ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
2014 Управление, вычислительная техника и информатика № 1 (26)

ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

УДК 004.4'233

К.Е. Афанасьев, А.Ю. Власенко

АВТОМАТИЗИРОВАННЫЙ АНАЛИЗ КОРРЕКТНОСТИ MPI-ПРОГРАММ НА ОСНОВЕ ОПРЕДЕЛЕННЫХ ПОЛЬЗОВАТЕЛЕМ ШАБЛОНОВ ОШИБОЧНОГО ПОВЕДЕНИЯ

Описан подход к построению системы отладки, использующей автоматизированный анализ корректности MPI-приложений. Особенностью системы является дополнение стандартного механизма работы инструментальных средств этого класса возможностью пользователя определять ошибочные ситуации, на наличие которых необходимо проверить программу. Приведены элементы разработанного языка описания шаблонов ошибочного поведения, архитектура и алгоритм работы предлагаемой системы отладки.

Ключевые слова: логические ошибки; анализ корректности; шаблон ошибочного поведения; MPI-программа.

MPI (Message Passing Interface) до настоящего момента остается стандартом де-факто при создании программ для вычислительных систем с распределенной памятью. Однако отладка MPI-программ является крайне трудоемким процессом. Существует несколько подходов к построению инструментальных средств обнаружения ошибок в MPI-программах [1; 2. С. 13–15]. Один из перспективных методов отладки – автоматизированный контроль корректности. В противоположность ему диалоговая отладка – наиболее развитый в настоящее время подход – предполагает наблюдение за значениями переменных в каждом из процессов параллельной программы (характерными представителями этой категории средств являются TotalView, Distributed Debugging Tool и интегрированный в Microsoft Visual Studio отладчик mpishim [3. С. 3–7]). Современная тенденция к росту масштабов суперкомпьютеров, а вслед за ними – к увеличению числа ветвей запускаемых параллельных приложений, приводит к тому, что работа с системами диалоговой отладки становится крайне кропотливым процессом.

В большинстве систем автоматизированного контроля корректности MPI-программ используется профилировочная библиотека, основанная на стандартном PMPI-интерфейсе [4. С. 63; 5. С. 465–466]. Внутри функции-обработчика из данной библиотеки производится некоторый анализ вызова, и управление передается соответствующей PMPI-функции, которая выполняет действия, ожидаемые пользователем. Все MPI-реализации предоставляют PMPI-интерфейс, благодаря чему становится возможным создание такой библиотеки.

Проблема использования подхода автоматизированного контроля корректности заключается в следующем. Даже версия стандарта MPI 1.1 насчитывала более 120 процедур и функций. Последующие версии (MPI 2, MPI 3) привнесли еще несколько десятков функций. У подавляющего большинства функций имеются аргументы, на многие из которых MPI-стандарт [6] накладывает определенные ограничения. Для каждого процесса это могут быть как ограничения общего характера (например, «число пересылаемых элементов не может быть отрицательным»), так и зависимые от операций, выполняемых другим процессом (например, номер root-процесса коллективной операции должен совпадать на всех процессах коммуникатора). Кроме того, MPI-стандарт накладывает некоторые ограничения на последовательность вызовов функций. В связи с этим даже правиль-

ность всех аргументов вызываемых функций не гарантирует корректность MPI-программы в целом. Как следствие, пользователю нужно соблюдать огромное количество условий при работе с MPI-функциями.

Из вышеизложенного можно сделать два вывода: 1) разработка систем, автоматизирующих процесс отладки MPI-программ, абсолютно необходима; 2) заложить в такую систему проверки на все возможные ошибки практически невозможно. Сложность организации в системе автоматизированного контроля корректности проверок на все возможные ошибки MPI-программы – следствие не только очень большого их количества, но и того, что поддерживать такую систему в актуальном состоянии чрезвычайно сложно, поскольку MPI-стандарт регулярно претерпевает обновления. В то же время пользователю далеко не всегда требуется выполнение системой всех возможных проверок на его программе – зачастую он представляет, в каких местах может возникнуть ошибка, а излишние проверки создают дополнительную нагрузку на вычислительную систему.

Таким образом, актуальной является задача разработки системы автоматизированного контроля корректности, предоставляющей пользователю возможность самому задавать проверки, которые должна произвести система при исполнении программы. При этом подразумевается не выбор подмножества из имеющихся вариантов, а самостоятельное определение пользователем такого поведения программы, которое должно быть выявлено системой отладки. Плюсом такой системы является также то, что возможно не только обнаруживать логические ошибки, но и осуществлять пользовательские проверки.

1. Язык описания шаблонов

Для определения пользователем ошибок, на наличие которых требуется проверить программу, необходимо было разработать некоторый язык описания. Поскольку разрабатываемая система использует подход автоматизированного контроля корректности, то она получает информацию о вызываемых MPI-функциях при помощи профилировочного интерфейса. Следовательно, все, чем может оперировать система, – это имена вызываемых функций, их аргументы и номера MPI-процессов. Эти параметры и легли в основу языка. Для описания ситуации пользователь должен указать:

– процессы, участвующие в данной ситуации;

– функции, вызываемые (или, наоборот, – не вызванные) каждым из процессов;

– условия на значения, принимаемые аргументами функций.

При этом отождествлять описываемую ситуацию с логической ошибкой будет не совсем верно, потому что указанная ситуация может возникнуть вследствие нескольких ошибок. Например, пользователь указывает, что один процесс должен вызвать функцию отправки типа точка-точка, а во втором должен отсутствовать соответствующий прием во время работы программы. Такая ситуация может возникнуть из-за того, что в программном коде, выполняемом вторым процессом, либо вообще отсутствует функция приема от первого, либо, не дойдя до нее, второй процесс завис в некоторой другой операции. Эти причины являются двумя различными ошибками, повлекшими одну и ту же ситуацию. Поэтому ситуации, описываемые при помощи данного языка, авторы назвали «шаблонами поведения», а сам язык – «языком описания шаблонов».

Ниже приведены элементы языка описания шаблонов. Каждый шаблон пользователь заполняет в отдельном файле, имеющем следующую стандартную структуру:

1. Название шаблона (параметр *«Name»*).

2. Ключевое слово *«1block»*, говорящее о том, что далее следует блок описания процессов.

3. Количество (параметр *«N»*) и, опционально, номера процессов (*«p1»*, *«p2»*, …). При этом можно указать, что в ситуации участвуют все процессы, применив обозначение *«n(MPI_COMM_WORLD)»*.

Пример 1: *N=2 p1=0 p2=1* (два процесса с номерами *«0»* и *«1»* в коммуникаторе *MPI_COMM_WORLD*).

4. Ключевое слово *«2block»*, после которого следуют описания функций.

5. MPI-функции, которые должны вызвать указанные процессы для возникновения ситуации. Каждая функция обозначается как «*Fi*», где *i* – номер функции. Указывается обозначенный в предыдущем разделе номер процесса (*pj*) и через двоеточие – имя MPI-функции без префикса *«MPI_»*.

Пример 2: *F1=p1:Bsend F2=p2:Recv*

Для обозначения любой функции из некоторого множества существуют специальные макросы. Так, запись «*F1=p1:Send_any*» обозначает, что первый процесс вызывает любую из блокирующих операций отправки. Существуют также макросы *«(I)PTP*» – любая (не)блокирующая операция типа точка-точка, «*Coll*» – любая коллективная коммуникация, *«(I)Recv_any*» – любая (не)блокирующая операция приема.

Пример 3: *F1=p1:Send_any F2=p2:Recv_any*

Для указания того, что во время работы процесса должен отсутствовать вызов определенной функции, введен символ отрицания (!). Другими логическими операциями языка являются конъюнкция (&&) и дизъюнкция (||). Так, если для возникновения ситуации процесс может вызвать блокирующую или неблокирующую операцию отправки, то применяется следующая запись.

Пример 4: *F1=p1:Send_any||Isend_any*

Введена также специальная комбинация «*Fi=pi:*», говорящая о том, что функцию должны вызвать все процессы, участвующие в описываемой ситуации.

6. Ключевое слово «3block», обозначающее начало блока функций.

7. Поскольку в предыдущем разделе все участвующие в ситуации функции были введены при помощи символа «*F*» с номером, то в этом блоке обращение к ним идет посредством данных обозначений. Общее правило составления условий на аргументы состоит в том, что пользователь указывает функцию, номер ее аргумента и значение, которое должен принимать этот аргумент. При этом в качестве значения можно употреблять конкретное число: «*F1(2)=7*»; номер процесса: «*F1(4)=p1*»; значение какого-либо аргумента другой функции: «*F1(2)!=F2(2)*» или макрос, определенный в стандарте MPI: «*F1(4)=MPI_ANY_SOURCE*». Условия базируются на логических отношениях равенства «=», неравенства «!=», больше «>» и меньше «<».

Шаблон ошибочного поведения отличается от традиционного понятия логической ошибки еще и тем, что может быть более или менее детализирован. Примером детализации шаблонов является указание разного количества условий на аргументы функций.

Ниже рассмотрен пример текста файла шаблона ошибочного поведения, с которым может работать создаваемая система автоматизированного контроля корректности. Описываемая шаблоном ошибка «*Повторное использование активного запроса обмена*» заключается в том, что некоторый процесс вызывает неблокирующую функцию типа точка-точка, а затем, не освобождая объект «запрос обмена» (request) при помощи функций *MPI_Wait* и *MPI_Test*, вызывает еще одну неблокирующую функцию, использующую тот же самый «запрос обмена». Здесь необходимо отметить, что существуют и иные функции для освобождения «запроса обмена». Но если исследователь не использует в своей программе эти функции, то прописывать в шаблоне все возможные варианты нет необходимости.

```
Name=Repeated using of active request
1block
N=1
2block
F1=p1:IPTP
F2=p1:Wait! && p1:Test!
F3=p1:IPTP
3block
F1(7)=F2(1)
F1(7)=F3(7)
```

Третий блок шаблона говорит о том, что параметр «запрос обмена» должен быть одинаковым в обеих коммуникациях и операции ожидания / тесте на завершение.

2. Алгоритм работы системы

Порядок взаимодействия пользователя и компонентов системы представлен на диаграмме последовательности (рис. 1).

I. Действия пользователя.

1. Заполнение конфигурационного файла. Содержание данного файла:

a. Имена файлов исходного кода MPI-программы.

b. Общее число MPI-процессов (далее по тексту – «*n*»).

c. (опционально) Имена вычислительных узлов кластера.

d. (опционально) Порт для взаимодействия сервера-анализатора и MPI-процессов.

e. Полные пути к компилятору, препроцессору, профилировочной библиотеке.

f. Рабочая директория проекта на головном узле кластера, подключаемая к остальным узлам по CIFS.

Единожды заполнив конфигурационный файл для некоторого кластера, в последующем пользователю необходимо будет менять не много параметров. Кроме этого, некоторые параметры могут быть опущены, и система возьмет их значения по умолчанию.

2. Формирование файлов шаблонов по приведенным выше правилам.

3. Загрузка утилиты запуска.

II. Работа утилиты запуска. Данная утилита предназначена для совершения ряда вспомогательных действий.

1. Считывание значений из конфигурационного файла.

2. Вызов препроцессора, которому в качестве параметров передаются пути к файлам исходного кода MPI-программы. В каждую строку файлов, содержащую вызов MPI-функции, добавляется оператор, присваивающий глобальным переменным имя текущего файла и номер текущей строки кода [7. С. 60].

3. Вызов компилятора для преобразованных файлов исходного кода.

4. Вызов линковщика, связывающего полученный на предыдущем шаге объектный файл со статическими библиотеками.

5. Удаленный запуск процесса сервера-анализатора на указанном в конфигурационном файле хосте посредством вызова утилиты «clusrun», входящей в системные команды Microsoft Windows Server 2008 HPC Edition (в текущее время разработка ведется для этой системы).

6. Запуск MPI-программы (*n* процессов) при помощи вызова «mpiexec» на указанных в конфигурационном файле вычислительных узлах.

Таким образом, после того как отработала утилита запуска, в действие вступают MPI-процессы пользовательской программы и сервер-анализатор.

III. Работа сервера-анализатора.

1. Считывание файлов шаблонов и составление двух динамических списков (см. рис. 2, элементы 1, 2). Один из списков содержит информацию о ситуациях, возникновение которых необходимо проверить. Содержание этого списка соответствует данным, внесенным в файлы шаблонов. Второй список предназначен для рассылки MPI-процессам (рис. 2, элемент 2). Каждый его элемент содержит информацию о функции, фигурирующей в каких-либо файлах шаблонов. Для каждой функции вносится информация о номерах MPI-процессов, которые, согласно первому блоку шаблона, должны ее вызывать. Если номер процесса пользователем в шаблоне не указывается, то в это поле списка записывается «*ANY*». Если при обработке некоторого шаблона для определенной операции были записаны конкретные номера MPI-процессов, а в одном из последующих, где фигурировала данная операция, номера процессов указаны не были, то этим же макросом заменяются записанные ранее значения.

2. Передача MPI-процессам информации, внесенной во второй динамический список. При этом если в поле списка, отвечающего за номера процессов, значится «*ANY*», то имя соответствующей функции и номера ее аргументов рассылаются всем процессам. Если нет, то эти данные передаются только указанным процессам.

3. Затем сервер-анализатор порождает столько потоков (нитей), сколько имеется процессорных ядер на данном вычислительном узле. Потоки по алгоритму «round robin» принимают асинхронные сообщения от MPI-процессов, содержащие параметры вызываемых MPI-функций, и производят обработку.

Рис. 1. Диаграмма последовательности для системы отладки

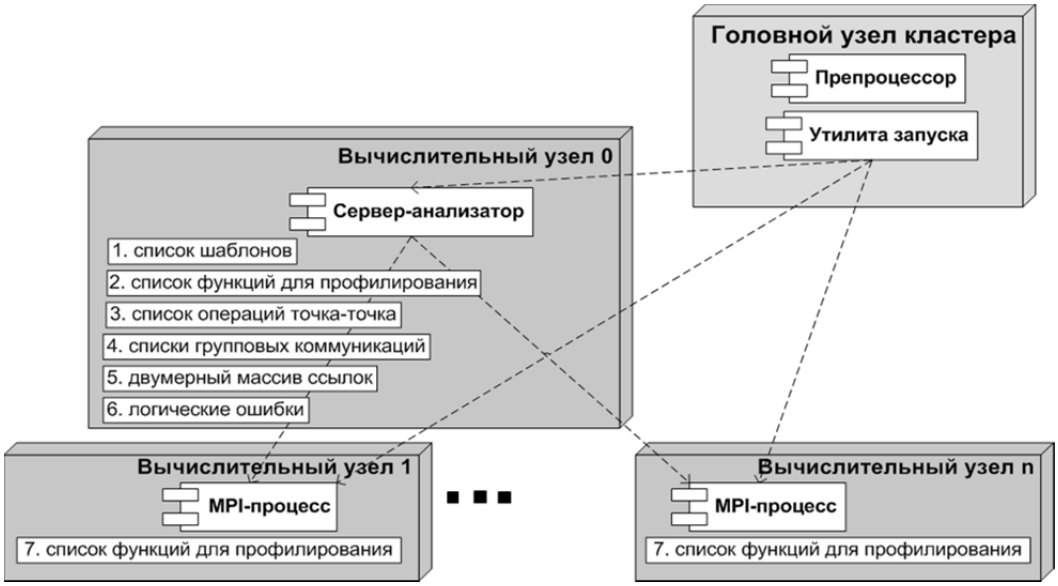

Рис. 2. Архитектура системы отладки

4. После того как каждый из параллельных процессов отправил серверу сообщение об окончании работы или о «зависании» в некоторой функции, сервер производит финальные проверки, записывает все обнаруженные ситуации в файл найденных ошибок и помещает его в рабочую директорию проекта. Данный файл является итогом работы системы и предназначен для анализа пользователем.

IV. Работа MPI-процессов.

1. Первой MPI-функцией в программе должна быть MPI_Init [8. С. 278]. В этой функции происходит прием от сервера списка функций, которые необходимо профилировать (рис. 2, элемент 7), и их аргументов.

2. При вызове любой последующей MPI-процедуры происходит обращение к принятому от сервера списку функций. Если данная функция присутствует в списке, то профилировочная библиотека собирает значения аргументов, указанных в списке, помещает их в текстовую строку вместе с номером строки исходного кода и номером процесса и отсылает серверу-анализатору.

3. Используемые структуры данных

Архитектура системы изображена на рис. 2. В виде стилизованных компонентов UML, размещенных на узлах вычислительного кластера, изображены исполняемые файлы, составляющие систему (препроцессор, сервер-анализатор и др.). В нумерованных прямоугольниках находятся классы, списки и другие ключевые структуры данных.

После приема сервером-анализатором от MPI-процесса профилировочной информации о некоторой функции сервер помещает эту информацию во внутренние структуры данных. Для коммуникаций типа точка-точка это динамический список (см. рис. 2, элемент 3), каждый элемент которого содержит имя операции; номер строки исходного кода, где она была вызвана; количество передаваемых элементов и др. Некоторые поля списка для элементов могут не заполняться, если ни в одном из шаблонов не значатся такие аргументы (например, тип пересылаемых данных и их количество).

В отличие от операций точка-точка, имеющих почти идентичный набор аргументов и в которых любая функция отправки может быть сопоставлена с любой функцией приема, у каждой из коллективных операций свой набор параметров. Поэтому информация обо всех вызовах операций точка-точка хранится в одном списке, а для каждой групповой функции на сервере-анализаторе ведется свой динамический список (рис. 2, элемент 4). При вызове коллективной функции в некотором MPI-процессе, по аналогии с операциями точка-точка, в данном списке добавляется элемент.

На сервере-анализаторе для каждого шаблона ведется двумерный массив $\{F_{ij}\}$ ссылок (рис. 2, элемент 5) на соответствующие элементы списка вызовов данной функции, значение которого разъясняется ниже.

4. Выявление ошибочных ситуаций

При приеме профилировочной информации о некоторой функции сервер обращается к динамическому списку функций (см. рис. 2, элемент 2), по которому определяются шаблоны, где участвует эта функция. Для каждого шаблона проводится следующий анализ.

Если во 2-м блоке шаблона указаны функции F_1, …, F_p, то для этого шаблона инициализируется двумерный массив ссылок с количеством столбцов, равным p. Каждая строка массива соответствует группе $\{F_{11}, F_{12}, \ldots, F_{1p}\}$, $\{F_{21}, F_{22}, \ldots, F_{2p}\}$, где F_{ij} – ссылка на элемент списка функции F_j, вызванной в i-й раз. Каждая проверка для шаблона содержит анализ вызовов функций, входящих в одну группу.

Для примера рассмотрим шаблон *«Неосвобождение производного типа данных»:*

F1=p1 :Type_contiguous
F2=p1:Type_free!
3block
F1(3)=F2(1)

Сервер-анализатор, принимая от MPI-процесса информацию о вызове MPI_Type_contiguous, добавляет в список этой функции элемент с параметрами вызова, создает в двумерном массиве ссылок новую строку и в первый элемент строки записывает ссылку на созданный элемент списка. При

вызове MPI_Type_free, удовлетворяющего условию *«F1(3)=F2(1)»*, параметры вызова также записываются в новый элемент списка данной функции, а в строку двумерного массива, куда внесена ссылка на соответствующий вызов функции MPI_Type_contiguous, – ссылка на этот элемент списка. Поскольку для срабатывания шаблона необходимо, чтобы вызов MPI_Type_free для нового типа данных отсутствовал, то такие строки массива F_{ij} не попадают в итоговый файл ошибок. Если по ходу работы MPI-процесса отсутствовал подходящий вызов MPI_Type_free, то после завершения всех процессов сервер-анализатор обнаружит данную ошибочную ситуацию во время финальных проверок. При этом просматриваются все не полностью заполненные строки двумерного массива ссылок и в файл ошибок записывается подробная информация о тех функциях, которые были вызваны, и перечислены функции, вызов которых отсутствует.

В некоторых случаях функция F_j, вызванная в i-й раз, может быть включена не в i-ю группу (одним из примеров этого являются функции MPI_Wait и MPI_Test из приведенного выше шаблона *«Повторное использование активного запроса обмена»*). Причиной таких исключений из общего правила является то, что в третьем блоке описания шаблона на F_j могут быть наложены ограничения, которым не удовлетворяет данный вызов функции.

Если для шаблона не указано соответствие номеров MPI-процессов p_1, p_2, … номерам в коммуникаторе MPI_COMM_WORLD, то перед каждым добавлением вызова функции в некоторую группу производится попытка занумеровать процессы таким образом, чтобы выполнялись условия из третьего блока описания шаблона, имеющие отношение к данной функции.

Система отладки завершает свою работу после завершения всех MPI-процессов и потоков сервера-анализатора, связанных с данными процессами. Сервер считает MPI-процесс завершенным, если поступило сообщение о вызове MPI_Finalize или в течение 30 секунд он не получает от MPI-процесса ни одного сообщения. В последнем случае окончание работы MPI-процесса происходит ненормально, и перед тем как завершить свою работу сервер записывает в файл найденных ошибок информацию о таких процессах.

После того как MPI-приложение завершено, сервер записывает в файл обнаруженные шаблонные ситуации. Для каждой шаблонной ситуации, выявленной системой в программе пользователя, в файл ошибок заносится следующее:

– название шаблона;

– номер конкретной ситуации, удовлетворяющей шаблону, если таких ситуаций в программе пользователя несколько;

– информация обо всех функциях, участвующих в ситуации:

- номер строки в файле исходного кода, где была вызвана функция;
- имя функции;
- номер MPI-процесса, сделавшего данный вызов;
- параметры данной функции.

Файл ошибок помещается в сетевую директорию на головном узле кластера.

Заключение

Разработанный авторами язык описания шаблонов ошибочного поведения MPI-программ позволяет прикладным исследователям определять ситуации, на наличие которых необходимо проверить параллельное приложение. Язык базируется на тех данных, которые возможно получить при помощи профилировочного интерфейса MPI-реализации. Шаблоны, записанные на данном языке, не зависят от конкретной MPI-программы и могут быть применены повторно для анализа другого параллельного приложения.

Поскольку разрабатываемая система отладки использует подход автоматизированного контроля корректности, то ограничениями в ее применении являются логические ошибки, обусловленные только некорректным MPI-программированием. В шаблонах пользователь может оперировать именами и аргументами только MPI-функций. Однако наиболее сложными для обнаружения ошибками являются именно те, которые возникают из-за неверного использования инструментария распа-

раллеливания. К настоящему моменту в системе отладки реализована обработка MPI-функций типа точка-точка и проведена апробация на ряде тестовых примеров.

В планах на будущее – разделение работы по обнаружению логических ошибок между сервером-анализатором и MPI-процессами, что позволит снизить накладные расходы со стороны системы на программу пользователя во время исполнения.

Ценность применения определенных пользователем шаблонов поведения при создании средств автоматизированного контроля корректности заключается в следующем. Во-первых, система легко сможет применяться при выходе новых версий MPI-стандарта (алгоритм обработки вызовов MPI-функций общий и останется неизменным для новых процедур). Во-вторых, шаблоны дают возможность задавать пользовательские проверки для параллельной программы, не являющиеся ошибками в общем случае.

Тестирование и отладка создаваемой системы проводится на вычислительных ресурсах центра коллективного пользования «Высокопроизводительные параллельные вычисления» (icp.kemsu.ru/CKP).

ЛИТЕРАТУРА

1. *Ефимкин К.Н., Жукова О.Ф., Крюков В.А.* Средства отладки MPI-программ. Анализатор корректности. URL: http://www.keldysh.ru/papers/2006/prep28/prep2006_28.html. 2006 (дата обращения: 12.10.2013).
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 КемГУ. 2009. Вып. 2. С. 13–20.
3. *Гергель В.П.* Введение в методы параллельного программирования. Лабораторная работа 3. Отладка параллельных MPI-программ в среде Microsoft Visual Studio 2005. URL: http://www.software.unn.ac.ru/ccam/mskurs/LAB/RUS/PPT/CCS_Lab03.pdf (дата обращения: 9.10.2013).
4. *Hilbrich T., Schulz M., de Supinski B.R.* MUST: A Scalable Approach to Runtime Error Detection in MPI Programs // Tools for High Performance Computing. Berlin : Springer, 2009. P. 53–66.
5. *Krammer B., Mueller M., Resch M.* MPI application development using the analysis tool MARMOT // Lecture Notes in Computer Science. Berlin : Springer, 2004. V. 3038. P. 464–471.
6. *The MPI* Standard. URL: http://www.mcs.anl.gov/research/projects/mpi/standard.html (дата обращения: 9.10.2013).
7. *Власенко А.Ю.* Модель масштабируемой системы автоматического контроля корректности параллельных программ // Вестник НГУ. 2009. Т. 7, вып. 4. С. 53–66.
8. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб. : БХВ-Петербург, 2002. 608 с.

Афанасьев Константин Евгеньевич
Власенко Андрей Юрьевич. E-mail: vlasenko@kemsu.ru
Кемеровский государственный университет

Поступила в редакцию 16 октября 2013 г.

Afanasiev Konstantin E., Vlasenko Andrey Y. (Kemerovo State University, Russian Federation).

Automated correctness analysis of the MPI-programs based on the templates of erroneous behavior defined by an user.

Keywords: logical errors; correctness analysis; template of erroneous behavior; MPI-program.

The article describes an enhancement of automated correctness control approach to debugging of the MPI-programs by possibility to define user's templates of erroneous behavior. New debugging tool is suggested, that use the MPI profiling interface for gathering information about callings of the MPI functions and carry out an analysis of presence of situations described in templates. For composing these templates new language is introduced, which composition is given in the article. Each of the templates is the text file, composed of three blocks: the block with description of processes that participate in situation; the functions that are caused by processes; the conditions on arguments of functions. For the logical grouping of functions, special signs are introduced. The templates composed by this language may be used for analysis of the different MPI-programs, and during one work session of debugging system, the amount of using templates is unlimited.

Before the launching of the debugging system, an user should fulfill the configuration file with the names of the initial code files; the parameters of launching; the paths to the compiler, the profiling library, the directory with templates. After the calling of the launching utility with the configuration file as a parameter, this utility calls the preprocessor that adds some of the operators to the initial code to get an additional information about the program. Then, the compiler is calling that links the program with the profiling library, and the obtained executable file is launched on the nodes of the HPC-cluster. The "server-analyzer" is launched on a node. This server gets from the MPI-processes information about the calling of communication functions and performs analysis on correspondence to situations, described in templates.

The advantage of templates is that we have the possibility to perform user's checks and make the specification of template situations, the presence of which is necessary to check in automated mode.

REFERENCES

1. Efimkin K.N., Zhukova O.F., Kryukov V.A. Sredstva otladki MPI-programm. Analizator korrektnosti. URL: http://www.keldysh.ru/papers/2006/prep28/prep2006_28.html. 2006 (data obrashcheniya: 12.10.2013). [Efimkin K.N., Zhukova O.F., Krukov V.A. MPI program debug tools. Analyzer of the correctness. Preprint, Institute of Applied Mathematics, the Russian Academy of Science, (2006). URL: http://www.keldysh.ru/papers/2006/prep28/prep2006_28.html.]
2. Afanas'ev K.E., Vlasenko A.Yu. Semanticheskie oshibki v parallel'nykh programmakh dlya sistem s raspredelennoy pamyat'yu i metody ikh obnaruzheniya sovremennymi sredstvami otladki. *Vestnik KemGU.* 2009. Vyp. 2. P. 13–20. [Afanasiev K.E., Vlasenko A.Yu. Semantic errors in parallel programs for systems with distributed memory and methods of their detection by modern debugging tools. *Bulletin of Kemerovo State University.* Kemerovo. V. 2. P. 13-20. (2009).]
3. Gergel' V.P. Vvedenie v metody parallel'nogo programmirovaniya. Laboratornaya rabota 3. Otladka parallel'nykh MPI-programm v srede Microsoft Visual Studio 2005. URL: http://www.software.unn.ac.ru/ccam/mskurs/LAB/RUS/PPT/CCS_Lab03.pdf (data obrashcheniya: 9.10.2013). [Gergel V.P. Introduction to methods of parallel programming. Laboratory work 3: Debugging of parallel MPI-programs in Microsoft Visual Studio 2005 environment. URL: http://www.software.unn.ac.ru/ccam/mskurs/LAB/RUS/PPT/CCS_Lab03.pdf]
4. Hilbrich T., Schulz M., de Supinski B. R. MUST: A Scalable Approach to Runtime Error Detection in MPI Programs. *Tools for High Performance Computing.* Springer, Berlin. P. 53-66. (2009).
5. Krammer B., Mueller M., Resch M. MPI application development using the analysis tool MARMOT. *Lecture Notes in Computer Science.* V. 3038. Springer, Berlin. P. 464–471. (2004).
6. The MPI Standard. URL: http://www.mcs.anl.gov/research/projects/mpi/standard.html
7. Vlasenko A.Yu. Model' masshtabiruemoy sistemy avtomaticheskogo kontrolya korrektnosti parallel'nykh program. *Vestnik NGU.* 2009. V. 7, no. 4. P. 53–66. [Vlasenko A. Yu. Automated correctness control of parallel programs scalable system's model. *Bulletin of Novosibirsk State University.* V. 7. No. 4. P. 53-66. (2009).]
8. Voevodin V.V., Voevodin Vl.V. Parallel'nye vychisleniya. SPb. : BKhV-Peterburg, 2002. 608 p. [Voevodin V.V., Voevodin Vl.V. Parallel computations. Saint-Petersburg: BHV. (2002).]