Автоматизированная отладка параллельных программ с использованием комбинации статического и динамического подходов*

А.Ю. Власенко, А.М. Гудов

Кемеровский Государственный Университет

В статье рассматривается проблема автоматизированной отладки параллельных приложений и различные подходы к ее решению. Описана авторская система автоматизированного контроля корректности MPI-программ, обнаруживающая ошибки во время работы приложения. Отличительной чертой системы является применение текстовых шаблонов ошибочного поведения, благодаря которым пользователь может гибко управлять проверками, выполняемыми над работающей MPI-программой. Обсуждается внедрение в систему отладки статического анализа исходного кода, позволяющего обнаружить ряд дополнительных ошибок. Таким образом, предлагается комбинированный подход автоматизированной отладки MPI-приложений, сочетающий элементы статического и динамического анализа, где выполняемые инструментальным средством проверки прозрачны для пользователя.

Ключевые слова: MPI, отладка параллельных программ, статический анализ, автоматизированный контроль корректности, шаблон ошибочного поведения.

1. Введение

Отладка параллельных программ продолжает оставаться существенной проблемой для прикладных исследователей. В настоящее время на рынке преобладают диалоговые отладчики (TotalView, DDT), которые не очень удобны при поиске ошибок в параллельных приложениях по ряду причин. Во-первых, при запуске множества процессов крайне сложно отслеживать поведение и анализировать значения переменных, двигаясь по шагам в каждом из процессов. Во-вторых, диалоговые отладчики подразумевают интерактивную работу с приложением, что не всегда возможно в условиях суперкомпьютерного центра. В связи с этим нарастает потребность в инструментальных средствах автоматизированной отладки параллельных приложений. Такие средства можно разделить по используемым подходам.

Статический анализ представляет собой разбор исходного кода приложения, выполняемый до компиляции с целью обнаружения логических ошибок. Родоначальником данного подхода является анализатор Lint [1]. Большинство подобных систем предназначены для обнаружения ошибок, не связанных со спецификой параллельных программ (использование неинициализированных переменных, обращение к NULL-указателям и др.) Некоторые подобные проверки встраивают в современные компиляторы. Существуют такие системы, как PC-Lint [2] и VivaMP [3], предназначенные для поиска ошибок в многопоточных программах. Однако для работы с программами, использующими параллелизм по процессам (в том числе с MPI-программами), средства данной категории развиты недостаточно.

Подход *сравнительной отладки* основан на сопоставлении значений, принимаемых переменными, при исполнении эталонной и анализируемой программ. В качестве эталонной может выступать, в частности, последовательная версия приложения. Для сравнения двух программ можно сначала накопить трассы, фиксирующие выполненные операторы и значения переменных, а затем сравнивать эти трассы, либо сначала собрать трассу при исполнении эталонной программы, а затем динамически сравнивать с ней работу анализируемой версии. Сравнительную отладку можно отнести к автоматизированным методам лишь в том случае, если разработчик имеет уже отлаженную корректную версию программы. Данный подход был реализован в российской системе DVM [4] и отладчике Guard [5].

* Проект поддержан государственным заданием КемГУ № 2014/64.

Верификация программы на модели предполагает анализ инструментальным средством некоторой абстракции (модели) для исследуемой программы, составленной на определенном псевдоязыке (примером такого средства для отладки MPI-программ может служить MPI-Spin [6]). Данную модель разработчик составляет, зачастую, как систему состояний программы и переходов между этими состояниями, после чего специфицирует наиболее значительные свойства программы, и система отладки производит перебор всех возможных маршрутов в полученном графе, проверяя данные свойства. Таким образом, верификация программы на модели подходит для тех случаев, когда требуется проверить сохранение базовой логики приложения, и малоэффективна для обнаружения ошибок, связанных, например, с неверным использованием MPI-функций (несоответствия в типах отправляемых и принимаемых данных при коммуникациях точка-точка, неверное управление внутренними ресурсами MPI и др.)

Автоматизированный контроль корректности представляет собой проверку на соответствие программы определенным правилам. Данная проверка может выполняться либо во время работы программы (как в системе MUST [7]), либо после завершения программы по собранной трассе (одним из первых таких инструментов был Intel Message Checker [8]). В случае MPI-приложений, инструментальные средства, использующие данный метод, зачастую применяют профилировочный интерфейс MPI-реализации для сбора информации о вызываемых MPI-функциях. Затем средство отладки сверяет поведение параллельной программы с правилами MPI-стандарта. В связи с этим область применимости таких систем ограничивается поиском некорректного использования инструментария MPI.

Наиболее эффективной методикой при построении системы автоматизированной отладки представляется комбинация нескольких подходов, каждый из которых подходит для обнаружения логических ошибок различных типов. На текущий момент подобные комбинированные инструменты автоматизированной отладки, обладающие способностью обнаруживать достаточно широкий спектр разнообразных ошибок, слабо представлены на рынке.

2. Система автоматизированного контроля корректности MPI-программ на базе текстовых шаблонов ошибочного поведения

Авторами настоящей работы ранее была создана система автоматизированного контроля корректности MPI-приложений [9]. Главной целью, поставленной при построении данной системы, было предоставление пользователю возможности управлять проверками, которые осуществляет система по ходу работы параллельной программы. В других инструментальных средствах, использующих данный подход (MUST, библиотека libVTmc системы Intel Trace Analyzer and Collector) набор проверок жестко закреплен в коде системы, и пользователь может только отключать или включать их часть. Во-первых, это лишает систему гибкости в случае необходимости добавления новых проверок – систему требуется перекомпилировать, и выпускать новый релиз. Такая необходимость возникает с выходом новых версий стандарта MPI, который регулярно обновляется, в результате чего появляются новые функции, внутренние MPI-объекты и новые ограничения на их использование. А, во-вторых, у пользователя нет возможности добавить собственные проверки на определенное поведение параллельной программы.

В предлагаемой системе обозначенная проблема решается введением в схему работы механизма текстовых шаблонов ошибочного поведения. В этих шаблонах на введенном языке описываются ошибочные ситуации, подлежащие обнаружению системой во время работы параллельной программы. В тексте шаблона произвольно задается имя (Name); количество процессов, участвующих в ошибочной ситуации (K); функции, которые должны быть вызваны или, наоборот, не вызваны в процессах для возникновения описываемой ситуации (FUNCTIONS); условия, накладываемые на функции (CONDITIONS). На рисунке 1 в качестве примера приведен шаблон «Вызов функции отправки типа точка-точка процессом, парный для которого вызвал коллективную операцию».

```
Name=Call of point-to-point function by process, which pair has called col-
lective operation
PROCESSES
K=2
```

```
FUNCTIONS
F1=p1:Send||Ssend
F2=p2:!Recv && !Irecv
F3=p2:Coll_data||Coll_reduc
CONDITIONS
F1(4)=p2
F2(4)=p1
```

Рис. 1. Шаблон «Вызов функции отправки типа точка-точка процессом, парный для которого вызвал коллективную операцию»

В описываемой ситуации участвуют 2 процесса (K=2). При этом их номера в любом из коммуникаторов могут быть любыми, а в шаблоне они условно обозначаются как p1 и p2. Первый процесс вызывает операцию отправки (MPI_Send или MPI_Ssend), из которой он может не выйти до тех пор, пока процесс-получатель не вызовет операцию приема. В первом условии «F1(4)=p2» обозначается то, что процессом-получателем должен быть p2. Вторая строка в блоке функций говорит о том, что к моменту возникновения ситуации процесс p2 не должен вызвать операцию приема, а то, что прием должен быть именно от процесса p1 гарантирует второе условие. Строка «F3=…» содержит 2 введенных в языке макроса, обозначающих коллективные функции, перераспределяющие данные (MPI_Bcast, MPI_Gather, …) – Coll_data и редукционные операции (MPI_Reduce, MPI_Allreduce, …) – Coll_reduc. Если в параллельной программе возникнет такая ситуация, то с большой вероятностью, это приведет к зависанию.

Ниже кратко описан общий алгоритм работы системы по шагам.

1. Пользователь формирует конфигурационный файл, где указывает файлы исходного кода своей MPI-программы; количество запускаемых параллельных процессов; полные пути к компилятору, MPI-библиотеке, статической профилировочной библиотеке, директории с текстовыми шаблонами, рабочей директории проекта; может указать дополнительные параметры, такие как TCP-порт для взаимодействия сервера-анализатора с MPI-процессами, имя узла, на котором должен быть запущен сервер-анализатор, имя исполняемого файла и файла найденных в программе логических ошибок и др.
2. Пользователь может внести изменения в готовые шаблоны (например, добавить дополнительные условия) или создать свои собственные шаблоны в специальной директории.
3. Далее на управляющем узле кластера загружается утилита запуска, которая разбирает конфигурационный файл и производит дальнейшие действия по загрузке MPI-приложения и компонентов системы автоматизированного контроля корректности:
   a. Запуск предварительной компиляции пользовательской MPI-программы без связывания с профилировочной библиотекой. Если были выявлены ошибки, то в рабочую директорию проекта выгружается файл с выводом компилятора, и система завершает работу.
   b. Запуск компонента системы «препроцессор», который добавляет в исходный код операторы присваивания глобальным переменным номеров строк с вызовами MPI-функций и имен файлов исходного кода. Впоследствии эта информация потребуется для указания локализации возникшей ошибки.
   c. Вызов компилятора по отношению к файлам MPI-программы, преобразованным препроцессором, с прилинковкой профилировочной библиотеки.
   d. Удаленный запуск компонента «сервер-анализатор» на одном из узлов кластера с передачей ряда параметров, считываемых из конфигурационного файла.
   e. Загрузка откомпилированной MPI-программы на кластере.
4. Начав работу, сервер-анализатор производит парсинг шаблонов, размещенных пользователем в директории, заносит информацию в собственные структуры данных и высылает MPI-процессам данные, необходимые для проверок на возникновение ситуаций, описанных в шаблонах с количеством процессов (K) равным «1», а также данные о тех MPI-функциях и их параметрах, значения которых требуется передавать «серверу-анализатору» для проверки на глобальные ошибки.
5. Во время работы параллельной программы информация о параметрах вызываемых MPI-функций собирается при помощи профилировочного интерфейса, и MPI-процессы либо производят анализ на появление локальных ошибок, либо передают эту информацию серверу-анализатору.

6. Сервер-анализатор сверяет поведение параллельного приложения с шаблонами и делает вывод о возникновении шаблонной ситуации. Информация о таких соответствиях вносится в специальные структуры данных. По окончанию работы MPI-программы информация о найденных MPI-процессами локальных ошибках и сервером-анализатором глобальных ошибках выгружается в текстовый файл, предназначенный для анализа пользователем.

3. Статический анализ

Как отмечалось выше, подход автоматизированного контроля корректности в случае MPI-программ пригоден только для обнаружения некорректного использования коммуникационного стандарта и других ситуаций, связанных с вызовом MPI-функций. Причиной этого является единственный источник информации о работающей параллельной программе - профилировочный интерфейс MPI-реализации. Для расширения возможностей по отладке в разработанную систему решено было добавить функционал статического анализа.

В составе системы уже имеется компонент, обрабатывающий исходный код пользовательской программы - препроцессор. Именно на данный компонент будет возложена работа по статическому анализу. При этом статический анализ будет проводиться в целях обнаружения ошибок, специфичных именно для MPI-программ, выявление которых требует получения информации, недоступной для профилировочного интерфейса. Примеры таких ошибок:

- отсутствие необходимых вызовов (MPI_Init, MPI_Finalize, функции ожидания или тесты на завершение неблокирующей коммуникации);
- несоответствия в типах переменных и массивов, использованных в MPI-функциях с их объявлениями;
- попытка передать коммуникационной MPI-функции большее количество элементов массива, чем было объявлено (выход за границу массива).

Ситуации, которые будут обнаруживаться во время статического анализа, также можно формализовать в текстовых шаблонах, расширив язык описания несколькими элементами. Например, ошибка несоответствия типа объекта, отправляемого или принимаемого в функции точка-точка с его объявлением, может быть описана следующим образом.

```
Name=Inconsistency in point-to-point function object datatype with its definition
PROCESSES
K=1
FUNCTIONS
F1=p1:Send_any || Recv || Isend_any || Irecv
CONDITIONS
DEF( F1(1) ) != F1(3)
```

Рис. 2. Шаблон «Несоответствие типа объекта в функции точка-точка с его объявлением»

Первым аргументом в коммуникациях точка-точка является адрес передаваемой переменной или массива. Третьим аргументом указывается тип данных. Поэтому в блоке «FUNCTIONS» шаблона присутствует одна функция F1, в качестве которой может выступать любая блокирующая или неблокирующая операция отправки или приема. В блоке «CONDITIONS» дополнительно введенная операция «DEF» означает, что необходимо найти в коде объявление передаваемого объекта и выяснить его тип. Условие шаблона заключается в том, что для возникновения описываемой ситуации значение третьего аргумента функции не должно совпадать с выявленным типом. Например, если объявление массива выглядело как «int M[1000];», а вызов функции отправки – «MPI_Send(M, 1000, MPI_FLOAT, …)», то препроцессор системы отладки запишет информацию об обнаруженной ситуации в файл ошибок.

На рисунке 3 изображена диаграмма последовательности, отражающая общий алгоритм работы системы отладки после внедрения механизма статического анализа. Как видно из рисунка, даже в том случае, если препроцессор обнаружит соответствие исходного кода с некоторым шаблоном, и файл ошибок после этой фазы обработки пользовательской программы будет не пустой, система, тем не менее, продолжит свою работу. Также во время последующего динамического анализа система не завершает пользовательскую программу аварийно при первом

найденном соответствии с шаблоном. Такое поведение обусловлено тем, что ситуации, описываемые в шаблонах, не всегда являются критическими ошибками, из-за которых MPI-приложение однозначно будет выводить некорректные результаты. Например, пользовательский код, обладающий свойством, отраженным на рисунке 2, может быть вполне корректным, если при отправке MPI_Send используется производный тип данных, состоящий из нескольких фрагментов типа MPI_INT.

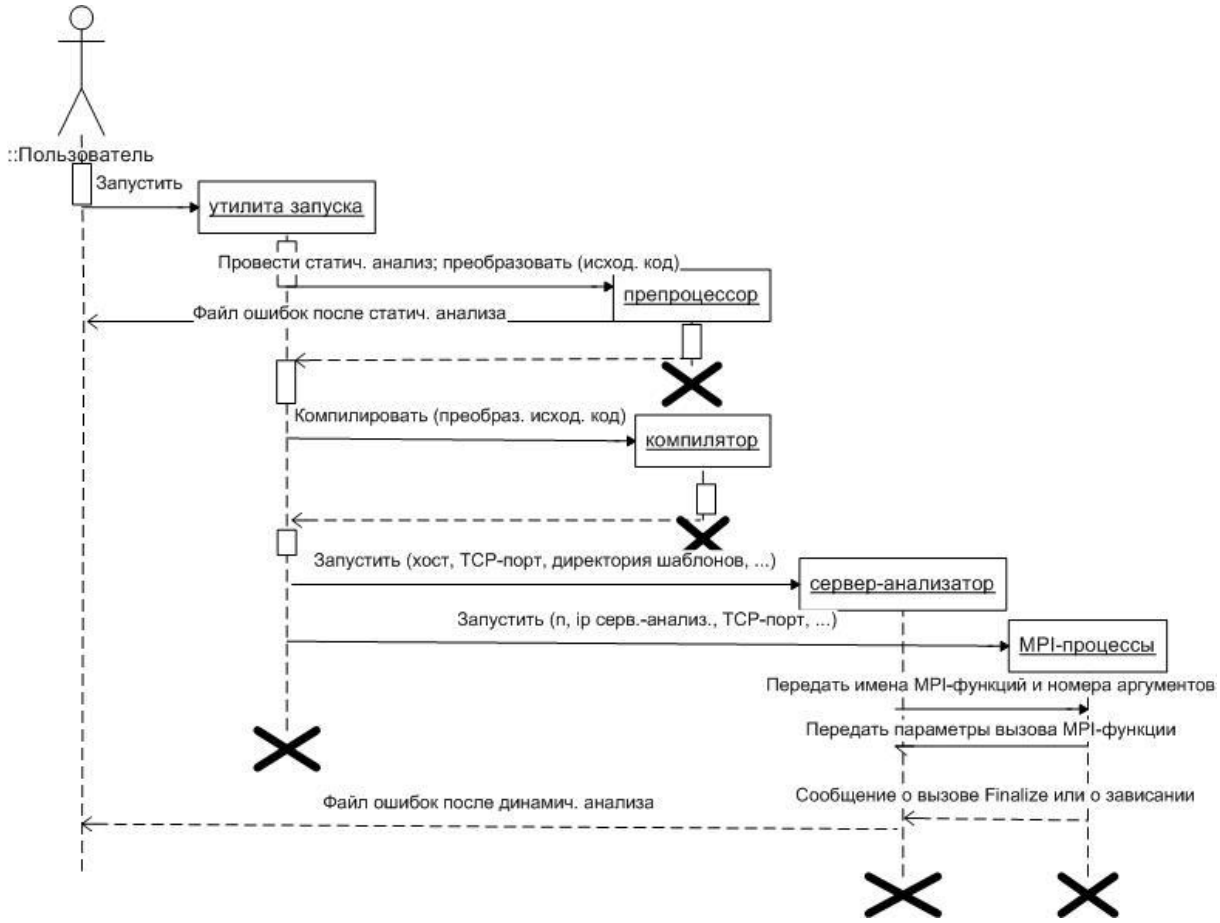

Рис. 3. Диаграмма последовательности для разрабатываемой системы

4. Заключение

Разработанная система отладки MPI-приложений в значительной степени может упростить трудоемкий процесс обнаружения логических ошибок, благодаря применению метода автоматизированного контроля корректности. По сравнению с другими инструментальными средствами, использующими тот же подход, система предоставляет б**о**льшую гибкость и прозрачность, поскольку все проверки формализованы в текстовых шаблонах ошибочного поведения, и пользователь легко может поправить какой-либо из шаблонов или создать собственный. Авторами разработана библиотека из 20 шаблонов для наиболее распространенных логических ошибок, возникающих вследствие некорректного использования MPI-стандарта первой версии, а также несколько шаблонов для ошибок, обусловленных неверным использованием функций и объектов, определенных во второй версии. Среди профилируемых функций – блокирующие и неблокирующие коммуникации точка-точка, коллективные операции, функции для работы с группами, коммуникаторами, производными типами данных и др.

Последующее внедрение статического анализа позволит существенно расширить возможности системы по выявлению некорректного поведения параллельного приложения, потому как исходный код программы будет еще одним источником информации о пользовательской программе, наряду с профилировочным интерфейсом MPI-реализации. Проверки, которые должны

выполняться во время статического анализа, планируется формализовать в текстовых шаблонах, что даст пользователю единый инструмент для настройки системы.

Система опробовалась в учебном процессе на математическом факультете КемГУ при проведении практических занятий по курсам «Параллельное программирование» и «Параллельные алгоритмы». Студенты с помощью системы выявляли ошибки в создаваемых программах. Также система использовалась при разработке учебного пособия [10], где рассматриваются способы распараллеливания некоторых вычислительных алгоритмов для классических задач линейной алгебры и программирования (умножение матриц и векторов, сортировка массивов действительных чисел, решение СЛАУ, параллельные методы на графах) при помощи MPI. Изложенные в пособии параллельные программы тестировались на разработанной системы.

Система установлена на вычислительном кластере центра коллективного пользования (ЦКП) научным оборудованием КемГУ (ckpno.kemsu.ru). Благодаря этому исследователи имеют возможность отлаживать свои MPI-приложения в автоматизированном режиме. Эффективность отладки в этом случае оценить сложно, можно сослаться лишь на то, что 62% пользователей ЦКП несколько раз использовали систему.

Литература

1. Johnson S. Lint, a C program checker. Computer Science Technical Report 65, Bell Laboratories, December 1977. 12 p.
2. Bezem J. How to wield PC Lint. URL: http://www.bezem.de/pdf/htwpl.pdf (дата обращения: 10.06.2016).
3. Колосов А.П. VivaMP, система выявления ошибок в коде параллельных программ на языке Си++ использующих OpenMP // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новогород, 30 марта – 3 апреля 2009 г.). Челябинск: Издательский центр ЮУрГУ, 2009.С. 535-541.
4. Алексахин В.А., Баринова В.О., Бахтин В.А. Средства отладки OPENMP-программ в DVM-системе // Материалы Всероссийской научной конференции «Научный сервис в сети Интернет: технология распределённых вычислений» (Новороссийск, 22 сентября - 27 сентября 2008г.). М.: Изд-во МГУ, 2008. С. 281-285.
5. Abramson D., Watson G., Dung L. Guard: A Tool for Migrating Scientific Applications to the .NET Framework. // Proceedings of the International Conference on Computational Science (ICCS 2002). Amsterdam, The Netherlands, April 21st 2002. P. 834 - 843.
6. Siegel S. Verifying Parallel Programs with MPI-Spin. Proceedings of the 14th European PVM/MPI Users' Group Meeting, Paris, France, September/October 2007. P.13-14.
7. Hilbrich T., Protze J., Schulz M. MPI Runtime Error Detection with MUST: Advances in Deadlock Detection. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. Salt Lake City,UT, USA, 2012. P. 1-10.
8. Desouza J., Kuhn B., Supinski B. Automated, scalable debugging of MPI programs with Intel Message Checker // Proceedings of the second international workshop on Software engineering for high performance computing system applications. St. Louis, Missouri. 2005. P. 78-82.
9. Афанасьев К.Е., Власенко А.Ю. Автоматизированный анализ корректности MPI-программ на основе определенных пользователем шаблонов ошибочного поведения // Вестник Томского Государственного Университета. Серия: Управление, вычислительная техника и информатика. №1 (26), 2014. С. 75-83.
10. Афанасьев К.Е., Григорьева И.В., Рейн Т.С. Основы высокопроизводительных вычислений. Т.3. Параллельные вычислительные алгоритмы: учебное пособие. Кемеровский государственный университет. – Кемерово, 2012. 185 с.

Automated debugging of parallel programs using the combination of static and dynamic approaches

A.Yu. Vlasenko, A.M. Goudov

Kemerovo State University

The article discusses problem of automated parallel programs debugging and different approaches for solving this problem. Author's system of MPI-programs automated correctness control is described. System performs the search of errors in running application. One of the main system features is using text templates of erroneous behavior. These templates give user flexibility to manage checks that debugging system performs on running MPI-program. Also paper discusses introduction to the system of source code static analysis that makes possible to discover a number of additional errors. Thus the composite approach of automated MPI-programs debugging is suggested, combining elements of static and dynamic analysis, where checks, performed by debugging tool, are transparent for user.

Keywords: MPI, parallel programs debugging, static analysis, automated correctness control, template of erroneous behavior.

References

1. Johnson S. Lint, a C program checker. Computer Science Technical Report 65, Bell Laboratories, December 1977. 12 p.
2. Bezem J. How to wield PC Lint. URL: http://www.bezem.de/pdf/htwpl.pdf (accessed: 10.06.2016).
3. Kolosov A.P. VivaMP, sistema vyavleniya oshibok v kode parallelnih program na yazike C++ ispolzuyushih OpenMP [VivaMP, system of errors detection in parallel programs code in C++, using OpenMP]. Parallelnye vychislitelnye tekhnologii (PaVT'2009): Trudy mezhdunarodnoj nauchnoj konferentsii (Nijniy Novgorod, 30 marta – 3 aprelya 2009) [Parallel Computational Technologies (PCT'2009): Proceedings of the International Scientific Conference (Nijniy Novgorod, Russia, March, 30 – April, 3, 2009)]. Chelyabinsk, Publishing of the South Ural State University, 2009. P. 535–541.
4. Alexahin V.A., Barinova V.O., Bahtin V.A. Sredstva otladki OPENMP-programm v DVM-sisteme [Debugging tools of OPENMP-programs in DVM-system]. Materiali Vserossiyskoy nauchnoj konferentsii "Nauchniy servis v seti Internet: tehnologii raspredelennih vichisleniy" (Novorossiysk, 22 sentyabrya - 27 sentyabrya 2008) [Proceedings of Russian Scientific Conference (Novorossiysk, Russia, September, 22 – September, 27, 2008)]. Moscow: Publishing of the Moscow State University, 2008. P. 281-285.
5. Abramson D., Watson G., Dung L. Guard: A Tool for Migrating Scientific Applications to the .NET Framework. // Proceedings of the International Conference on Computational Science (ICCS 2002). Amsterdam, The Netherlands, April 21st 2002. P. 834 - 843.
6. Siegel S. Verifying Parallel Programs with MPI-Spin. Proceedings of the 14th European PVM/MPI Users' Group Meeting, Paris, France, September/October 2007. P.13-14.
7. Hilbrich T., Protze J., Schulz M. MPI Runtime Error Detection with MUST: Advances in Deadlock Detection. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. Salt Lake City, UT, USA, 2012. P. 1-10.
8. Desouza J., Kuhn B., Supinski B. Automated, scalable debugging of MPI programs with Intel Message Checker // Proceedings of the second international workshop on Software engineering for high performance computing system applications. St. Louis, Missouri. 2005. P. 78-82.

9. Afanasiev K.E., Vlasenko A.Yu. Avtomatizirovanniy analiz korrektnosti MPI-programm na osnove opredelennih polzovatelem shablonov oshibochnogo povedeniya [Automated correctness analysis of MPI-programs based on user-defined templates of erroneous behaviour]. Vestnik Tomskogo Gosudarstvennogo Universiteta. Seriya: Upravlenie, vichislitelnaya tehnika i informatika [Bulletin of Tomsk State University. Series: Management, computing technics & informatics]. 2014. No. 1(26). P. 75–83.
10. Afanasiev K.E., Grigorieva I.V., Rein T.S. Osnovi visokoproizvoditelnih vichisleniy. T.3. Parallelnie vichislitelnie algoritmi: uchebnoe posobie [Basics of high performance computing. Vol.3. Parallel computing algorithms: tutorial] Kemerovo, Publishing of the Kemerovo State University, 2012. 185 p.