся величин не позволит корректно проводить операции дифференцирования и интегрирования. С другой стороны, при слишком глубоких стеках значительно увеличится время между получением данных блоком и началом их обработки. Это может привести к рассинхронизации блоков между собой, и, как следствие, неверным результатам моделирования. Также дополнительными осложняющими факторами является то, что, во-первых, глубина всех стеков в рамках одной модели должна быть одинакова, и, во-вторых, эта величина должна быть известна перед началом процесса моделирования.

Авторами предлагается следующая формула расчета глубины стека:

$$N = \frac{\gamma_{\text{дискр}}}{\gamma_{\text{ист}}} 4,$$

где $\gamma_{\text{дискр}}$ – частота дискретизации системы, определяемая из длительности периода наблюдения и необходимой точности (количества точек); $\gamma_{\text{ист}}$ – макси-

мальная из частот источников, присутствующих в моделируемой системе.

Добавится также блок стека и в список базовых блоков в системе. В простейшем случае его описание на языке Erlang будет выглядеть так (глубина стека здесь равна 10):

```
stack(Server_Node) ->
        receive
        stop ->
              exit(normal);
    {pin1, signal1_type, signal1} ->
      signal1_type -> signal1_type, signal2_type,
signal1_type
        delay(signal1, 10)-> signal
{modeling, Server_Node}!{self(), output1, 1,
signal1_type, signal1}end.
```

С помощью этих улучшений планируется реализовать систему моделирования переходных процессов для силовых электронных схем, работающих, в частности, с большим количеством электронных ключей (повышающие импульсные источники питания с ШИМ) и высокой мощностью.

Библиографические ссылки

1. Хайнеман Р. PSpice. Моделирование работы электронных схем : пер. с нем. М. : DMK-Пресс, 2001.

2. Норенков И. П. Основы автоматизированного проектирования. М. : Изд-во МГТУ им. Н. Э. Баумана, 2002.

3. Богданов К. В., Ловчиков А. Н. Архитектура EDA-системы на основе конкурирующих параллельных процессов // Изв. вузов. Приборостроение. 2011. Т. 54, № 4. С. 63–67.

4. Богданов К. В., Ловчиков А. Н. Построение EDA-системы на основе синхронизированных параллельных процессов // Вестник СибГАУ. 2009. Вып. 4 (25). С. 58–61.

5. Open Source Erlang [Электронный ресурс]. 2008. URL: http://www.erlang.org/. (date of visit: 12.11.2012).

6. Armstrong J. Programming Erlang: Software for a Concurrent World. Pragmatic Bookshelf, 2007.

K. V. Bogdanov, A. N. Lovchikov

SIMULATION OF SYSTEMS WITH ESSENTIAL NONLINEARITY WITH THE HELP OF PARALLEL PROGRAMMING

The authors suggest a new approach to EDA-systems designed to simulate circuits containing elements, functioning of which is related with abrupt change of parameters and in the process of mathematical description leads to substantial increase of derivatives of state variables, that in the process of traditional modeling leads to breakdown of the computational process.

Keywords: electronic devices, computing algorithms, CAD-systems, EDA-systems.

УДК 621.45.017

А. Е. Буров

ОЦЕНКА ЖИВУЧЕСТИ КОНСТРУКЦИИ КРЕПЛЕНИЯ КРЫШКИ ГИДРОТУРБИНЫ В АВАРИЙНОЙ СИТУАЦИИ*

Рассмотрены подходы к количественной оценке живучести конструкций технических систем. Представлены результаты анализа живучести конструкции разъемного соединения крышки турбины гидроагрегата в условиях аварийной ситуации при прогрессирующем разрушении несущих элементов. На основе моделирования напряженно-деформированного состояния определены показатели живучести в зависимости от числа отказавших элементов.

Ключевые слова: разъемное соединение, напряженно-деформированное состояние, разрушение, живучесть.

Традиционные методы проектирования конструкций предполагают исключение возникновения предельных состояний несущих элементов при проектных нагрузках. Однако отказы, разрушения и аварийные ситуации неизбежны практически для любой технической системы. Стало быть, существующий подход не отвечает современным требованиям обеспечения безопасности, которые предполагают изучение работоспособности конструкций с учетом запроектных нагрузок и воздействий. Неотъемлемой частью такого анализа является определение параметров живучести несущих конструкций.

*Работа выполнена при финансовой поддержке РФФИ (код проекта 11-08-00945).