МАТЕМАТИЧЕСКИЕ МЕТОДЫ ОЦЕНКИ РЕЗУЛЬТАТОВ ОТРАБОТКИ ЭЛЕМЕНТОВ КОСМИЧЕСКИХ АППАРАТОВ

УДК 519.688

К. В. Богданов, А. Н. Ловчиков

АРХИТЕКТУРА EDA-СИСТЕМЫ НА ОСНОВЕ КОНКУРИРУЮЩИХ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ

Предлагается архитектура EDA-системы, в которой отсутствует единая математическая модель анализируемого устройства и вместо этой модели реализуется совокупность элементарных моделей, при этом для каждой из них выделен отдельный вычислительный процесс. Вычислительные процессы выполняются в распределенной виртуальной вычислительной машине, и обмен асинхронными сообщениями между процессами осуществляется по маршрутам, определяемым топологией исходной системы. Данный подход позволяет избежать влияния нелинейностей на точность моделирования.

Ключевые слова: *системы автоматизированного проектирования, EDA-системы, архитектура программного обеспечения.*

Моделирование и анализ работы электронного оборудования — весьма сложная задача, для решения которой активно используется специализированное программное обеспечение — EDA-системы. Развитие таких систем осуществляется в основном экстенсивным путем. Как правило, улучшаются пользовательские интерфейсы, расширяются базы данных электронных компонентов и т.п., в то время как основные вычислительные алгоритмы остаются прежними. Существующие методы анализа, используемые в EDA-системах, сводятся к решению результирующей системы линеаризованных дифференциальных уравнений, являющейся математической моделью моделируемого устройства. Численные методы решения при этом позволяют получать весьма точные результаты, но проблемы возникают при моделировании систем с большим количеством электронных компонентов, что приводит к усложнению математической модели, а также при моделировании систем, имеющих существенные нелинейности, которые значительно огрубляют результаты моделирования либо вообще приводят к расхождению вычислительного процесса [1—3].

В связи с этим предлагается диаметрально противоположный подход: каждый блок либо компонент устройства должен быть смоделирован отдельно. Реализация полученных моделей должна осуществляться с использованием отдельных вычислительных процессов. При этом если моделирование выполняется в некотором диапазоне времени, то на один или несколько логических входов каждого из таких вычислительных процессов должны поступать некоторые параметры (например, мгновенное значение напряжения относительно общей шины питания). Эти процессы также должны выдавать в качестве выходных данных результаты обработки входных параметров.

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 каждый компьютер максимально „самостоятелен“, но имеет возможность обмена данными, разбитыми на пакеты, с любым другим компьютером сети в произвольный момент времени. Существенным отличием предлагаемого подхода является то, что при каждом процессе прием и передача информации должны осуществляться строго по синхронизирующему сигналу. В случае если это требование не выполняется по каким-либо причинам, возможны два подхода: ожидание либо уничтожение результатов процесса. В первом случае ни один набор данных, поступивших при выполнении других процессов, не будет принят и не будет передан, пока не будут получены результаты всех процессов модели. Во втором случае процессы, данные которых не получены по истечении установленного времени, будут уничтожены либо перезапущены, а недостающие значения будут заменены на нулевые либо заранее заданные. Возможна и гибридная стратегия, когда результаты процесса уничтожаются после некоторого ожидания.

Реализация асинхронного обмена сообщениями (как, 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 данными необходим отдельный процесс-маршрутизатор. В его функции входит сбор данных, полученных при выполнении остальных процессов, уничтожение результатов процессов, данные которых не были получены в течение отведенного интервала времени, рассылка данных по процессам в соответствии с таблицей взаимосвязей. Наглядно потоки данных представлены на DFD-диаграмме в нотации Гейна — Сарсона [4] на рис. 1.

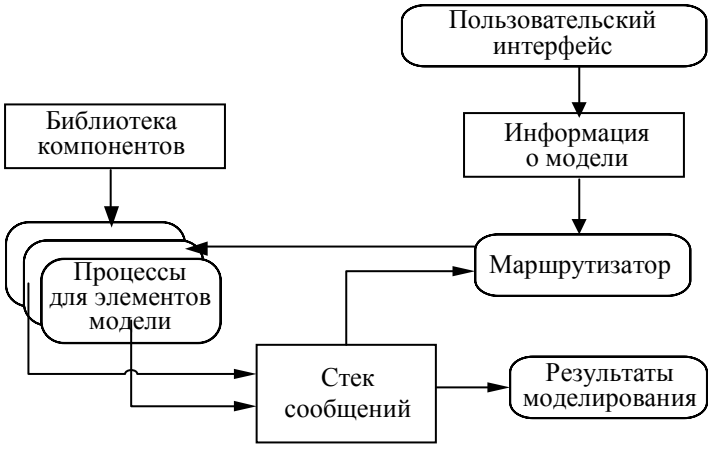

Рис. 1

Техническая реализация в этом случае требует наличия среды, обеспечивающей одновременное (параллельное) выполнение большого количества несложных вычислительных процессов, которые реализуют атомарные блоки системы, и поддерживающей обмен сообщениями между этими несложными процессами.

Наиболее подходящей для реализации средой представляется Erlang (Эрла́нг) — функциональный язык программирования (разработан и поддерживается компанией “Ericsson”), позволяющий разрабатывать программное обеспечение для разного рода распределенных систем. Язык Erlang включает в себя средства, определяющие порождение параллельных процессов и их коммуникацию с помощью посылки асинхронных сообщений. Программа транслируется в байт-код, исполняемый виртуальной машиной, что обеспечивает возможность ее выполнения в различных операционных системах. Функциональная парадигма языка Erlang позволяет 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 уникальный идентификатор и содержит информацию о существовании других узлов на данной машине или в сети. Создание и взаимодействие процессов в разных узлах не отличается от взаимо-

действия 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 системы (кластера) с ростом ее мощности [6].

Между процессами, моделирующими каждый отдельный элемент схемы, может быть осуществлен обмен сообщениями (кортежами) вида:

```
{<имя корневого процесса>, <номер элемента>, <номер вывода>,
<тип сигнала1>, <величина сигнала 1>…, <тип сигнала n>,
<величина сигнала n>}.
```

Простейший процесс, моделирующий элемент схемы, на языке Erlang описывается следующим образом (Server_Node — заранее определенное имя сервера, modeling — имя исполняемой программы):

```
element(Server_Node) ->
 receive
stop ->
 exit(normal);
      {pin_number, signal1_type, signal1} ->
% обработка входных значений %,
{modeling,  Server_Node}!{self(),  element_number,  pin_number,  sig-
nal1_type, signal1 }
end.
```

Шаблон маршрутизатора описывается следующим образом (Element_List — список всех элементов):

```
server(Element_List)   ->
 receive
      {element_number, pin_number, signal1_type, signal1} ->
            % описание таблицы маршрутизации %
            element_number ! {pin_number, signal1_type, signal1 }
end.
```

Количество рассылаемых сообщений будет зависеть исключительно от топологии моделируемой системы, в примере показан простой вариант без множественных соединений типа „один выход — много входов“.

Для функционирования системы необходимо иметь несколько различных типов элементов (процессов).

1) Обычный процесс (передаточная функция, пример реализации приведен выше) — обеспечивает преобразование входного потока данных в выходной; в каждый момент синхронизации осуществляется прием и передача одного набора (кортежа) данных.

2) Маршрутизатор — обеспечивает перераспределение сигналов по нескольким выходным каналам в зависимости от соотношения количества входов и выходов (маршрутизатор, по сути, является совокупностью всех узлов); простейший пример симметричного (распределяющего входные сигналы равномерно) маршрутизатора размерности „2 на 3“ имеет следующий вид:

```
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 выходом:

```
commutator(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 процесса, является адаптером между моделью и пользовательским интерфейсом; имеет один вход, выходов нет.

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

После запуска системы все ее элементы осуществляют передачу сообщений маршрутизирующему процессу — серверу, который выполняет их распределение в соответствии с заданной топологией моделируемой системы.

В целом, программный продукт, построенный на основе данной схемы, состоит из следующих компонентов:

— ядро системы — виртуальная машина Erlang, основные функции — моделирование;

— библиотека компонентов — хранилище процедур на декларативном языке Erlang;

— схема построения модели — приложение на императивном языке высокого уровня (например, C#), формирующее в автоматическом режиме матрицу взаимосвязей для маршрутизатора и исходный Erlang-код всех процессов модели;

— графический интерфейс — приложение на императивном языке высокого уровня, обеспечивающее ввод и вывод информации для пользователя.

Упрощенная UML-диаграмма работы (activity) приложения [4], построенного на основе данной архитектуры, приведена на рис. 2.

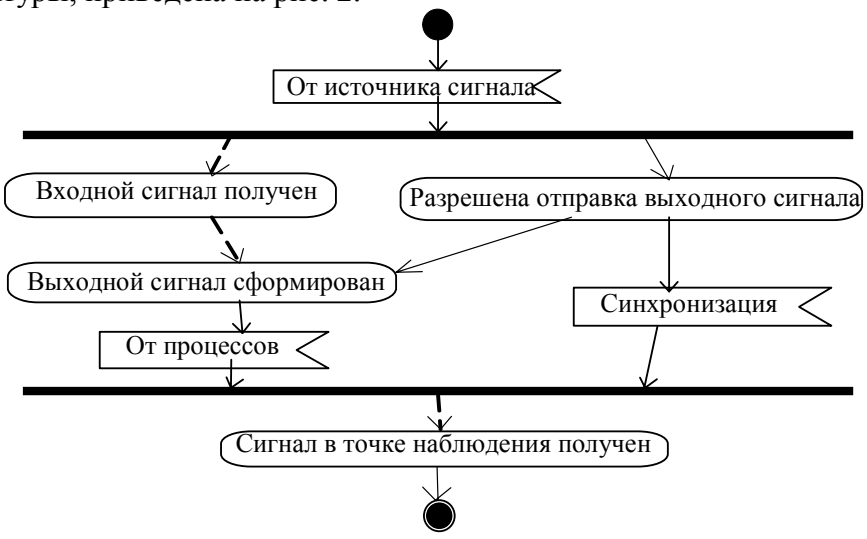

Рис. 2

Реализация системы, основанной на описанной архитектуре, позволяет повысить точность и скорость моделирования систем с существенными нелинейностями, а также проводить моделирование в течение длительных интервалов времени без риска потерь данных.

СПИСОК ЛИТЕРАТУРЫ

1. *Хайнеман Р.* PSpice. Моделирование работы электронных схем: Пер. с нем. М.: DMK Пресс, 2001. 336 с.

2. *Норенков И. П.* Основы автоматизированного проектирования. М.: Изд-во МГТУ им. Н. Э. Баумана, 2002. 336 с.

3. Численные методы, параллельные вычисления и информационные технологии: Сб. науч. трудов / Под ред. *Вл. В. Воеводина и Е. Е. Тыртышникова*. М.: Изд-во МГУ им. М. В. Ломоносова, 2008. 320 с.

4. *Киммел П.* UML. Основы визуального анализа и проектирования: Пер. с англ. М.: НТ Пресс, 2008. 272 с.

5. Open Source Erlang [Электронный ресурс, англ.]: <http://www.erlang.org/>.

6. *Armstrong J.* Programming Erlang: Software for a Concurrent World. Lewisville, USA: Pragmatic Bookshelf (The Pragmatic Programmers, LLC), 2007. 440 с.

Сведения об авторах

Константин Валериевич Богданов — канд. техн. наук, доцент; Сибирский государственный аэрокосмический университет им. акад. М. Ф. Решетнёва, кафедра информатики и вычислительной техники, Красноярск; E-mail: darkstone@rambler.ru

Анатолий Николаевич Ловчиков — д-р техн. наук, профессор; Сибирский государственный аэрокосмический университет им. акад. М. Ф. Решетнёва, кафедра информатики и вычислительной техники, Красноярск; E-mail: lanlov8@mail.ru

Рекомендована СибГАУ

Поступила в редакцию
19.11.10 г.