ПОВТОРНОЕ ИСПОЛНЕНИЕ ПАРАЛЛЕЛЬНЫХ MPI-ПРОГРАММ С ПРИМЕНЕНИЕМ МОДЕЛЕЙ В СЕТЯХ ПЕТРИ

Г.В. Тарасов

Отладка параллельного программного обеспечения весьма сложный и трудоемкий процесс. В настоящее время большинство средств отладки параллельных программ заимствованы из последовательного программирования. Так, основной метод отладки состоит в циклическом выполнении трех основных шагов: пошел, остановился, посмотрел, а основным инструментом отладки является традиционный отладчик, расширенный возможностями одновременного контроля нескольких потоков управления. Удобство и эффективность применения последовательных средств отладки обусловлены принципом повторяемости исполнения. Однако, если в последовательном программировании этот принцип сохраняется фактически всегда, то в параллельном программировании наблюдается обратная ситуация. В результате, для одной и той же параллельной программы с одним и тем же набором входных данных можно получить несколько различных вариантов ее исполнения. Следствием этой проблемы является сложность выявления и обнаружения ошибок в программе, а также сложность поиска причин их возникновения. Таким образом, актуальной задачей является разработка и реализация современных отладочных средств, обеспечивающих повторное исполнение параллельных программ.

В данной работе предлагается подход, позволяющий добиться однозначного исполнения параллельной MPI-программы и частично разрешить сформулированную проблему. Для этого предлагается использовать модель среды взаимодействия процессов в терминах простых сетей Петри, а именно, модель MPI, которая не только формально описывает работу среды, но и контролирует порядок и допустимость выполнения тех или иных операций [1]. В первой части работы приводится краткая теория простых сетей и дается описание модели MPI для некоторого набора MPI-вызовов. Далее описываются принципы связи исполняемого представления программы и модели. В заключении рассматривается архитектура инструментального средства, использующего модель среды взаимодействия для контроля выполнения MPI-вызовов.

Простой сетью Петри [2] называется набор $N=(S,T,A,M_0)$, где S – множество мест, T – множество переходов, $F=\mu(S\times T\cup T\times S)$ – конечное мультимножество дуг, описывающее связь мест и переходов и $M_0\in\mu S$ – начальная маркировка. Места моделируют возможные состояния, которые может принимать MPI-среда, переходы — события, происходящие в MPI-среде, фишки — процессы MPI-программы. Маркировка в модели описывает текущее состояние, в котором находится MPI-среда в результате обращения к ней со стороны процессов. Обозначим через $^o(),()^o$ входные и выходные функции элементов. Переход t в модели называется возбужденным при некоторой маркировке M, если выполняется условие $M\geq{}^o t$, где $^o t$ обозначает множество входных мест перехода. Возбужденным шагом называется множество переходов, для каждого из которых выполняется предыдущее условие. Если переход (или шаг) является возбужденным при некоторой маркировке M, то он может сработать, при этом образуется новая маркировка M' такая, что $M'=M-{}^o t+t^o$, где t^o обозначает множество выходных мест перехода. Под шагом срабатывания также будем понимать многократное срабатывание одного и того же перехода.

Стандарт MPI определяет большое количество функций, позволяющих организовать разнообразные схемы обмена данными между процессами. В данной работе ограничимся использование небольшого, но часто используемого, набора функций. MPI_Init и MPI_Finalize — функции инициализации и завершения работы с MPI-средой. MPI_Send и MPI_Recv — базовые функции блокирующих парных взаимодействий. MPI_Bcast — функция широковещательной рассылки данных всем процессам. MPI_Barrier – функция барьерной синхронизации. MPI_Scatter и MPI_Gather — групповые функции распределения и сбора данных со всех процессов. Также ограничимся использованием только базового коммуникатора, который доступен по умолчанию — MPI_COMM_WORLD.

Модель в терминах сетей Петри показана на рисунке 1. Для удобства восприятия модель представлена в виде отдельных компонент, каждая из которых описывает схему работы одного или нескольких MPI-вызов. Места с одинаковыми именами (отображаются внутри кружка) обозначают одно и то же место. Дадим некоторые пояснения при условии, что исследуемая параллельная программа состоит из N процессов. Рассмотрим рисунок 1а, на котором описана структура выполнения двух вызовов: MPI_Init и MPI_Finalize. Место BEGIN содержит N фишек и обозначает начальное состояние параллельной программы. Первым действием, которое должен совершить каждый процесс, является вызов функции инициализации среды MPI. Этот факт отражает переход INIT. В результате срабатывания перехода все процессы переходят в состояние P и G, где P означает, что процесс может вызвать любую функцию парного взаимодействия, а G означает, что процесс может вызвать любую функцию группового взаимодействия. Иначе места P и G можно охарактеризовать как основные рабочие состояния процессов, с которых начинается любое взаимодействие в

параллельной программе. Соответственно, по окончанию выполнения все N процессов должны выполнить действие FINALIZE, которое переводит модель в конечное состояние END. На рисунке 1б показана модель выполнения функций блокирующего парного обмена сообщениями (MPI_Send и MPI_Recv), которая описывается двумя событиями: начало вызова и завершение вызова. Промежуточное место DoIt описывает факт блокировки процесса. При выполнении парных блокирующих вызовов процесс не может участвовать в групповой операции, что отражено использованием одной фишки из места G. На рисунке 1в показана модель выполнения групповых функций (MPI_Bcast, MPI_Scatter и MPI_Gather). Здесь уже различается четыре типа событий: вход и выход главного процесса (Root) и вход и выход подчиненных процессов (NonRoot). Промежуточные места s_1, s_2 и s_3 обозначают, соответственно, выполнение операции главным процессом, выполнение операции подчиненным процессом и допуск подчиненного процесса до взаимодействия с главным. При этом соответствующие фишки извлекаются из места G для обозначения участия процесса в групповой операции. По завершению выполнения групповой операции главным или подчиненным процессом, соответствующие фишки возвращаются в места P и G. На рисунке 1г показана модель выполнения функции барьерной синхронизации, которая также состоит из двух действий: вход и выход из барьерной функции. При этом выход из барьерной функции возможен только в том случае, когда во взаимодействии поучаствовали все N процессов. Таким образом, данная модель описывает предполагаемый правильный ход следования действий, которые должны выполнять процессы программы при использовании соответствующих MPI-вызовов.

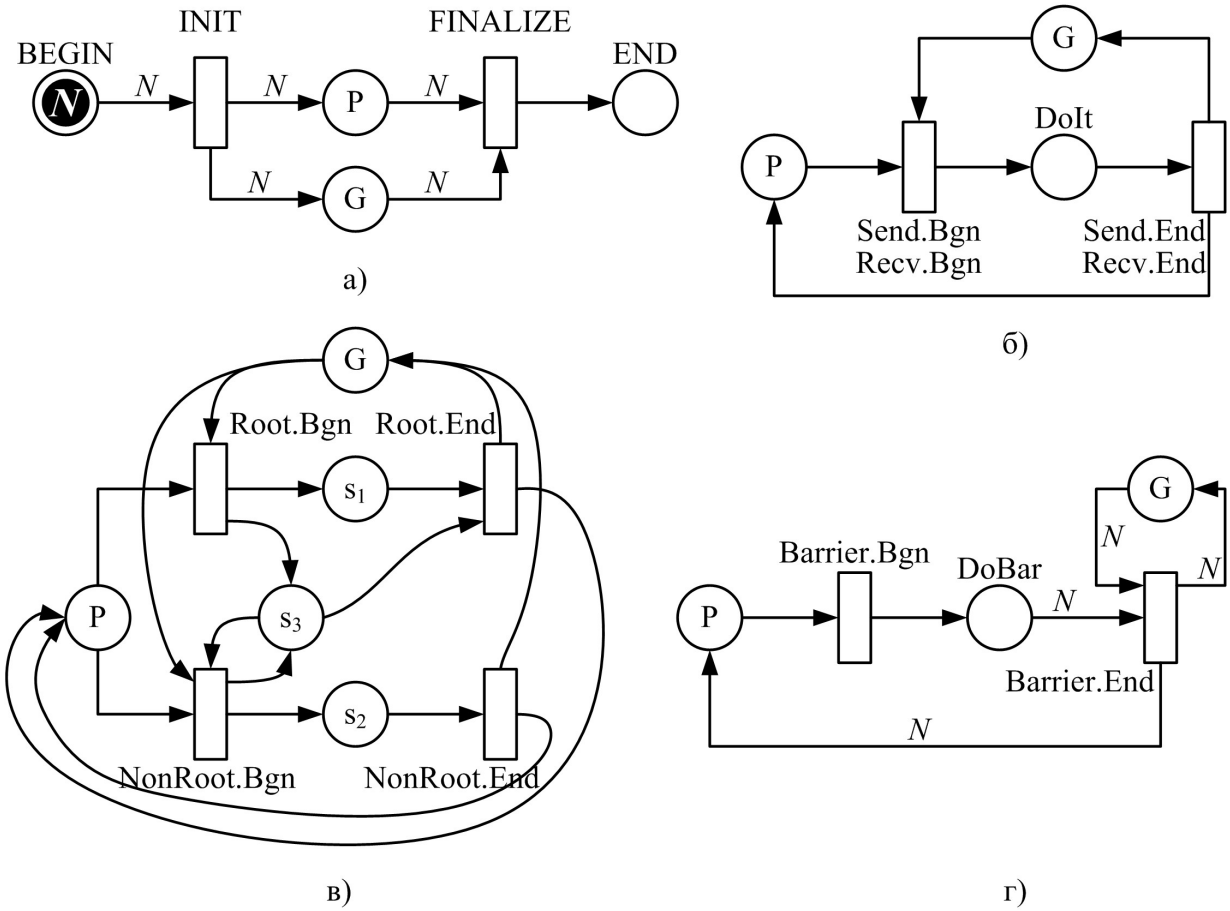

Рис. 1. Модель MPI.

Получение реальных действий из программы достигается дополнением исходного кода программы отладочными функциями [2]. Данный способ модификации широко используется во многих системах визуализации состояния параллельной программы. Роль отладочных функций состоит в накоплении и сохранении некоторой служебной информации, которая идентифицирует выполненное в программе действие, а также место, где она было получена. Каждый процесс генерирует собственную отладочную информацию, и для модели MPI она может быть двух видов. Первый относится к тому, что исполнение процесса программы достигло определенной точки. В модели это эквивалентно нахождению фишки в определенном месте. Второй относится к тому, что программа выполняет некоторый MPI-вызов. В модели это эквивалентно срабатыванию соответствующего перехода. Идентификация мест и переходов в отладочной информации происходит по их именам. Таким образом, поток информации из программы приводит к функционированию модели и изменению текущего состояния среды MPI. Определим правила функционирования модели в соответствии с информацией, получаемой из программы.

1) переход считается возбужденным, если из программы поступили имена всех входных мест данного перехода.
2) шаг переходов считается возбужденным, если каждый переход шага удовлетворяет первому правилу.
3) возбужденный переход в модели может сработать, если из программы получена информация с названием этого перехода.
4) невозбужденный переход не может сработать, даже если из программы получена информация с названием этого перехода.

В соответствии с этими правилами проверяется допустимость функционирования переходов. Если в результате обработки всей отладочной информации обнаружено, что все переходы срабатывают по правилам 2) и 3), то такая история может использоваться повторно для воспроизведения исполнения программы.

Описанный способ связи исходного представления программы и модели MPI реализован в виде инструментального средства, которое состоит из трех компонент (см. рис. 2): модуль модификации, сервер отладки и редактор сетей Петри. Модуль модификации содержит реализацию отладочных функций. В настоящее время разработано четыре основные функции: PN_DBG_Init – инициализирует модуль, создает вспомогательные буферы хранения отладочной информации, устанавливает связь с сервером отладки, должна вызываться в начале программы до MPI_Init, PN_DBG_Finalize – корректно завершает работу модуля, PN_DBG_Place — записывает в буфер название места, PN_DBG_Transition — записывает в буфер название перехода. При заполнении буфера он передается на сервер отладки. Сервер отладки является диспетчером сбора отладочной информации и запускается до начала работы программы. Редактор сетей Петри связан с сервером отладки, от которого получает всю историю накопленной отладочной информацию. Данная история запоминается, анализируется, графически отображается на модели и по завершению работы параллельной программы может использоваться для воспроизведения ее исполнения. В последнем случае поток отладочной информации идет от модели в программу. При этом функции модуля модификации сравнивают локальную и полученную информации. Если она совпадает, то такая отладочная функция завершается и управление передается обратно в программу.

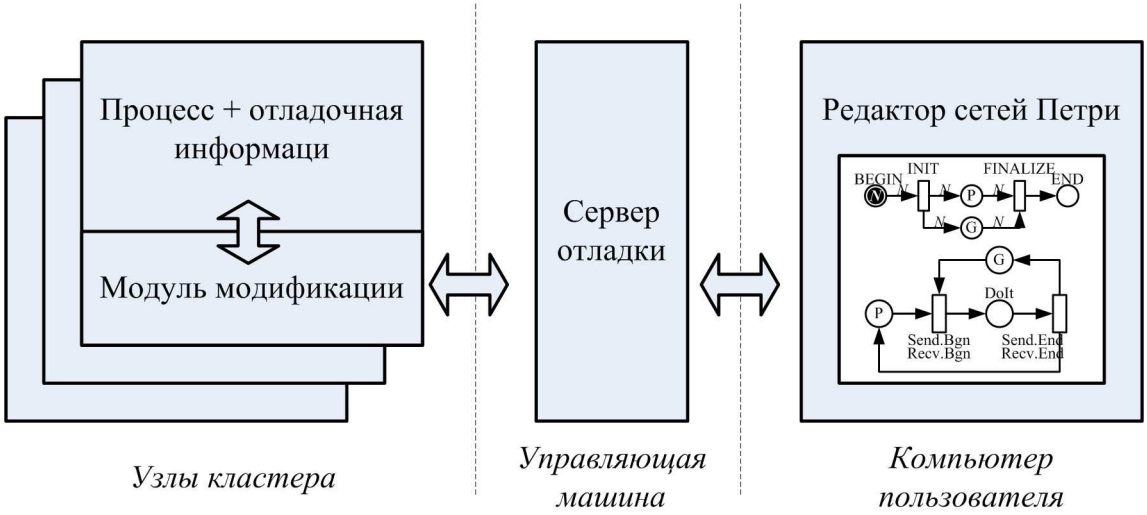

Рис. 2. Архитектура инструментального средства.

Таким образом, схема работы разработанного инструментального средства может быть как односторонней, то есть процессы параллельной программы передают информацию в редактор, так и двухсторонней — редактор может влиять на исполнение программы. Цикл отладки с использованием редактора сетей Петри выглядит следующим образом. Первый раз программа запускается в одностороннем режиме и записывается последовательность отладочной информации со всех процессов. Накопленная история анализируется на предмет соответствия модели — правила 1-4. Второй раз параллельная программа запускается в режиме управления по истории. В результате такого цикла удается воспроизвести исполнение программы и добиться многократного повторного ее исполнения.

ЛИТЕРАТУРА:

1. Г.В. Тарасов, Д.И. Харитонов. Построение системы визуализации параллельных программ на основе сетей Петри // Информационных технологии, №7, 2008. с. 35-41.
2. В.Е. Котов. Сети Петри — М.: Наука. 1984 — 160 с.