УДК 519.6

МЕТОД АВТОМАТИЧЕСКОГО ПОСТРОЕНИЯ МОДЕЛИ ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ В ТЕРМИНАХ СЕТЕЙ ПЕТРИ

Е. А. Голенков[1], **А. С.Соколов**[1]

В статье рассматривается метод автоматического построения моделей для параллельных программ, выполняющихся в среде передачи сообщений. Предлагаемый метод позволяет строить модели параллельных программ, поток управления которых представляется в терминах сетей Петри, а описание данных и их модификация остаются в терминах исходного языка программирования. Предлагаемый метод разработан в рамках проекта создания экспериментальной среды разработки параллельных программ, использующей аппарат теории сетей Петри, и реализован в виде одной из ее подсистем. Работа выполнена при финансовой поддержке Президиума РАН (программа № 17) и РФФИ (проект № 04–07–90287).

Ключевые слова: параллельное программирование, сети Петри, построение модели, трансляция.

1. Введение. При проверке корректности параллельных программ помимо тестирования и отладки используют различные техники верификации, среди которых одной из основных является техника модельного подхода. Эта техника верификации основывается на анализе свойств моделей программ, выраженных в терминах различных математических теорий. Такие теории предоставляют точные способы и методы для верификации, позволяющие формально проанализировать интересующие свойства параллельных программ (например, выявлять ошибки, возникающие при организации взаимодействия процессов). В рамках данной техники верификации для создания моделей используют представления из теорий автоматов, графов и сетей Петри, а также потоковые диаграммы и машины Тьюринга.

На сегодняшний день существует целая линейка программных систем, предоставляющая возможности для графического моделирования и анализа распределенных систем [1]. В таких системах программист берет на себя непростую задачу по отображению параллельной программы в терминах, обеспечивающих высокий уровень абстракции для спецификации параллелизма. Однако хотя построение моделей параллельных программ является трудной задачей, в данной области ведутся работы, направленные на автоматизацию процесса построения модели. Так, можно выделить группу средств автоматического построения моделей программ по их спецификациям [3, 5]. В то же время существуют работы по построению и анализу моделей для “готовых” параллельных программ или их частей [2].

В области построения и анализа моделей готовых параллельных программ одним из эффективных и наглядных математических формализмов являются сети Петри. Здесь можно упомянуть работы, выполненные под руководством В. А. Соколова по построению моделей параллельных программ, запрограммированных на языке Си, а также работы, ведущиеся под руководством В. А. Непомнящего по построению моделей параллельных программ, запрограммированных на языке Паскаль. Из зарубежных работ можно упомянуть методы трансляции программ на языке Ада [4, 6].

Тем не менее следует отметить, что несмотря на перспективность построения и анализа моделей готовых параллельных программ, заметен разрыв между теоретической базой и существующими автоматическими средствами построения и анализа моделей.

В данной статье для преодоления такого разрыва авторами предложен метод автоматизации процесса построения моделей параллельных программ, работающих в среде передачи сообщений. Описание метода приводится по следующей схеме: в разделе 2 формулируется метод и дается краткое описание этапов построения модели параллельной программы, причем конечная модель параллельной программы представляется композицией моделей исполнения отдельных последовательных процессов программы и модели межпроцессного взаимодействия; в разделе 3 кратко обсуждается содержательная характеристика сетей Петри для программирования [7], в терминах которых выражается конечная модель; в разделе 4 приводится описание модели исполнения отдельных последовательных процессов программы в терминах сетей Петри для программирования; в разделе 5 рассматривается пример построения модели среды взаимодействия.

[1] Институт автоматики и процессов управления ДВО РАН, 690041, ул. Радио, д. 5, г. Владивосток; e-mail: also@dvo.ru

Представленный в статье метод базируется на результатах, полученных ранее при выполнении работ по теме “Системы и методы исследования, оптимизации и автоматизации распараллеливания вычислений и обработки данных” в рамках Федеральной целевой научно-технической программы “Исследования и разработки по приоритетным направлениям развития науки и техники”.

2. Формулировка метода. Для спецификации параллельных программ, запрограммированных императивными языками программирования, достаточно описать поток управления и поток данных. Сети Петри считаются точным и наглядным формализмом для представления потока управления в параллельной программе. Однако при построении моделей в терминах сетей Петри возникают трудности при отображении стандартных для большинства языков программирования данных и при описании их модификации.

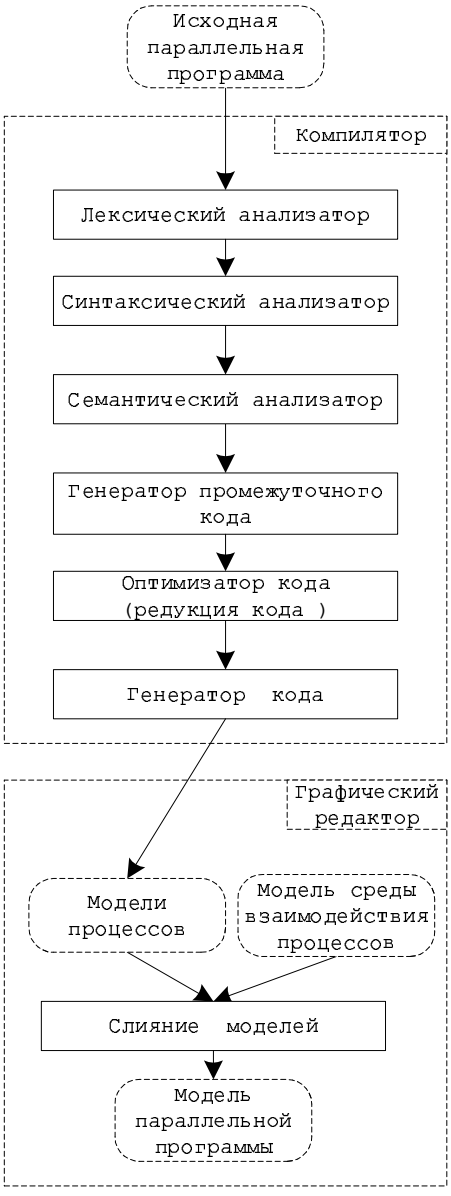

Рис. 1. Фазы построения модели параллельной программы

Подходы к работе с данными в моделях, представляемых сетями Петри, можно классифицировать следующим образом. Во-первых, данные и их модификацию можно оставлять за пределами сетей Петри и отображать только поток управления параллельной программы. Такой подход реализуют ряд графических систем моделирования и анализа распределенных систем [1]. Во-вторых, можно использовать специализированный язык описания данных (например, язык SML в среде CPN-Design [9]). Применение этого подхода требует изучения вопроса об идентичности использования данных и операций над ними в принципиально различных языках. В-третьих, данные и их модификацию можно оставлять в терминах исходного языка программирования. Такой подход требует адаптации сетей Петри для описания структуры параллельных программ с использованием типов данных с большим количеством значений, описанных в терминах императивных языков программирования.

Предлагаемый метод разработан в рамках третьего подхода и основывается на представлении модели в терминах сетей Петри для программирования. В соответствии с выбранным подходом метод построения модели параллельной программы состоит из следующей последовательности действий: поток управления параллельной программой преобразуется в структуру сетей Петри; процессы параллельной программы вместе с данными преобразуются в маркеры сетей Петри; поток управления полученной модели контролируется данными, описание операций над которыми остается в терминах исходного языка программирования.

Автоматическое построение модели параллельной программы предлагается выполнять по схеме, представленной на рис. 1. На нем показано, что конечная модель параллельной программы представляется композицией моделей исполнения отдельных последовательных процессов программы и модели межпроцессного взаимодействия. Такое разбиение исходной программы на части обусловлено тем, что рассматриваемый класс параллельных программ представляет собой набор последовательных взаимодействующих между собой процессов, в котором процессы определяют общий алгоритм вычислений, а функции организации взаимодействия реализуют только передачу данных между процессами и не могут быть интегрированы в алгоритм решаемой задачи.

Построение моделей отдельных последовательных процессов программы укладывается в схему работы классического компилятора. Здесь можно выделить фазы разбора текстов параллельной программы с целью выявления отдельных последовательных процессов и генерации некоторого промежуточного представления. Кроме того, при построении моделей отдельных процессов возможно выполнение фазы частичной редукции исходной программы в зависимости от целей модели (критерий редукции задается как входной параметр компилятора). Так, например, в случае анализа модели на выявление ошибок межпроцессного

взаимодействия функции, не вызывающие прямо или косвенно функции из библиотеки среды организации взаимодействия, можно рассматривать как неделимое событие. На этапе генерации кода производится сопоставление семантическим конструкциям отдельных процессов параллельной программы соответствующих шаблонных конструкций в терминах сетей Петри для программирования.

Рассматриваемый метод предполагает наличие графического редактора сетей Петри для отображения получившихся моделей. Кроме того, предполагается, что редактор должен предоставлять возможности неавтоматического построения и изменения моделей в терминах сетей Петри.

Построение модели межпроцессного взаимодействия осуществляется средствами графического редактора на основе определенных заранее целей модели. Отметим, что хотя создание модели межпроцессного взаимодействия не происходит автоматически, тем не менее данный этап может быть сведен к выбору модели из ограниченного списка заранее подготовленных моделей среды взаимодействия.

3. Сети Петри для программирования. Сети Петри для программирования представляют собой расширение цветных сетей Петри, основная цель которого — описание структуры параллельной программы с большим количеством значений переменных, заданных в терминах императивных языков программирования.

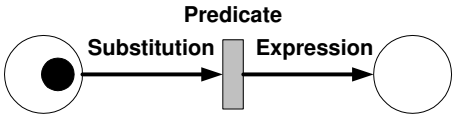

Рис. 2. Последовательный участок сети Петри для программирования

Модель, описанная в терминах сетей Петри для программирования (далее сокращенно СПП), представляет собой либо плоскую сеть Петри, либо иерархическую сеть Петри.

Плоская СПП описывается своей структурой и множеством предписаний, назначаемых каждому элементу структуры (рис. 2).

Структура плоской СПП так же, как и в простых сетях Петри, описывается множеством мест, дуг и переходов. Для описания состояний модели используются места сети Петри, которые изображаются на рисунках кружками; им могут быть назначены предписания, которые описывают маркеры. Для описания событий, возможных в модели, используются переходы, которые изображаются на рисунках прямоугольниками. Переходам могут быть назначены предикаты возбуждения (Predicate), определяющие возможность срабатывания на основании маркеров, поступающих в переход. Дугам, входящим в переход, сопоставляются предписания (Substitution), назначающие входящим в переход маркерам имена, используемые в предикатах и выражениях на исходящих из перехода дугах. Дугам, исходящим из переходов, сопоставляются выражения (Expression), вычисляющие новые значения маркеров на основании значений маркеров, поступающих в переход. Кроме того, переходам могут назначаться предписания, определяющие участие перехода в операциях композиции с другими СПП. Срабатывание перехода СПП возможно только в том случае, если предикат возбуждения выдает значение true на основании входящих в переход маркеров. При срабатывании перехода маркеры, участвовавшие в вычислении предиката возбуждения, изымаются из мест, входящих в переход, и на их основании высчитываются новые значения маркеров на дугах, исходящих из перехода.

Для обеспечения независимости от правил записи предписаний используется интерпретатор, который обеспечивает функционирование конструкций императивных языков программирования в моделях, описанных в терминах СПП.

Иерархическая СПП определяется при помощи операции композиции других СПП. В операции композиции СПП используются две СПП и определенные в них точки доступа. На рисунках точки доступа изображаются квадратиками, а линии, соединяющие их, обозначают операции композиции этих сетей. При этом правила слияния СПП гарантируют, что одинаково помеченные переходы могут сработать только одновременно. На рис. 3 изображен пример описания композиции модели функции с ее вызовом. На рис. 4 изображен пример слияния композиции сетей, изображенных на рис. 3. Здесь и далее переходы, которые входят в точки доступа, указаны на сером фоне; в их названиях слева от точки указано наименование точки доступа, справа — пометка перехода; переходы, которые не входят в точки доступа, указаны на белом фоне.

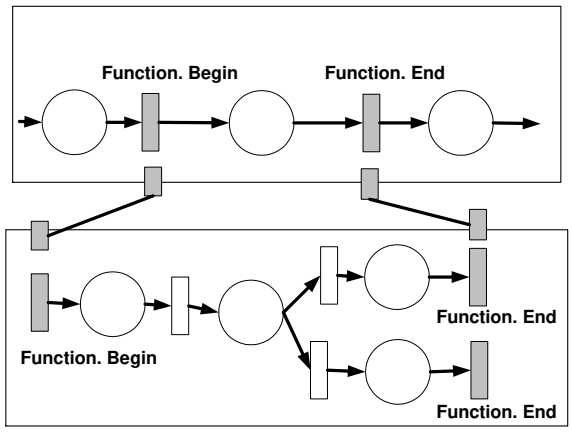

Рис. 3. Композициональное представление вызова функции

4. Семантические конструкции в терминах СПП. Программирование отдельных процессов большинства параллельных программ, работающих в среде передачи сообщений, происходит на императивных языках программирования (Си и его модификации, Паскаль, Фортран, Ада, Кобол) с использованием объектно-ориентированного и процедурного стилей программирования.

Процедурное программирование определяет возможность разбиения программы на составляющие элементы. При этом создание программы идет по принципу сверху вниз, т.е. сначала определяются необходимые для решения программы части, а затем эти части разрабатываются и объединяются. В 60-х годах Дейкстра доказал, что любой алгоритм при процедурном программировании можно реализовать с использованием лишь трех управляющих конструкций: последовательное выполнение выражений, ветвление и цикл [8]. Он же указал, что при наличии соответствующих операторов возможно исключить из языка команду перехода GOTO.

Появление в середине 70-х годов объектно-ориентированного подхода к созданию программ дало возможность использовать сложнейшие структуры данных и заметно упростило структуру программ, что обусловило популярность такого подхода программирования. Тем не менее можно отметить, что парадигма объектно-ориентированного подхода не отрицает, а только расширяет возможности процедурного стиля программирования. Так, в рамках одного и того же языка программирования объектно-ориентированную программу можно представить в виде процедурной программы и наоборот. Здесь имеется в виду, что язык программирования поддерживает оба стиля программирования (например, язык C++).

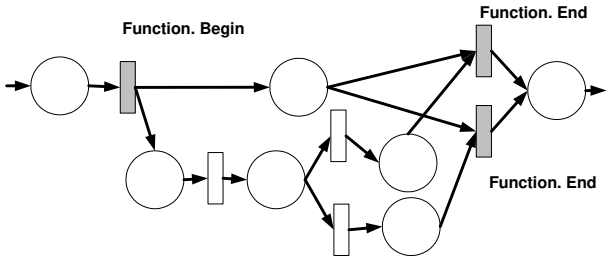

Рис. 4. Пример слияния функции и ее вызова

Таким образом, модель исполнения последовательного процесса в рамках объектно-ориентированного и процедурного программирования выражается через точно обозначенные управляющие конструкции и автономные программные блоки, поддерживающие рекурсию и локальные переменные.

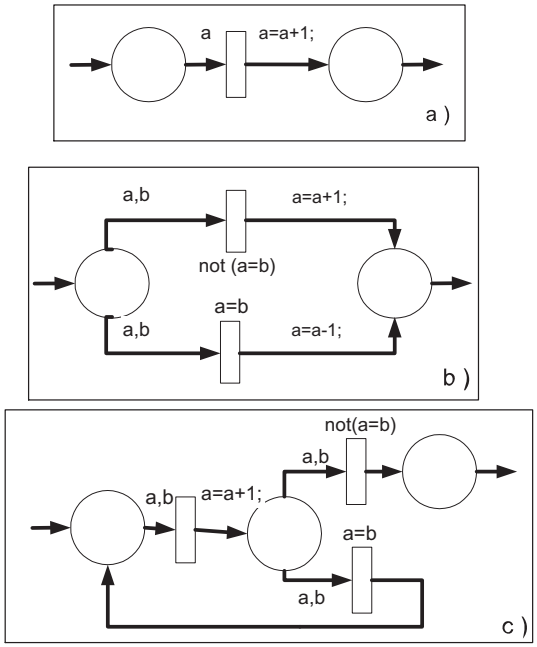

Рис. 5. Представление основных семантических конструкций в сетях Петри для программирования

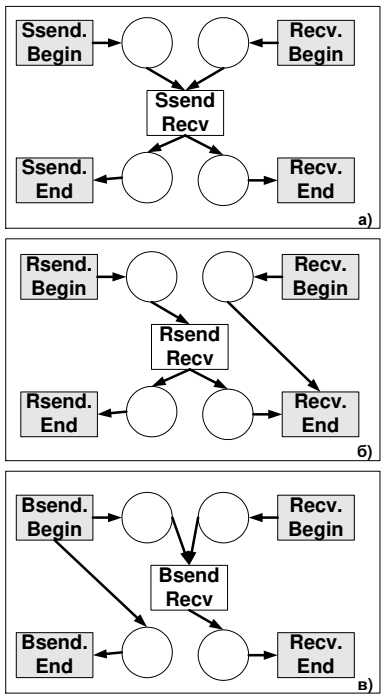

Рис. 6. Сеть Петри для пар функций а) Ssend–Recv, б) Rsend–Recv, в) Bsend–Recv

Представление структуры управления программ в сетях Петри для программирования основывается на четырех основных семантических конструкциях: последовательная запись выражений (рис. 5 а),

конструкция условного выполнения (рис. 5 б), цикл (рис. 5 с) и вызов функции (рис. 3).

Более сложные конструкции языка программирования определяются комбинацией четырех базовых. В сетях Петри для программирования функции могут представляться двумя различными способами: в развернутом виде и в виде одного неделимого действия. В первом случае сама функция выделяется в отдельную подсеть (композициональное представление модели программы) и связывается с основным телом программы механизмом точек доступа. На рис. 3 точки доступа обозначены квадратами на границе сети. Формально точка доступа определяется пометкой и списком переходов, которые ее образуют. Во втором случае вызов функции представляется как конструкция последовательной записи выражения (рис. 5 а).

5. Модель среды взаимодействия в терминах сетей Петри. Рассмотрим пример практического построения одной модели межпроцессного взаимодействия для MPI [10] — одного из наиболее популярных средств на сегодняшний день для организации среды взаимодействия между процессами параллельной программы. MPI предоставляет программисту единый механизм взаимодействия отдельных процессов внутри параллельного приложения независимо от машинной архитектуры. В настоящее время разными коллективами разработчиков написано несколько программных пакетов, удовлетворяющих спецификации MPI (пакеты MPICH, LAM, HPVM и др.).

Определим, что конечной целью построения модели параллельной программы является анализ, направленный на выявление ошибок, возникающих при взаимодействии процессов, и предположим, что для организации межпроцессного взаимодействия в параллельной программе использовались только базовые коммуникационные функций блокирующего типа. Исходя из постановки задачи, сформулированной в рамках модели, мы можем не рассматривать функции, выполнение которых связанно в основном с настройками параметров межпроцессного взаимодействия (таких как Init, Comm_size, Comm_rank и др.).

В стандарте MPI существуют четыре блокирующие функции посылки типа точка–точка с различными режимами выполнения: стандартная посылка — Send, синхронная посылка — Ssend, буферизованная посылка — Bsend и согласованная посылка — Rsend. Единственной рассматриваемой парной функцией (функцией приема) для них является функция Recv.

Согласно стандарта MPI, на вызов блокирующего Send (стандартный коммуникационный режим) накладывается следующее ограничение: он не возвращает управления до тех пор, пока данные и атрибуты сообщения надежно не сохранены в другом месте, чтобы процесс-отправитель мог обращаться к буферу посылки и перезаписывать. Сообщение может быть скопировано непосредственно в соответствующий приемный буфер или во временный системный буфер. Таким образом, в стандартном коммуникационном режиме решение о том, будет ли исходящее сообщение буферизовано или нет, принимает MPI. MPI может буферизовать исходящее сообщение. В таком случае работу Send можно выразить функциями Rsend или Bsend. С другой стороны, буферное пространство может отсутствовать или MPI может отказаться от буферизации исходящего сообщения из-за ухудшения характеристик обмена. В таком случае работу Send можно выразить функцией Ssend. Учитывая сказанное, ограничимся построением модели для трех функций блокирующего типа точка–точка (Ssend, Rsend и Bsend).

Рассмотрим отличия между этими функциями.

Первая функция — это функция синхронной посылки сообщения Ssend, которая должна завершить посылку только по получении (или началу получения) сообщения на принимающей стороне (рис. 6 а).

В отличие от функции Ssend функция Rsend может закончить работу сразу же, как только удается переслать сообщение на сторону приемника. Модель функций Rsend–Recv представлена на рис. 6 б.

Практически аналогичная ситуация имеет место с буферизованной посылкой сообщений Bsend (рис. 6 в). Только в этом случае функция Bsend заканчивает работу, разместив свое сообщение в буфере на стороне передатчика.

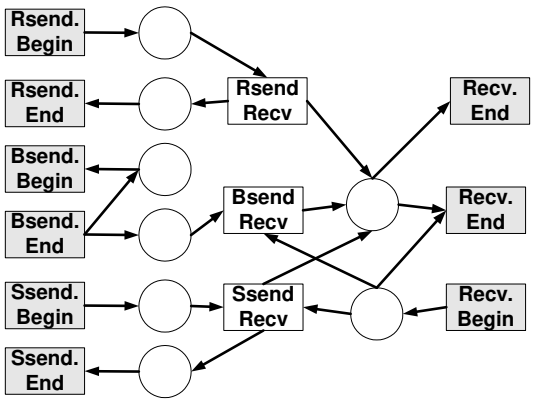

Рис. 7. Совместная модель функций Send–Recv

Конечная модель среды взаимодействия, предоставляемой MPI, составляется из множества всех функций, реализованных в среде. Таким образом, для рассмотренного нами случая — это множество из трех функций Send и одной функции Recv, объединение моделей которых представлено на рис. 7. Объединение построенной модели с моделями последовательных процессов, создание которых описано выше, при помощи механизмов композиции, рассмотренных в разделе 3, представляет собой ко-

нечную модель параллельной программы.

6. Заключение. В статье рассмотрен метод автоматического построения моделей параллельных программ, работающих в рамках среды передачи сообщений, для которых становится возможно построение моделей, исполнение которых будет полностью адекватно исполнению самой параллельной программы. Такое соответствие достигается за счет моделирования потока управления параллельных программ конструкциями сетей Петри, срабатывание которых вызывает модификацию данных в терминах исходного языка параллельной программы. Описание редукции моделей такого рода и практические примеры построения моделей выходят за рамки данной статьи и будут рассмотрены в дальнейших работах по данному направлению.

СПИСОК ЛИТЕРАТУРЫ

1. Средства создания и проектирования параллельных программ (http://parallel.ru/tech/tech_dev/build_par.html)
2. *Havelund K., Pressburger T.* Model checking Java programs using Java PathFinder // Int. Journal on Software Tools for Technology Transfer. 2000. **2**, N. 4. 366–381.
3. SPIN (http://spinroot.com/spin/whatispin.html)
4. *Stansifer R., Beaven M., and Marinescu D.C.* Modeling concurrent programs with colored Petri nets // Journal of Systems and Software. 1994. **26**, N 2. 37–43.
5. *Bause F., et al.* SDL and Petri net perfomance analysis of communicating systems // Proc. IFIP Intern. Symp. on Protocol Specification, Testing and Verification. Warsaw, 1995. 259–272.
6. *Stansifer R. and Marinescu D.* Petri net models of concurrent Ada programs // Microelectronics and Reliability. 1991. **31**, N 4. 577–594.
7. *Харитонов Д.И.* Трансляция параллельных программ, описанных сетями Петри, в исполняемое представление. Владивосток, 2003.
8. *Дейкстра Э.* Дисциплина программирования. М.: Мир, 1978.
9. CPN/Design (http://www.daimi.au.dk/designCPN/)
10. MPI: A Message-Passing Interface Standart // Message Passing Interface Forum. Int. J. Supercomputing. 1994. **8**, N 3/4. 165–416.

Поступила в редакцию
12.05.2005