УДК 519.681.2

© 2018 г. **Д.И. Харитонов**, канд. техн. наук,
Р.В. Парахин,
Г.В. Тарасов
(Институт автоматики и процессов управления, Владивосток)

МОДЕЛИРОВАНИЕ КОЛЛЕКТИВНЫХ ОПЕРАЦИЙ MPI В ТЕРМИНАХ СЕТЕЙ ПЕТРИ

Предложен подход к моделированию MPI операций взаимодействия групп процессов в терминах сетей Петри. Вводятся крупноблочная композициональная модель среды MPI и расширения нотации композициональных сетей Петри для сокращения описания объектов в модели. Описывается модель коммуникатора MPI и приводятся примеры моделей коллективных операций MPI.
Ключевые слова: моделирование систем, сети Петри, интерфейс передачи сообщений, MPI.

DOI: 10.22250/isu.2018.56.35-49

Введение

Большая часть параллельного вычислительного программного обеспечения использует интерфейс передачи сообщений (MPI – Message Passing Interface) для обмена данными расчетов между параллельными процессами. Несмотря на широкое распространение, трудоемкость разработки параллельных программ с использованием MPI чрезвычайно высока. По некоторым данным, на отладку программы тратится до 2/3 всего времени разработки [1]. Связано это не только со сложностью решаемых научных или инженерных задач, но и с недетерминированным поведением параллельных программ, с которым сталкивается программист в процессе поиска причин ошибок. Недетерминированное поведение приводит к отсутствию повторяемости исполнения программы, что препятствует использованию традиционных методов отладки. В результате даже при одинаковых входных данных ошибки могут проявляться не всегда. Они могут проявляться при масштабировании задачи, т.е., например, при запуске количества параллельных про-

цессов, кратного семи или большего определенного числа. При этом с увеличением количества процессов для программ, состоящих из тысяч и десятков тысяч процессов, традиционные средства отладки, ориентированные на анализ и управление потоков команд, становятся фактически непригодными [2, 3].

Можно выделить несколько вариантов преодоления упомянутых проблем, связанных с отладкой распределенных программ. Во-первых, это полный отказ от императивного программирования взаимодействий и переход в область декларативных параллельных конструкций. Например, такой подход поддерживается в системе DVM, разработанной в ИПМ им. Келдыша [4, 5]. Схожие идеи прослеживаются в стандартах OpenMP и OpenACC [6, 7]. Декларативный подход упрощает процесс отладки до состояния, практически близкого к традиционной последовательной отладке. Однако плата за простоту – эффективность и масштабируемость выполнения программы, а также возможность переносимости программы на различные вычислительные и коммуникационные архитектуры, что фактически ограничивает применение такой модели программирования определенным классом задач.

Во-вторых, вариантом, используемым в качестве основного метода отладки параллельных программ, является «посмертная» отладка (post-mortem debugging) с применением различных инструментов воспроизведения исполнения программы (replay-based debugging). К таким инструментам можно отнести Intel Trace Analyzer and Collector (ITAC) [8, 9], Stack Trace Analysis Tool (STAT) [10], Program Record/Replay Toolkit [11] и др. Проблемой применения данного инструментария является отсутствие гарантий, что после окончания отладки не обнаружатся новые скрытые ошибки, не выявленные в результате тестирования и анализа истории исполнения, записанной для одного или нескольких вариантов запуска параллельной программы.

И наконец, третьим вариантом преодоления проблем параллельной отладки является решение более масштабной задачи – частичной или полной верификации программы [12]. Необходимо отметить, что в настоящее время решения данной задачи не существует ни для последовательных, ни для параллельных программ. Однако уже длительное время усилия научного сообщества направлены на построение математического аппарата, и уже есть инструментальных средства, позволяющие проверить правильность отдельных параллельных программ, основываясь только на их исходном коде [13, 14]. Одним из наиболее известных средств верификации является MPI-SPIN [15], являющийся расширением верификатора SPIN [16]. Особенности применения MPI-SPIN связаны в первую очередь с необходимостью перевода исследуемой программы на язык PROMELA, что накладывает дополнительные ограничения на его использование в цикле разработки. Также следует отметить, что реализация MPI-SPIN включает далеко не

все элементы стандарта MPI, а лишь некоторые основные функции взаимодействия. Несмотря на то, что текущие исследования в области верификации идут со значительным отставанием от технологий программирования и архитектур вычислительных систем, направление, связанное с верификацией, является наиболее перспективным, учитывая растущую сложность решаемых задач, языков и технологий программирования, архитектур вычислительных систем.

Данная работа также относится к направлению, связанному с верификацией программ, но в качестве формального инструмента используются сети Петри, хорошо известные как формальный аппарат для моделирования параллельных и распределенных систем [17 – 20]. Для анализа программ в терминах сетей Петри необходима модель, включающая как непосредственно исполняемую программу, так и ее окружение, в частности среду взаимодействия процессов. В нашей статье рассматривается концептуальное описание подхода построения модели MPI. При этом описание ограничивается коллективными функциями, представляющими наибольшую сложность для моделирования. Концептуальный уровень предполагает введение основных понятий без строгой формализации математических определений. Сначала коротко описывается способ моделирования функции и вызова функции с передачей параметров в терминах сетей Петри. Затем вводятся токены, содержащие уникальные идентификаторы для моделирования точек возврата из вызовов функций, идентификаторов процессов, а также идентификаторов коммуникаторов MPI. Затем описывается подход, позволяющий использовать места в качестве списков или таблиц. На основании введенных обозначений приводится композициональное представление среды взаимодействия MPI и описывается понятие коммуникатора. Приводятся примеры моделирования коллективных функций взаимодействия MPI, иллюстрирующие применение введенных понятий.

Интерфейс передачи сообщений

Message Passing Interface (MPI) – это современный, развивающийся стандарт среды взаимодействия вычислительных процессов. Первая версия стандарта была принята в 1994 г. Последняя версия стандарта MPI-3.1 опубликована 4 июня 2015 г. Стандарт описывает интерфейс взаимодействия процессов в виде набора функций с определенными параметрами, каждой из которых придается неформальное описание способа взаимодействия. Стандарт недвусмысленно определяет основные семантические правила взаимодействия процессов и ожидаемое поведение работы соответствующих функций, но не накладывает жестких ограничений для разработчиков на реализацию их внутренней структуры. Благодаря этому существует множество реализаций MPI, совместимых на уровне исходного,

а иногда и на уровне бинарного кода.

Для удобства изучения и использования функции MPI разбивают на группы по их функциональному назначению. Можно выделить следующие группы функций взаимодействия:

передача сообщений между отдельными процессами;

коллективные взаимодействия процессов;

группировка процессов и топологии;

динамическое порождение процессов и управление процессами;

односторонние коммуникации (Get/Put).

Построение полной модели MPI в терминах сетей Петри является технически сложной задачей вследствие большого количества функций, описанных в стандарте, а также из-за большого числа входящих параметров, определяющих исполнение этих функций.

Ориентируясь на долгосрочную цель использования модели в задачах верификации MPI-программ, сформулируем основные тезисы концепции построения модели MPI:

1) *модульный подход* – полная модель MPI должна составляться из элементарных компонент, логически разделенных между собой. Каждая компонента должна представлять собой отдельную сеть Петри с фиксированным интерфейсом сопряжения с другими компонентами. Поскольку стандарт MPI опирается на вызовы функций для осуществления операций взаимодействия, то и в модельном представлении мы будем использовать конструкции, позволяющие промоделировать факт передачи управления внутрь функции MPI;

2) *работа с данными* – корректность взаимодействия процессов зачастую зависит от вычисляемых в процессе работы программы данных (ранги процессов, данные сообщений и т.д.). Модель должна учитывать необходимость представления сложных типов данных для последующего анализа состояний исполнения;

3) *пригодность для анализа* – полученная модель должна быть пригодна для анализа существующими методами и инструментальными средствами на предмет наличия или отсутствия определенных свойств. Под пригодностью следует понимать получение результатов анализа за разумное время с использованием разумного количества вычислительных ресурсов.

Модель функции и передача параметров

Основой для моделирования среды взаимодействия MPI является представление функции и вызова функции. Введем очень коротко набор понятий, необходимых для моделирования программ в терминах сетей Петри. Сеть Петри – это набор, состоящий из множества мест, переходов и дуг. Графически сеть удобно

представлять двудольным ориентированным графом, в котором места обозначаются кругами, а переходы – прямоугольниками. Дуги, исходящие и входящие в переход, определяют исходящее и входящее мультимножества мест переходов. Аналогично определяются исходящее и входящее мультимножества переходов каждого места. Маркировка сети Петри определяется токенами (фишками), размещаемыми в местах сети. Маркировка формально представляется мультимножеством мест. Переход сети Петри называется возбужденным, если мультимножество входящих мест этого перехода меньше или равно маркировке сети Петри. Функционирование сети определяется срабатыванием возбужденных переходов, удаляющим токены из входящих мест и добавляющим токены в исходящие места переходов. Переходы сети Петри могут иметь пометки, определяющие видимость срабатывания перехода в виде мультимножества символов из заданного алфавита. На основе пометки определяются точки доступа по переходам к сетям Петри. Синхронизация двух сетей Петри по точкам доступа позволяет определить события одновременного срабатывания одинаково помеченных в обеих сетях Петри переходов. При добавлении направления синхронизации можно разрешить срабатывание в сети только переходов, имеющих во второй сети одинаково помеченное мультимножество переходов. Направление синхронизации определяет тип используемых точек доступа.

В статье [21] авторами предложен композициональный подход к моделированию императивных программ. Каждой функции программы сопоставляется один объект, состоящий из сети Петри, моделирующей поток управления этой функции, входящей точки доступа и множества исходящих точек доступа по переходам. Входящая точка доступа моделирует точку входа в функцию. Исходящие точки доступа моделируют точки возврата из функции. Вызову функции в теле императивной программы сопоставляется тройка *переход-место-переход*, в которой первый переход отмечает передачу управления в функцию, а второй переход – прием управления от функции. При таком подходе композициональная модель императивной программы представляет собой множество сетей Петри, связанных между собой точками доступа в соответствии с вызовами функций, заданных в программе.

Из композиционального представления может быть получено плоское представление выполнением всех операций синхронизации.

На рис. 1 сверху показан пример вызова из сети *N1*, представляющей поток управления, простой функции *Test*, моделируемой сетью *N2*. Переходы, моделирующие передачу управления в функцию, имеют пометку *Test:beg* и *Test:end*.

Цветные сети Петри можно определить как расширение сетей Петри, добавляющее к местам тип данных (множество значений) токенов, попадающих в это место [22]. Дугам, исходящим из мест, добавляется имя переменной токена

для манипуляции с данными в переходе и на дугах, исходящих из переходов. Переходам добавляется предикат возбуждения, определяемый как функция от переменных, определенных на входящих в переход дугах.

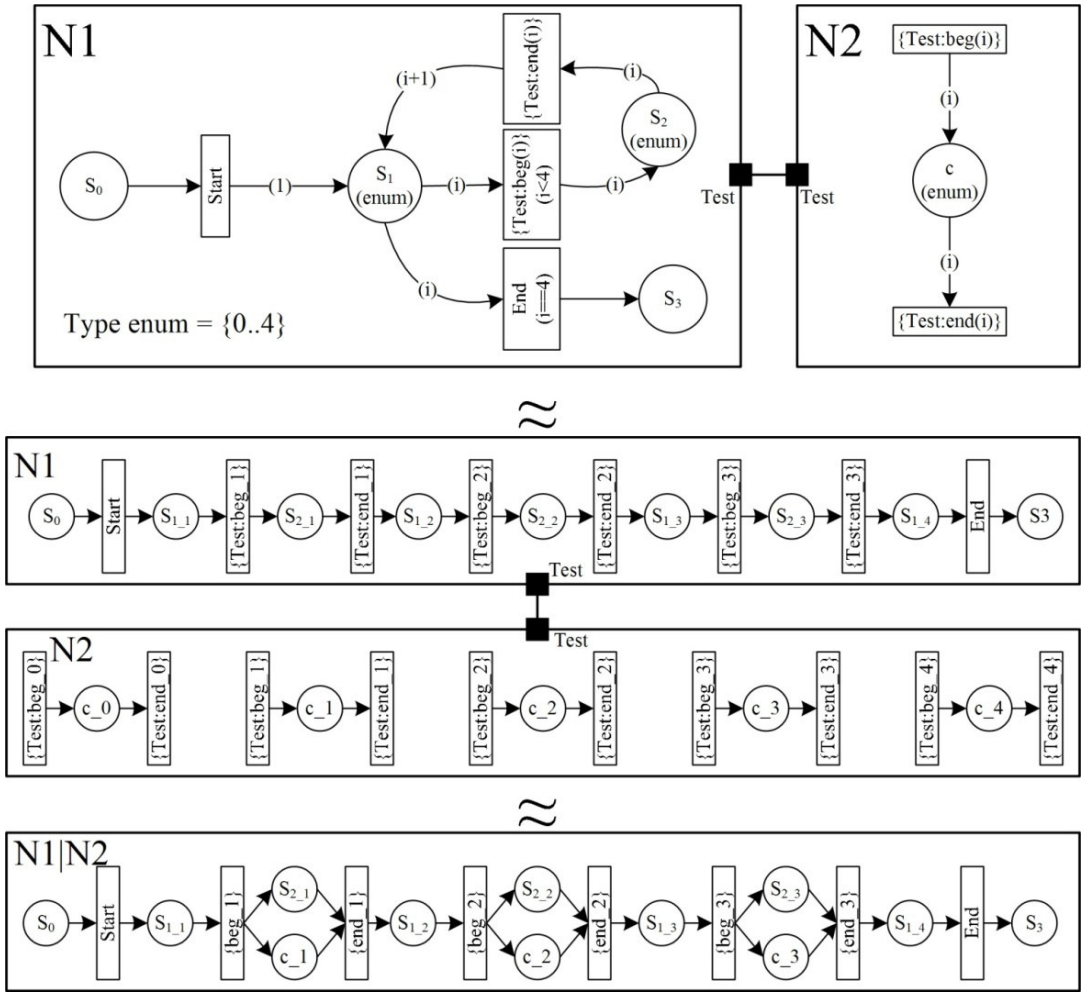

Рис. 1. Эквивалентные преобразования параметризованных сетей.

Исходящим из перехода дугам добавляется функция расчета значения токена относительно переменных, определенных на входящих в переход дугах. При дискретном ограниченном множестве значений токенов в местах цветные сети Петри могут быть преобразованы в обычные сети Петри. Для этого каждое место расщепляется на множество мест, каждому из которых сопоставляется одно значение из множества значений типа данных исходного места, а каждый переход расщепляется на количество вариантов результирующих наборов токенов, помещаемых в выходные места. Такое преобразование называется разверткой цветной сети.

Для моделирования передачи параметров при вызове функции предлагается использовать параметризованную *пометку переходов*, использующую в дополнение к символам алфавита еще и наименование токенов, входящих в переход. При развертке такой цветной сети Петри пометка расщепленных переходов будет состоять из исходного мультимножества символов, дополненного символами, обо-

значающими значение каждого параметра в исходном переходе. Такая пометка позволяет при синхронизации определять события одновременного срабатывания одинаково помеченных и имеющих одинаковые параметры в обеих сетях Петри переходов, что может быть использовано для моделирования передачи значений из одной сети Петри в другую как в цветных сетях Петри, так и в их развертке. В средней части рис. 1 дана развертка вызова функции *Test* относительно значений типа данных enum, а в нижней части – результат синхронизации потока управления и функции. Необходимо отметить, что нижняя сеть, полученная слиянием разверток сетей *N*1 и *N*2, эквивалентна развертке слияния исходных сетей.

Далее на рисунках пометка, относящаяся к работе с данными, будет отображаться в круглых скобках; пометка, относящаяся к точкам доступа, будет отображаться в фигурных скобках. При моделировании групповых функций будет использован еще один вид групповой пометки, в квадратных скобках. Линии соединения точек доступа будут обозначаться сплошной линией, когда подразумевается, что должна быть выполнена операция синхронизации по переходам. Пунктирная линия соединения обозначает объединение точек доступа нескольких внутренних объектов в точку доступа внешнего объекта.

Моделирование уникальных идентификаторов объектов программы

Нотация цветных сетей Петри позволяет предписывать токенам перечисляемые типы данных. Развертка мест и переходов по всем допустимым значениям токенов преобразует цветную сеть Петри в обычную. Существует, однако, ряд объектов программы, для которых необходимы уникальные идентификаторы, позволяющие отличить один объект от другого, при этом само по себе значение идентификатора не имеет смысловой нагрузки и не меняется на протяжении всего жизненного цикла модели. Применение отдельного типа данных для идентификаторов уникальных объектов позволяет сократить затраты на построение дерева достижимости. Для таких идентификаторов используются две функции преобразования на дугах, исходящих из переходов. Первая функция простого копирования значения идентификатора, а вторая функция *New* – для создания нового идентификатора.

Особенность применения специализированного типа данных для уникальных идентификаторов проявляется при развертке цветной сети. Так как уникальные идентификаторы не меняются в ходе своего жизненного цикла, то набор мест таких идентификаторов расщепляется на независимые множества так, что токен с одним уникальным идентификатором никогда не попадет в места с другим идентификатором. Это, в свою очередь, позволяет рассматривать исходную сеть как множество отдельных сетей, параметризованных идентификатором.

На рис. 2 сверху изображен пример сети Петри, использующей уникальный идентификатор. Ниже представлено расщепление мест и переходов этой сети по значениям уникального идентификатора. Видно, что между множествами мест и переходов, принадлежащих разным значениям идентификатора, нет пересечений. Поэтому сеть может быть представлена в виде композиции сетей, где движение токенов, сопоставленных одному значению уникального идентификатора, вынесено в отдельные сети.

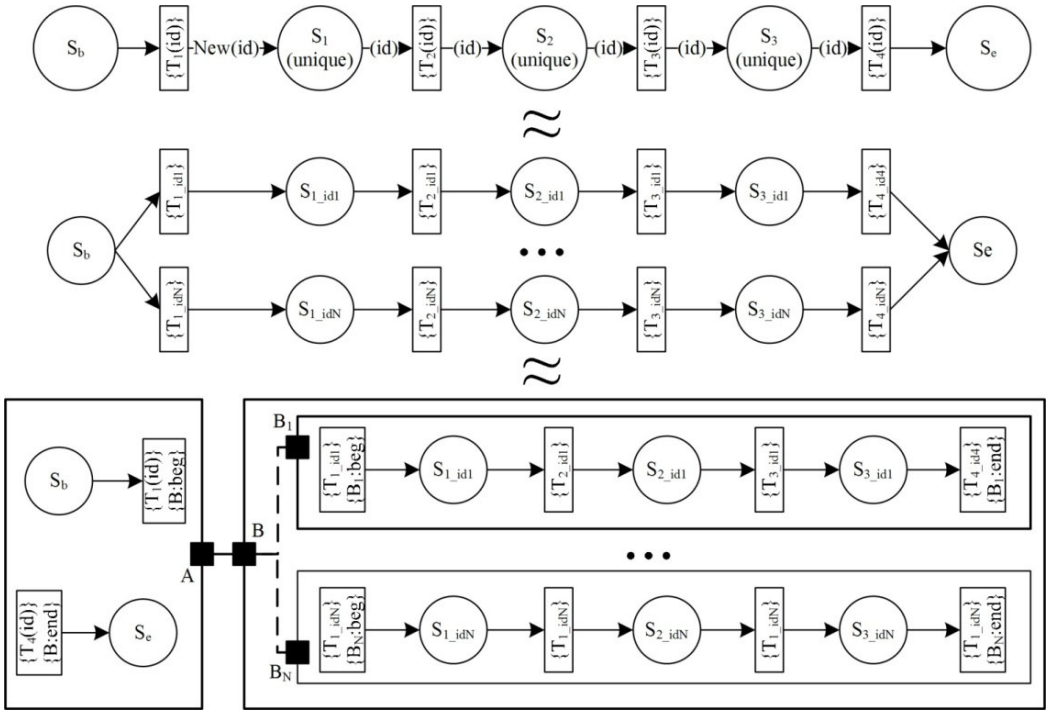

Рис. 2. Эквивалентные сети Петри с уникальным идентификатором.

Подход к моделированию списков

Коллективные операции MPI используют в качестве параметра коммуникатор, определяющий группу процессов, в рамках которой будет выполняться обмен сообщениями. Для формирования группы процессов и дальнейшей работы с ней необходим механизм работы со списками в сетях Петри. С этой целью предлагается использовать так называемый «синтаксический сахар», задачей которого является увеличение выразительной мощности сетей Петри без увеличения вычислительной мощности. Фактически «синтаксический сахар» – это подстановка шаблонных конструкций в модель, использующую этот «сахар». Подстановка, аналогичная, например, используемым в C++ макросам, позволяющим существенно упростить рутинные обращения к сложным функциям и подпрограммам.

Базовым элементом, необходимым для работы со списками, является «защищенное» место, про которое заранее известно ограничение на максимальное количество токенов. Любые входящие или исходящие переходы в «защищенное»

место должны срабатывать по одному за раз, при этом есть возможность определить текущее количество токенов в месте. На рис. 3 показаны шаблоны конструкций для подстановки вместо «защищенного» места. «Защищенное» место *A* представлено на рис. 3 в виде двойной окружности, где внутренняя окружность отображена пунктиром. Слева на рисунке отображено преобразование исходящих из «защищенного» места дуг, в центре – входящих, а справа на рисунке изображено преобразование, когда у перехода есть входящие и исходящие из «защищенного» места дуги. «Защищенное» место *A* в шаблонах представлено двумя местами, одно из которых используется токенами из «защищенного» места, а второе содержит счетчик количества токенов. Для обозначения максимального значения счетчика используется нотация *A*, где *A* – это идентификатор места. Буквами *k* и *m* без скобок на дугах обозначена их кратность, надпись *(N)* обозначает значение счетчика токенов.

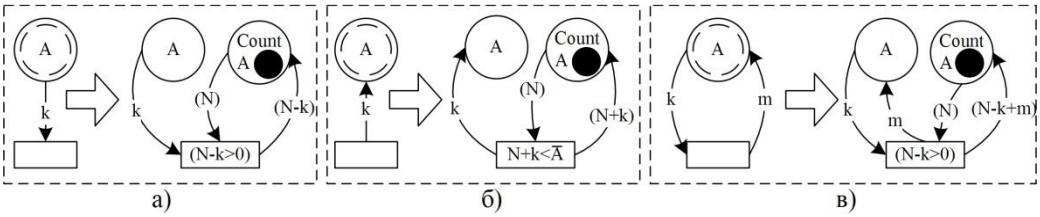

Рис. 3. Подстановка «защищенного» места: а) входящего в переход; б) исходящего из перехода; в) общий случай.

С использованием «защищенного» места нетрудно определить коллективную операцию перемещения всех токенов из одного места в другое. Графически эта операция помечается предписанием *[All]* в квадратных скобках для перехода. На рис. 4 отображена шаблонная подстановка для реализации этой операции.

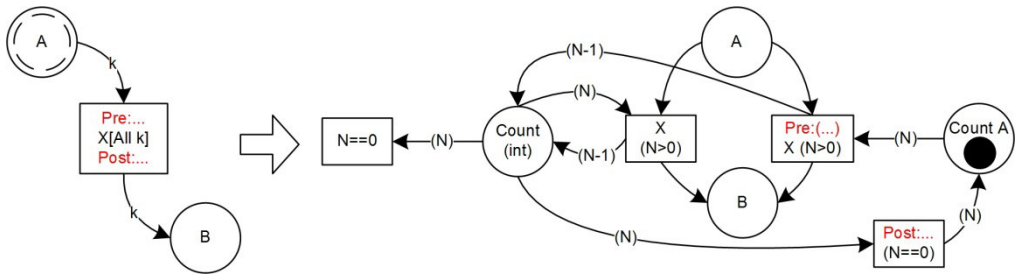

Рис. 4. Перемещение всех токенов.

Нетрудно видеть, что первым действием в этой операции изымается токен из места счетчика. Поэтому до конца перемещения токенов никакие другие действия с «защищенным» местом не могут быть совершены. Последним же действием операции возвращается токен с нулевым значением, т.е. вместо счетчика, опять разрешая действия над «защищенным» местом. Этот шаблон позволяет обозначить условия для начала срабатывания операции и событие окончания перемещения токенов при помощи префиксов *Pre:* и *Post:* у предписаний коллективного перехода.

И наконец, далее нам потребуются операции получения максимальных и минимальных значений из списка. Для этого вводится обозначение операций *Min* и *Max,* помещаемых в квадратные скобки в предписаниях на переходе, зависящем от экстремальных значений. На рис. 5 изображена подстановка, необходимая для получения из «защищенного» цветного места минимального значения.

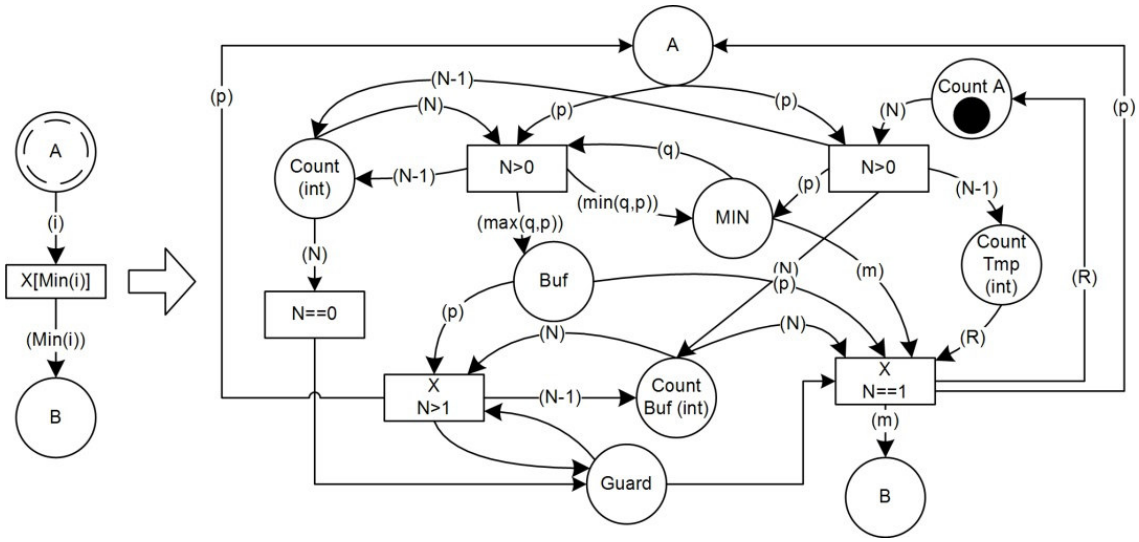

Рис. 5. Получение токена с минимальным значением.

Как видно на рисунке, сначала из «защищенного» места изымается токен в буферное место, названное *MIN*, одновременно изымается и токен из сопутствующего счетчика. Затем все остальные токены в исходном месте сравниваются с токеном из *MIN*. По окончании сравнения эти токены возвращаются в исходное место. После всего этого срабатывает переход, зависящий от минимального значения, и одновременно возвращается токен в счетчик «защищенного» места.

Моделирование групповых функций MPI

Стандарт MPI обеспечивает набор операций коллективного взаимодействия, предназначенных для сбора и распределения данных, синхронизации процессов и выполнения вычислительных функций над распределенными массивами данных. Все групповые функции принимают в качестве одного из входящих параметров коммуникатор. Этим термином объединяется группировка процессов, участвующих в операции коллективного взаимодействия, а также определенный в этой группе порядок процессов (их ранги). С технической точки зрения коммуникатор также может скрывать под собой способ доставки сообщений, разделяя группы процессов, размещаемые в общей памяти и в коммуникационной сети.

Для построения общей модели групповых функций MPI выделяются три различных по функциональности блока. На рис. 6 изображено композициональное представление модели групповых операций MPI, в которой эти три блока выделены в отдельные сети Петри.

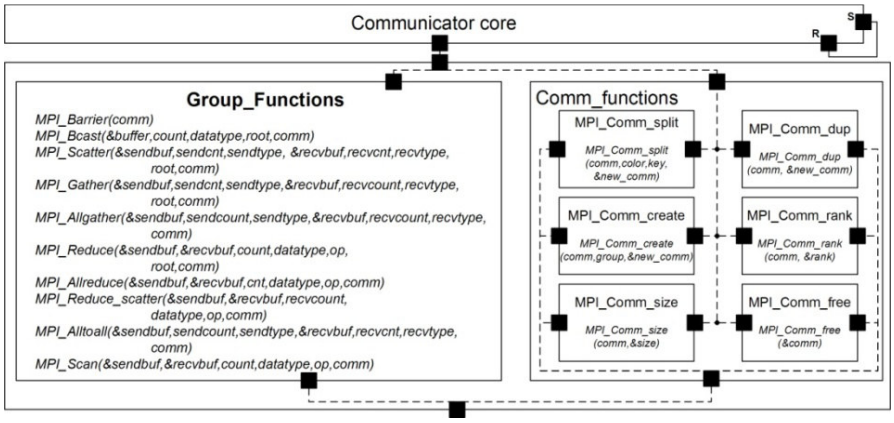

Рис. 6. Общая модель групповых операций MPI.

Сеть *Comm_functions* состоит из множества моделей функций MPI, предназначенных для работы с коммуникаторами, в том числе для создания новых и удаления существующих коммуникаторов. Сеть *Group_functions* содержит множество моделей групповых функций MPI, не взаимодействующих между собой. Сеть *Communicator_core* моделирует внутреннее устройство коммуникатора, а также внутренние взаимодействия, необходимые для реализации функций MPI. Пунктирные линии между точками доступа отображают объединение точек доступа в одной сети. Сплошные линии между точками доступа соответствуют операциям синхронизации по точкам доступа. Такое композициональное представление позволяет «манипулировать» с используемой моделью в процессе ее анализа. Все модели не используемых в анализируемой программе функций могут быть безболезненно удалены из композиции.

Множество групповых функций взаимодействия MPI внесено в предписания сети *Group_functions* (рис. 6). В качестве примера на рис. 7 изображена модель функции *MPI_Bcast* из этого множества.

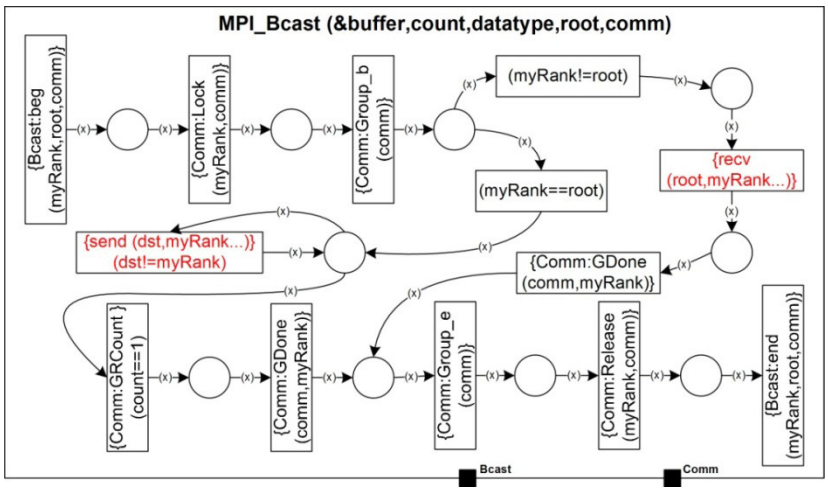

Рис. 7. Модель функции MPI_Bcast.

Модель функции состоит из вступления и заключения, практически одинакового для всех групповых операций, а также из части, реализующей собственную функциональность. Начальный и конечный переходы в этой модели помече-

ны точкой доступа *Bcast* как переходы, необходимые для оформления вызова функции из каждого процесса параллельной программы. Для синхронизации с коммуникатором переходы модели помечены точкой доступа *Comm*. Вторым переходом модель функции блокирует коммуникатор процесса от выполнения альтернативных групповых операций. Когда все процессы заблокируют на своей стороне коммуникатор, третий переход переведет коммуникатор в выполнение групповой операции у всех процессов. На этом заканчивается вступительная часть. В собственной части все процессы, ранг которых не совпадает со значением *root,* принимают данные широковещательной рассылки, а процесс с рангом *root* рассылает данные остальным процессам. В заключительной части каждый процесс, выполняющий функцию *MPI_Bcast*, сообщает коммуникатору, что выполнение групповой операции закончено. После чего процессы дожидаются выхода коммуникатора из групповой операции, затем каждый процесс освобождает коммуникатор для других групповых операций и выходит из функции.

Работа групповых функций опирается на взаимодействие с коммуникатором, вариант модели которого изображен на рис. 8.

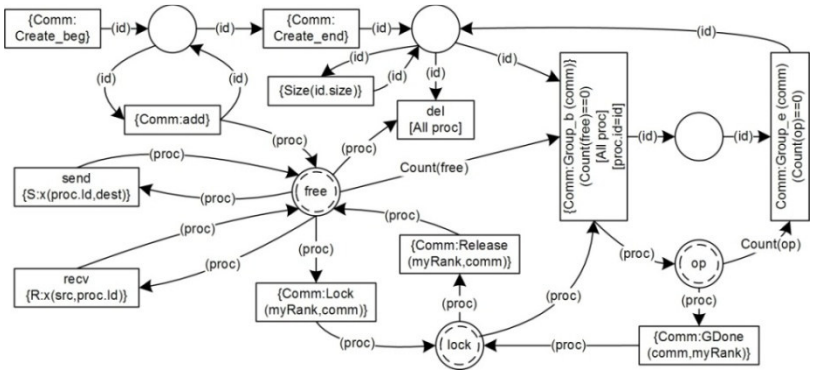

Рис. 8. Модель коммуникатора MPI.

В этой модели используются два типа токенов. Первый тип – это токен, соответствующий абстрактному объекту коммуникатора в параллельной программе, он содержит уникальный идентификатор коммуникатора и количество процессов в коммуникаторе. Второй тип токена соответствует участию процесса параллельной программы в коммуникаторе и содержит уникальный идентификатор коммуникатора, уникальный идентификатор процесса, а также ранг процесса в коммуникаторе. На рисунке используется предписание *(id)* на дугах для отображения движения токенов первого типа токенов и *(proc)* – для второго. Пара переходов *Create_beg* и *Create_end* моделирует событие создания коммутатора. Переход *Create_beg* заводит токен с уникальным идентификатором коммуникатора. Необходимо отметить, что для каждого коммуникатора в параллельной программе будет только один токен первого типа. Далее переход *add* позволяет добавить процессы параллельной программы в коммуникататор. Переход *Create_end* заканчивает создание коммуникатора, после чего могут вызываться групповые

функции, собственная часть каждой из которых начинается переходом *Group_b* и заканчивается переходом *Group_e*. Коммуникатор завершает свою работу «шаблонным» переходом del, удаляющим токены всех процессов коммуникатора.

Блок *Comm_functions* содержит шесть моделей функций для работы с коммуникатором, все они отображены на рис. 6 в виде сетей Петри без содержимого. Каждая из этих моделей имеет внутреннюю и наружную точку доступа. Все внутренние точки доступа разных функций объединяются в одну для синхронизации с сетью *Communucator_core*. На рис. 8 продемонстрирован подход к описанию функций для работы с коммуникатором на примере функции *MPI_Comm_split*. Модель состоит из трех сетей Петри, первая из которых отображает общий поток управления, начинающийся тремя вступительными переходами и заканчивающийся четырьмя заключительными переходами групповой операции. Собственная часть функции состоит из двух процедур, контролируемых дополнительными сетями. Первая процедура состоит в выборе максимального значения параметра цвета и всех процессов параллельной программы, вызвавших функцию с этим значением. Процессы, вызвавшие функцию с другим значением параметра цвета, перемещаются во временный буфер, ожидая выполнения второй процедуры, которая заводит новый коммуникатор и назначает ранги процессам этого коммуникатора.

Необходимо отметить, что при построении моделей функций MPI везде, где предполагается накопление токенов, применяются «защищенные» места. Это позволяет гарантировать ограниченность сети Петри за счет заблаговременного выбора параметров сети разработчиком модели. Кроме того, использование «защищенных» мест и коллективных переходов существенно сокращает модели функций, одновременно облегчая их прочтение.

Заключение

В настоящей статье изложены в неформальном виде некоторые концепции, позволяющие моделировать групповые функции стандарта MPI в терминах сетей Петри. Продемонстрировано, как при объединении нотаций композициональных и цветных сетей Петри параметризовать пометку точек доступа, чтобы имитировать передачу параметров из одной сети Петри в другую при синхронизации. При моделировании MPI такая пометка точек доступа необходима для более полной спецификации вызовов функций. Показано, что использование специализированного типа для неизменяемых в процессе всего «жизненного цикла» идентификаторов позволяет упростить анализ моделей. Такие идентификаторы целесообразно применять для токенов, моделирующих процессы и коммуникатор в параллельной программе. Предложен вариант шаблонной конструкции «защищенного»

места для описания списковых значений. «Защищенное» место позволяет срабатывать только одному инцидентному с ним переходу за раз и допускает сложные операции по переносу всех токенов, получению минимального и максимального значения токенов вместе. Хотя формализм цветных сетей Петри позволяет специфицировать операции над списками, предложенный в настоящей статье «синтаксический» сахар делает спецификацию намного более удобной. В последнем разделе статьи приведена общая модель групповых операций MPI и два примера моделирования функций, иллюстрирующих применение концепций. Авторы предполагают продолжить работы по моделированию библиотеки MPI рассмотрением функций группировки процессов и топологий, функций динамического порождения процессов и управления процессами, а также односторонних коммуникаций. Отдельную задачу авторы видят в построении наглядных примеров моделирования параллельных программ, иллюстрирующих применение модели библиотеки MPI. Необходимо отметить, что авторам не встречалось публикаций, посвященных построению формальных моделей протокола MPI, в то время как применение предложенных концепций для моделирования MPI позволяет существенно приблизиться к построению модели среды взаимодействия, необходимой для верификации параллельных программ.

ЛИТЕРАТУРА

1. *Власенко В.Ю.* Модель масштабируемой системы автоматического контроля корректности параллельных программ // Вестник Новосибирского государственного университета. Серия "Информационные технологии". – 2009. – Т. 7, № 4. – С. 53-65.
2. *Афанасьев К.Е., Власенко А.Ю.* Семантические ошибки в параллельных программах для систем с распределенной памятью и методы их обнаружения современными средствами отладки // Вестник Кемеровского государственного университета. – 2009. – №2(38). – С. 13-20.
3. *Галатенко В.А., Костюхин К.А.* Проблемы отладки многопроцессорных систем // Программные продукты и системы. – 2017. – Т. 30, № 3. – С. 378-383.
4. *Бахтин В.А., Клинов М.С., Крюков В.А. и др.* Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами // Вестник Южно-Уральского университета. Серия «Математическое моделирование и программирование». – 2012. – Вып. 12, № 18 (277). – С. 82-92.
5. *Бахтин В.А., Колганов А.С., Крюков В.А., и др.* Расширение возможностей DVM-системы для решения задач, использующих нерегулярные сетки // Труды Международной конференции "Суперкомпьютерные дни в России". – М.: Изд-во МГУ, 2016. – С. 596-603.
6. OpenMP Application Programming Interface. Version 4.5 November 2015 / OpenMP Architecture Review Board. URL: http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf (дата обращения: 09.11.2017).
7. The OpenACC Application Programming Interface. Version 2.5, October 2015. URL: https://www.openacc.org/sites/default/files/inline-files/OpenACC_2pt5.pdf (дата обращения: 09.11.2017).

8. Intel Trace Analyzer and Collector. https://software.intel.com/en-us/intel-trace-analyzer (дата обращения: 09.11.2017).
9. P. Ohly W. Krotz-Vogel. Automated MPI Correctness Checking: What If There Were a Magic Option? // Proceedings of 8th LCI International Conference on High-Performance Clustered Computing. – South Lake Tahoe, California, 2007. URL: http://www.linuxclustersinstitute.org/conferences/archive/2007/PDF/ohly_19839.pdf (дата обращения: 09.11.2017).
10. *Dong H. Ahn, Bronis R. de Supinski, Ignacio Laguna, Gregory L. Lee, Ben Liblit, Barton P. Miller, Martin Schulz.* Scalable Temporal Order Analysis for Large Scale Debugging // Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (Supercomputing 2009). – Portland, USA. – Article No. 44.
11. *Patil H., Carlson T.E.* Pinballs: Portable and Shareable User-level Checkpoints for Reproducible Analysis and Simulation // Proceedings of the Workshop on Reproducible Research Methodologies (REPRODUCE'2014). – Orlando, USA.
12. *Кларк Э.М., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking / пер. с англ., под ред. Р. Смелянского. – М.: МЦНМО, 2002.
13. *Gopalakrishnan G., Kirby R.M., Siegel S., Thakur, R., Gropp W., Lusk E., De Supinski B.R., Schulz M., Bronevetsky G.* Formal analysis of MPI-based parallel programs // Communications of the ACM. – 2011. – 54(12). – P. 82-91.
14. *Цимина Н.А., Чернов А.В., Кононенко А.А.* Применение алгоритмов верификации к программам, использующим интерфейс MPI // Интернет-журнал «Науковедение». – 2012. – №4 (13). – С. 224.
15. *Siegel S.F.* Model Checking Nonblocking MPI Programs // In: Cook B., Podelski A. (eds) Verification, Model Checking, and Abstract Interpretation. VMCAI 2007. Springer, Berlin, Heidelberg. – Lecture Notes in Computer Science. – V. 4349. – P. 44-58.
16. *Holzmann G.J.* The SPIN Model Checker: Primer and Reference Manual 1st Edition. – Addison-Wesley Professional, 2003..
17. *Котов В.Е.* Сети Петри. – М.: Наука. Глав. ред. физ.-мат. лит-ры, 1984.
18. *Питерсон Дж.Л.* Теория сетей Петри и моделирование систем / пер. с англ. – М.: Мир, 1984.
19. *Ломазова И.А.* Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой. – М.: Научный мир, 2004.
20. *Зайцев Д.А.* Универсальная сеть Петри // Кибернетика и системный анализ. – 2012. – № 4. – С. 24-39.
21. *Тарасов Г.В., Харитонов Д.И., Голенков Е.А.* Об одном представлении функции в модели императивной программы, заданной сетями Петри // Моделирование и анализ информационных систем. – 2011. – Т. 18, № 2. – С. 18-38.
22. *Jensen K.* Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts. Monographs in Theoretical Computer Science. – Springer-Verlag, 2nd corrected printing, 1997.

Статья представлена к публикации членом редколлегии М.А. Гузевым.

E-mail:
Харитонов Дмитрий Иванович – demiurg@dvo.ru;
Парахин Роман Валерьевич – fadak@dvo.ru;
Тарасов Георгий Витальевич – george@dvo.ru.