Архитектура и принципы работы масштабируемого инструментального средства динамического обнаружения семантических ошибок в MPI-программах

А.Ю. Власенко

Кемеровский государственный университет

Введение

Из практики разработки программного обеспечения хорошо известно, что примерно 2/3 времени, затрачиваемого на создание системы, приходится на отладку [3]. В случае разработки параллельных программ ситуация усложняется тем, что одновременно выполняется несколько экземпляров приложения (то есть процессов) при программировании для вычислительной системы с распределенной памятью, или несколько потоков при программировании для системы, обладающей общей памятью. В настоящей работе описываются проблемы в программах, использующих интерфейс MPI [1], поэтому ниже будут рассматриваться исключительно процессы. По ходу работы процессы обмениваются посылками-сообщениями, что является потенциальным источником дополнительных логических ошибок по сравнению с последовательными программами. Среди данных ошибок можно выделить несколько подклассов [2]:

- некорректное использование интерфейса MPI (отсутствие или неверный порядок вызовов необходимых функций MPI-стандарта, неверное управление внутренними объектами MPI-реализации, несоответствия в параметрах или длине сообщения в операциях посылки/приема);
- ошибки соревнования (конкуренция за доступ к области памяти между основной программой и параллельно работающей MPI-функцией, недерминизм в порядке приема при посылке сообщений от нескольких отправителей одному получателю, вызвавшему функцию приема «от всех отправителей» и некоторые другие случаи);
- взаимные блокировки процессов или дедлоки (при вызове функции посылки/получения сообщения двумя процессами друг другу/друг от друга, при вызове коллективной коммуникации частью процессов группы).

Таким образом, отладка MPI-приложений зачастую является крайне сложным и трудоемким процессом. Для помощи прикладному программисту к настоящему моменту создано несколько программных систем, которые можно классифицировать по используемым подходам к обнаружению логических ошибок:

- диалоговые отладчики (TotalView, Distributed Debugging Tool);
- средства верификации модели программы (MPI-Spin);
- средства сравнительной отладки (Distributed Virtual Machine);
- средства автоматического анализа корректности (Intel Trace Analyzer and Collector, MARMOT).

Среди остальных подход автоматического анализа корректности выделяется тем, что посредством применения эвристических алгоритмов позволяет обнаруживать ошибки, обусловленные недетерминированным поведением взаимодействующих процессов (ошибки соревнования, потенциальные дедлоки) в отличие от диалоговых отладчиков; избавлен от проблемы комбинаторного взрыва при увеличении числа процессов, которой обладают системы верификации программы на модели; не требует предварительной разработки корректно работающей версии программы, которую можно принять за эталонную в противоположность подходу сравнительной отладки.

Существующие в настоящее время системы MARMOT [4] и Intel Trace Analyzer and Collector используют довольно грубые алгоритмы выявления семантических ошибок. Дедлоки в данных средствах обнаруживаются по тайм-ауту; о потенциальной воз-

можности возникновения ошибок соревнования MARMOT сообщает при наличии в программе функций приема с макросом «от любого процессора» без проведения какого-либо анализа на то, оправданно ли использование этого макроса или нет; системы не могут обнаруживать ошибки несоответствия в аргументах парных функций передачи/приема при применении производных типов данных и т.д.

Но главной слабой стороной каждого из указанных инструментов является архитектура. В системе MARMOT анализом корректности занимается выделенный сервер отладки, на который остальные узлы кластера передают данные о вызванных функциях и параметрах вызовов. Если параллельная программа с интенсивными коммуникациями запущена на большом количестве процессоров, то сервер отладки может стать узким местом и оказывать существенное негативное влияние на производительность параллельного приложения. ITAC к каждому запущенному MPI-процессу на кластере добавляет отдельный поток, который производит анализ вызываемых MPI-функций на наличие локальных ошибок (ошибок, возникающих в пределах одного процесса), а для обнаружения глобальных (для выявления которых требуется информация от нескольких MPI-процессов) обменивается данными с дополнительными потоками на других вычислительных узлах. Поскольку современные процессоры обладают несколькими ядрами и в будущем их число будет возрастать, то создание нового потока не будет играть решающую роль для производительности выполняющегося приложения. В связи с этим данная архитектура хорошо подходит для поиска локальных ошибок, но для анализа на глобальные ошибки необходимо большое количество дополнительных коммуникаций, поскольку проверки не могут осуществляться централизованно. Чем больше вызовов MPI-функций в программе пользователя, тем больше дополнительных посылок выполняет ITAC, что может дать значительную нагрузку на сетевой сегмент.

Модель масштабируемой системы динамического анализа корректности параллельных программ

Таким образом, в настоящее время стоит задача разработки системы отладки, наименее влияющей на производительность запущенного MPI-приложения. Предлагаемое в данной работе средство объединяет преимущества MARMOT и ITAC. Его архитектура представлена на рис.1. Узлы кластера, выделенные для запуска MPI-программы, делятся на узлы-рабочие, где выполняется программа пользователя, и узлы-анализаторы, которые, принимая данные о параметрах вызываемых в программе MPI-функций, отыскивают глобальные логические ошибки посредством аналитических алгоритмов. Для разделения работы на несколько узлов-анализаторов, разные вычислительные узлы посылают параметры MPI-функций разным узлам-анализаторам. В случае операций точка-точка каждой паре из множества неупорядоченных пар *{x, y},* где *0<=x<N, 0<=y<N*, *x≠y,* а *N* – количество вычислительных узлов, ставится в соответствие свой узел-анализатор по следующему правилу: {0, 1} – 1; {0, 2} – 2; …; {0, P} – P; {0, P+1} – 1; {0, N-1} – K; {1, 2} – K+1;… Здесь предполагается, что P – количество узлов-анализаторов, которое выбрал пользователь, их нумерация начинается с единицы и 1<=K<=N. При вызове процессом *a* функции посылки сообщения процессу *b*, соответствующему паре *{a, b}* узлу-анализатору отправляются время вызова и параметры функции. Когда процесс *b* вызывает парную операцию получения сообщения, то, естественно, аргументы функции отправляются тому же анализатору, и, таким образом, информация, необходимая для обнаружения семантических ошибок в паре коммуникаций, будет находиться на одном узле.

На каждом анализаторе запускаются несколько потоков, между которыми разделяется работа по поиску ошибок. В качестве примера такого разделения нагрузки можно привести следующую ситуацию. Для хранения информации о вызовах MPI-функций типа точка-точка в программе пользователя на узлах-анализаторах находятся списки,

каждый элемент которых содержит свой номер; номер парного узла; номера процессов отправителя и получателя сообщения; сокращенное имя операции; число посылаемых/принимаемых элементов; тип пересылаемых данных; тэг сообщения и некоторые другие служебные поля. Каждый раз, когда от узла-рабочего приходят данные о параметрах вновь вызванной функции точка-точка, в списке создается новый элемент и его поля инициализируются принятыми значениями. Далее просматриваются предыдущие элементы списка и отыскиваются те из них, которые соответствуют парным коммуникациям. Парным для элемента, соответствующего функции отправки сообщения считается элемент, где в поле названия операции записано rcv или irc (то есть элемент был создан в результате вызова MPI_Recv или MPI_Irecv); в поле процесса-отправителя записано то же значение, что и в поле процесса-получателя вновь добавленного узла списка или константа MPI_ANY_SOURCE; поле тэга содержит такое же значение или константу MPI_ANY_TAG. В вычислительно емкой задаче может быть большое количество коммуникаций, и, следовательно, список сообщений на узлах-анализаторах может разрастаться до значительных размеров. А поскольку при каждом новом добавлении элемента для поиска парного узла списка нужно сравнивать несколько параметров, то данная процедура может создать немалую нагрузку на узел-анализатор. В связи с этим рационально использовать все имеющиеся ядра процессора. Работа по поиску парного узла легко может быть распараллелена на несколько потоков. Далее, найдя парные узлы, становится возможным провести анализ на наличие ошибок несоответствия в аргументах, ошибок соревнования (например, когда один процесс вызвал функцию приема сообщения с константой MPI_ANY_SOURCE, а 2 других – функцию отправки первому), дедлоков (когда 2 процесса одновременно посылают или принимают друг от друга данные).

Поиском локальных ошибок занимаются служебные потоки, порождаемые в пределах MPI-процессов. Так, при вызове неблокирующих функций точка-точка дополнительный поток просматривает список неосвобожденных «запросов обмена» и ищет элемент, идентичный используемому в обрабатываемой операции; в списке используемых областей памяти ищется пересечение буфера памяти в анализируемой операции с буферами происходящих в то же время неблокирующих коммуникаций и т.д. Кроме того, обмен служебными сообщениями с узлами-анализаторами осуществляется также посредством служебных потоков. По возможности служебный поток занимается передачей данных по сети в случае, если основной MPI-поток не находится в это время в вызове коммуникационной функции, чтобы не создавать излишнюю нагрузку на сетевой сегмент во время работы параллельного приложения. Таким образом, работа основного MPI-потока, выполняющего программу пользователя и проверка на семантические ошибки в вызываемых коммуникационных функциях происходят одновременно, используя многоядерность современных процессоров.

Отладка приложений, использующих параллелизм по процессам и созданных при помощи коммуникационного интерфейса MPI, является чрезвычайно трудоемким процессом, который усложнен большим множеством новых логических ошибок, не свойственных последовательным программам. Для поддержки процесса отладки наиболее полезными могут оказаться системы автоматического контроля корректности параллельных программ. Известные на сегодняшний день инструментальные средства этой группы обладают несовершенной архитектурой, что может быть следствием резкого снижения производительности параллельной программы с интенсивными передачами сообщений при увеличении количества используемых вычислительных узлов. Кроме того, нельзя считать оптимальными методы обнаружения данными средствами нескольких классов семантических ошибок.

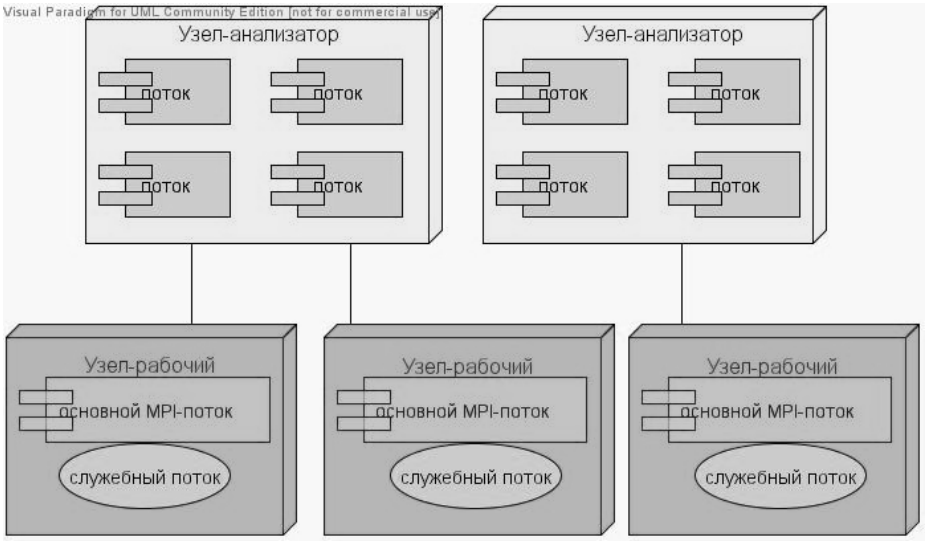

Рисунок 1. Компонентная модель системы.

Заключение

В данной работе представлена архитектура и принципы действия системы автоматического контроля корректности, минимально влияющей на быстродействие запущенной на вычислительном кластере параллельной программы. Данное качество обеспечивается тем, что локальные ошибки обнаруживаются служебным потоком, работающем на свободном ядре многоядерного процессора каждого из узлов вычислительного кластера, а для анализа на наличие глобальных ошибок при запуске MPI-приложения пользователь может зарезервировать несколько узлов кластера. Выделение некоторого количества узлов исключительно для целей отладки не стоит считать недостатком системы, поскольку современные кластерные системы состоят из нескольких сотен или даже тысяч узлов, в то время как для счета редко используется больше нескольких десятков. Для нахождения глобальных ошибок служебные потоки передают данные о параметрах и времени вызова узлам-анализаторам в то время, когда вычислительная сеть не загружена коммуникациями между основными MPI-потоками программы. Благодаря перечисленным особенностям функционирования системы, ожидается что она будет обладать потенциально неограниченной масштабируемостью, не влиять на скорость обмена данными между узлами кластера по сети и минимально замедлять вычислительную работу пользовательской программы.

Литература

1. Афанасьев К. Е., Стуколов С. В. Многопроцессорные вычислительные системы и параллельное программирование. Кемерово: Кузбассвузиздат, 2003. - 233 с.
2. Власенко А. Ю., Демидов А. В. Отладка MPI-программ на персональной ЭВМ // Материалы Всероссийской научной конференции «Инновационные недра Кузбасса. IT-технологии». Кемерово: ООО «ИНТ», 2008. с. 236-241.
3. Эдмунд М. Кларк, Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. М.: издательство московского центра непрерывного математического образования, 2002. – 416 с.
4. Krammer B., Mueller M., Resch M. MPI Application Development Using the Analysis Tool MARMOT // Lecture Notes in Computer Science. Vol. 3038. P. 464 – 471. Springer Berlin, 2004.