СИСТЕМА АВТОМАТИЗИРОВАННОГО ПОИСКА ОШИБОК И НЕЭФФЕКТИВНОСТЕЙ В ПАРАЛЛЕЛЬНЫХ ПРОГРАММАХ

Д.Ю. Андреев, А.С. Антонов, Вад.В. Воеводин, С.А. Жуматий, Д.А. Никитенко, К.С. Стефанов, П.А. Швец

1. Введение

В условиях стремительного увеличения вычислительных возможностей суперкомпьютерных систем и масштабов решаемых задач проблемы эффективности выполнения параллельных программ и эффективности работы самих вычислительных систем встают особенно остро.

Существует множество причин, влияющих на итоговый ход выполнения программы: выбор метода решения задачи и соответствующих алгоритмов, выбор и особенности технологии программирования, нюансы в самой реализации, специфика программно-аппаратной архитектуры и т. д. Существенным образом на итоговой эффективности работы сказываются так называемые логические и семантические ошибки [1]. Ошибки такого рода могут приводить как к серьезному снижению эффективности, так и к получению неверных результатов или зависанию программы. Вопрос автоматизированного поиска ошибок и неэффективностей такого рода является актуальным, так как это позволит повысить скорость (и сам факт) их обнаружения для основной массы пользователей, включающей в себя не только профессионалов, но и специалистов с малым опытом. Тем самым это повысит не только эффективность отладки приложений, но и эффективность функционирования всего суперкомпьютерного центра.

Известны различные подходы к поиску ошибок в параллельных программах и оценке эффективности выполнения задач на суперкомпьютерных системах, такие как инструментирование кода, профилировка, предполагающие в общем случае внесение различного рода изменений в исходный код программы или предоставление дополнительной информации о структуре программы. Это может являться весьма трудоемкой задачей как для специалистов, так и, прежде всего, для рядового пользователя. Отчасти поэтому такие подходы зачастую не удается автоматизировать.

Существенно более удобным для конечного пользователя в этом отношении является подход, основанный на апостериорном анализе данных системного мониторинга или собранных трасс событий. В данном подходе чрезвычайно важным и актуальным является вопрос правильного выявления и сопоставления возможных причин и признаков нестандартного поведения программы и самих потенциальных ошибок. Выбор правильного набора анализируемых метрик, частоты сбора данных, выделение зависимостей между ними, набора профилируемых функций или библиотек – вот лишь несколько непростых вопросов, которые необходимо тщательным образом изучить для корректного решения задачи анализа эффективности параллельной программы, выявления ошибок и определения шаблонов неэффективного поведения.

В рамках данной работы проведен анализ существующих подходов, основывающийся на данных системного мониторинга и профилирования. На его основании разработаны алгоритмы автоматизированного поиска логических и семантических ошибок, базирующиеся на соответствующих технологиях. Получена программная реализация инструментария для применения разработанных алгоритмов.

Обычно в параллельных программах, помимо синтаксических ошибок, обнаруживаемых транслятором, выделяют также семантические или логические ошибки [2]. Разделение ошибок на семантические и логические является достаточно тонким, иногда эти типы ошибок не различаются. Под логическими ошибками понимается нарушение логики программы, приводящее к снижению эффективности работы приложения или неверному результату, а под семантическими ошибками - смысловые ошибки в программе, не связанными с нарушением синтаксиса языка программирования. Мы разграничиваем два этих типа по способу их обнаружения. Логические ошибки определяем на основе анализа результатов профилирования, а семантические ошибки — на основе анализа данных системного мониторинга.

В работе рассматриваются также шаблоны неэффективного поведения параллельных программ в модели PGAS, рассмотренных в рамках языка UPC. В представленных шаблонах внимание сконцентрировано на неоптимальной работе с чужими разделами общей памяти, а также с коллективными операциями. И это неудивительно, поскольку именно данные проблемы во многих случаях приводят к существенному увеличению времени выполнения UPC-программ. Поиск и последующее устранение этих проблем должно привести к оптимизации приложений и, как следствие, более эффективному использованию оборудования.

2. Алгоритмы автоматизированного обнаружения логических ошибок в параллельных программах на MPI

Для реализации алгоритмов автоматизированного обнаружения логических ошибок в параллельных программах для вычислительных систем с распределенной памятью был выбран стандартный интерфейс для профилирования MPI-программ PMPI. С использованием этого средства становится возможным профилировать все вызовы функций MPI.

Рассмотрим методику сбора данных с использованием PMPI. Согласно стандарту, каждая функция MPI может быть вызвана как с префиксом MPI_, так и с префиксом PMPI_. Таким образом, можно создать , например, свою собственную функцию MPI_Send(), в которой будет вызываться функция PMPI_Send(), а также будут производиться некоторые действия по сохранению параметров и условий вызова исходной функции. После выполнения программы будет собрана трасса, анализируя которую, можно обнаружить многие логические ошибки в параллельных программах для вычислительных систем с распределенной памятью.

Для первой версии создаваемой системы автоматизированного поиска ошибок и неэффективностей в параллельных программах разработаны 17 алгоритмов автоматизированного обнаружения логических ошибок в программах на MPI.

Несколько алгоритмов обнаруживает слишком долгое время ожидания отдельных процессов на различных типах блокирующих операций. Это может означать как реальную ошибку в программе, приводящую, например, к тупиковой ситуации (дедлоку), так и несбалансированность, которая может свидетельствовать о неэффективности алгоритма распараллеливания в анализируемой программе.

Другой распространенной логической ошибкой являются гонки данных, когда несколько потоков исполнения пытаются обновить общие ресурсы. В этом случае при отсутствии необходимой синхронизации результат выполнения зависит от того, какой процесс обновит общий ресурс последним. Данный тип ошибок может возникать или не возникать в зависимости от внешних условий, и поэтому крайне сложно отлавливается. Применительно к MPI вероятность гонок данных возникает при использовании в функциях, реализующих пересылки, предопределенных констант MPI_ANY_SOURCE и MPI_ANY_TAG. В данном случае пользователю выдается предупреждение.

Еще ряд алгоритмов отлавливают ошибки, при которых пользователь в разных процессах указал не соответствующие друг другу параметры соответствующих функций. В зависимости от типа ошибки, такая операция либо не может начаться вообще, либо приведёт к неверному результату.

Наконец, еще одна группа отслеживаемых логических ошибок связана с теми ситуациями, когда часть процессов, вовлеченных в некоторую коллективную или двустороннюю операцию, не вызывает соответствующую функцию. В некоторых случаях это приводит к тупиковой ситуации (дедлоку), в других случаях может проявляться, например, в утечках памяти.

Рассмотренные группы не исчерпывают все множество логических ошибок, встречающихся в MPI-программах. Однако разрабатываемая система автоматизированного поиска ошибок и неэффективностей является легко расширяемой, дальнейшее ее развитие может быть связано с разработкой новых алгоритмов для более полного покрытия всех вариантов ошибок и неэффективностей.

3. Автоматизированное обнаружение семантических ошибок в параллельных программах

В качестве основного инструмента для автоматизированного обнаружения семантических ошибок в параллельных программах был выбран анализ данных системного мониторинга. Основным инструментом получения данных мониторинга в рамках предлагаемого подхода является дайджест задачи (Job Digest) [3].

Данные мониторинга представляются в виде значений, полученных в моменты времени $T_0, \ldots, T_N$. Конкретные моменты времени, в которые фиксируются значения, определяются системой мониторинга, которая поставляет данные, с таким расчетом, чтобы уменьшить поток данных путем более частой передачи изменяющихся значений.

Данные в момент времени V_{T_i} представляют собой значение соответствующего параметра системы мониторинга (загрузка процессора, объем переданных данных в единицу времени, количество кэш-промахов и т. п.), от которого взяты минимум, максимум или среднее по всем процессорам (вычислительным узлам), на которых выполняется задача. Мы будем обозначать CPU_{maxT_i} максимальную по всем процессорам загрузку процессора в момент времени T_i .

Усредненным значением максимальной загрузки процессора мы будем называть следующую величину:

$$CPU_{maxavg} = \frac{\sum_{T_i} CPU_{maxT_i}(T_i - T_{i-1})}{T_N - T_0}$$

Также можно проводить усреднение не по всему времени работы задачи, а по какому-то временному окну.

Введенные оценки позволяют обнаруживать некоторые семантические ошибки параллельных программ.

Дисбаланс загрузки процессоров. Взяв небольшое временное окно (порядка 10-30 с) и определив на нем максимальную и минимальную загрузку процессора, мы можем обнаруживать дисбаланс в распределении нагрузки между вычислительными узлами. Кажется разумным установить порог такого дисбаланса в 20% загрузки.

Заметим, что такой алгоритм определения дисбаланса не будет работать, если не все процессоры вычислительных узлов задействованы в работе задачи. Такие ситуации, в частности, могут происходить, если

число процессов задачи не кратно числу процессоров на вычислительном узле. Другая частая ситуация – когда процессы распределяются не на каждое процессорное ядро узла, а, например, на каждое второе ядро, чтобы на каждый работающий процесс приходилось больше оперативной памяти.

Использование большого количества коротких сообщений. Для обнаружения такой ошибки мы анализируем отношение максимального по всем узлам, на которых работает задача, объема переданных по системной сети данных к максимальному по всем узлам, на которых работает задача, количеству отправленных по системной сети пакетов. Если это отношение меньше, чем заданный порог, мы фиксируем наличие ошибки.

Кажется разумным установить значение этого порога в 800 байт, хотя для разных реализаций MPI и разных коммуникационных сред могут быть заданы и другие пороги.

Слишком большое число порожденных процессов или нитей. Если в процессе работы программы число одновременно активных процессов (нитей) больше числа имеющихся вычислительных ядер, то возникает ситуация, при которой некоторым процессам (нитям) не хватает ресурсов.

Для обнаружения такой ошибки мы используем параметр LoadAverage, который система мониторинга получает от ядра ОС вычислительного узла. Этот параметр представляет собой усредненное по некоторому интервалу (5 с, 5 мин и 15 мин) число процессов (нитей), готовое к выполнению на процессоре. Мы фиксируем наличие ошибки, если в течение 30 сек. максимальное значение LoadAverage по всем узлам, на которых выполнялась задача, превосходит число процессорных ядер вычислительного узла плюс 1. Увеличение порога, при котором фиксируется ошибка, объясняется тем, что на небольшой промежуток времени могут становиться активными служебные процессы ОС, процессы системы времени выполнения реализации MPI и т. п.

4. Шаблоны неэффективного поведения параллельных программ на языке UPC

Язык UPC (Unified Parallel C) [4] представляет собой расширение языка программирования Си, предназначенное для работы на многопроцессорных системах. Основным достоинством языка UPC является относительная простота создания параллельных программ, при этом эффективность реализации достаточно высока и зачастую сравнима с использованием технологии MPI [5].

Однако, несмотря на простоту, при создании UPC-программ достаточно просто совершить ряд стандартных действий, которые приводят к снижению эффективности программы. В рамках данной работы был выделен набор таких действий – шаблонов неэффективностей, а также разработаны методы их обнаружения. Это призвано помочь пользователю в оптимизации UPC-программ.

Было выделено 12 различных шаблонов неэффективного поведения, которые логически разбиты на 4 группы. Далее будет рассмотрена каждая из этих групп.

Взаимодействие с чужим разделом общей памяти. В языке программирования UPC используется общая память, которая разбита на логические разделы, и каждый раздел привязан к некоторому процессору. Обычно это соответствует и физическому распределению – свой раздел общей памяти реализован на локальной для данного процессора оперативной памяти, поэтому доступ к своему разделу общей памяти осуществляется быстрее, чем к чужим разделам общей памяти. Соответственно, для эффективной работы с памятью желательно по возможности избегать обращения в чужие разделы общей памяти.

Вследствие этого были выделены шаблоны, которые оценивают число, долю и объем обращений в чужой раздел общей памяти. При превышении установленного порога по любому из этих параметров пользователю должно выдаваться предупреждение о слишком активной работе с чужой памятью, поскольку это может приводить к снижению эффективности программы.

Блоки данных. В данной группе оценивается, какими блоками данных происходит работа с общей памятью, то есть, каков средний размер считываемых или записываемых данных в UPC-операциях. Если средний размер блока невелик, то возрастет доля накладных расходов, связанных со служебной работой с указателями на общую память. При этом в общем случае обращение по указателю в общую память является более дорогостоящей операцией по сравнению с обращением в локальную память [6], поэтому подобное поведение UPC-программы может приводить к достаточно серьезному снижению эффективности.

В данной группе выделен один шаблон, который оценивает средний размер блока при взаимодействии с общей памятью. Отметим, что, в отличие от предыдущих шаблонов, здесь рассматриваются обращения как в чужой, так и в свой раздел общей памяти.

Коллективные операции. Здесь рассматриваться шаблоны неэффективного поведения, возникающие из-за неоптимального применения коллективных операций. Основные причины подобного неэффективного поведения заключаются в расхождении времени выполнения или начала/завершения коллективной операции на разных потоках, что приводит к ожиданию и, как следствие, простою потоков. Данные параметры и учитываются в выделенных нами шаблонах.

Стоит отметить, что подобный простой может возникать не только из-за неоптимизированной реализации UPC-программы, но и из-за неправильных настроек реализации самого языка UPC или даже работы ОС.

Синхронизация. Далее идут шаблоны неэффективного поведения, которые возникают из-за использования лишних операций синхронизации. В качестве точки синхронизации может выступать барьерная синхронизация или коллективные операции с соответствующей выбранной опцией синхронизации.

Неэффективное поведение в данных шаблонах заключается в использовании любых двух указанных точек синхронизации, между которыми каждый поток выполняет работу только с локальными данными. В таком случае вторая синхронизация, скорее всего, не требуется (хотя в некоторых редких случаях все-таки может быть нужна).

Важно заметить, что все приведенные шаблоны сигнализируют о возможных причинах снижения эффективности программы, но не позволяют выявить корень возникновения данных проблем. Они сообщают пользователю, что эффективность выполнения его программы может снижаться по такой-то причине, но происходит ли это из-за особенностей алгоритма или его неоптимальной реализации, необходимо определять самому пользователю.

Сбор информации о поведении UPC-программ осуществляется в динамике с помощью трассировки, поскольку такой способ позволяет получить всю необходимую информацию о последовательности всех произошедших в программе событий. Основным средством для этого в языке UPC служит интерфейс GASP. Данный интерфейс был разработан для использования в любых языках программирования на основе модели PGAS, однако на данный момент его поддержка реализована только в языке UPC [7].

Инструмент выделения шаблонов состоит из двух компонент: программы для получения во время выполнения UPC-программы трассы с помощью GASP интерфейса, а также утилиты для последующего разбора и анализа полученных трасс с целью выделения шаблонов. Для каждого шаблона разработан и реализован алгоритм, который по значениям определенных GASP событий позволит определить, попадает ли исследуемая UPC-программа под этот шаблон или нет.

5. Структура системы автоматизированного поиска ошибок и неэффективностей в параллельных программах

Ошибки и неэффективности параллельной программы определяются на этапе анализа результатов трассировки программы и данных системного мониторинга, которые могут быть получены в процессе экспериментального запуска на целевом суперкомпьютере. Суперкомпьютеры зачастую различаются набором компиляторов и системой постановки задач. Для автоматизации поиска ошибок и неэффективностей в системе между пользователем и суперкомпьютером присутствует промежуточный узел — сервер, который содержит шаблоны использования каждого включенного в систему суперкомпьютера и логику поиска ошибок.

Пользователь обращается к серверу через специальный интерфейс, в то время как все взаимодействие с суперкомпьютером и анализ данных трассировки происходит на стороне сервера. Доступ к суперкомпьютеру осуществляется с использованием стандартных интерфейсов ssh и rsync, которые доступны на всех суперкомпьютерах с Unix-подобной операционной системой.

На каждом суперкомпьютере имеется специальная учетная запись, которая позволяет серверу системы пройти авторизацию и от имени этой учетной записи производить компиляцию и запуск задач. При компиляции происходит подмена стандартных функций передачи данных на функции, которые дополнительно производят запись параметров передачи данных в файл с трассой. В результате на сервер передаются файлы с записанной трассой.

6. Информационно-вычислительный портал системы

Для удобной работы с системой был разработан специализированный web-портал. В качестве базовой технологии разработки был взят пакет sinatra (http://sinatrarb.com), предоставляющий средства обработки запросов через протокол HTTP, а также базовые средства работы с базами данных.

В качестве СУБД была выбрана sqlite3, как наиболее простая в установке и обслуживании. В условиях низкой нагрузки на портал (не более десятка запросов в секунду) такая база данных не может стать узким местом. При необходимости замены СУБД, это осуществляется без изменения логики работы портала — необходимо только заменить параметры доступа к БД, а также перенести данные.

Разработанный портал поддерживает авторизацию и аутентификацию, поэтому разные пользователи могут работать независимо.

В своей сессии пользователь может выбрать суперкомпьютер, на котором он будет работать, загрузить на портал исходный текст программы на языках Си+MPI или UPC, откомпилировать его и запустить полученный исполняемый файл, указав нужное число вычислительных ядер. Результаты выполнения задачи загружаются в БД и доступны пользователю в любое время.

По окончании работы задачи проводится автоматизированный поиск ошибок и неэффективностей на основании разработанных алгоритмов, результаты этого анализа также сохраняются в базе данных и могут быть просмотрены пользователем.

Приведем пример типичного сеанса работы. Пользователь входит на сайт и вводит данные для аутентификации. После этого он попадает на страничку выбора суперкомпьютера и оттуда — на страничку со списком загруженных файлов и действий (рис. 1).

При необходимости пользователь закачивает новый файл с исходным текстом программы и компилирует его, выбрав компилятор и нажав кнопку «Compile».

Скомпилированный файл можно запустить на суперкомпьютере, выбрав число требуемых вычислительных ядер и нажав «Run MPI» или «Run UPC» соответственно для MPI- или UPC-программы.

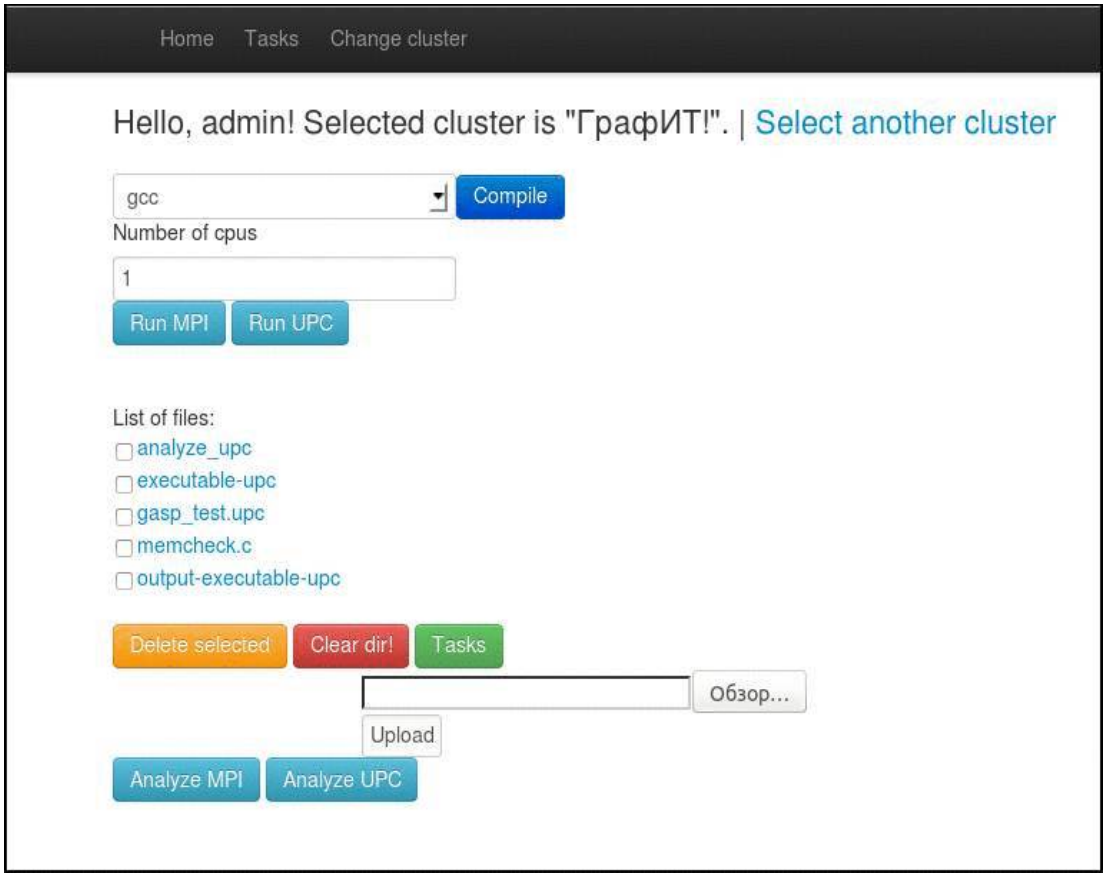

Рис. 1. Список файлов и действий

После этого будет создано задание, и браузер будет перенаправлен на страничку с его статусом. Когда статус изменится на «завершено», на страничке кроме общей информации о задании будет отображен вывод задачи и информация с анализом ее работы — рекомендации по оптимизации и замечания по характеру работы задачи (рис. 2).

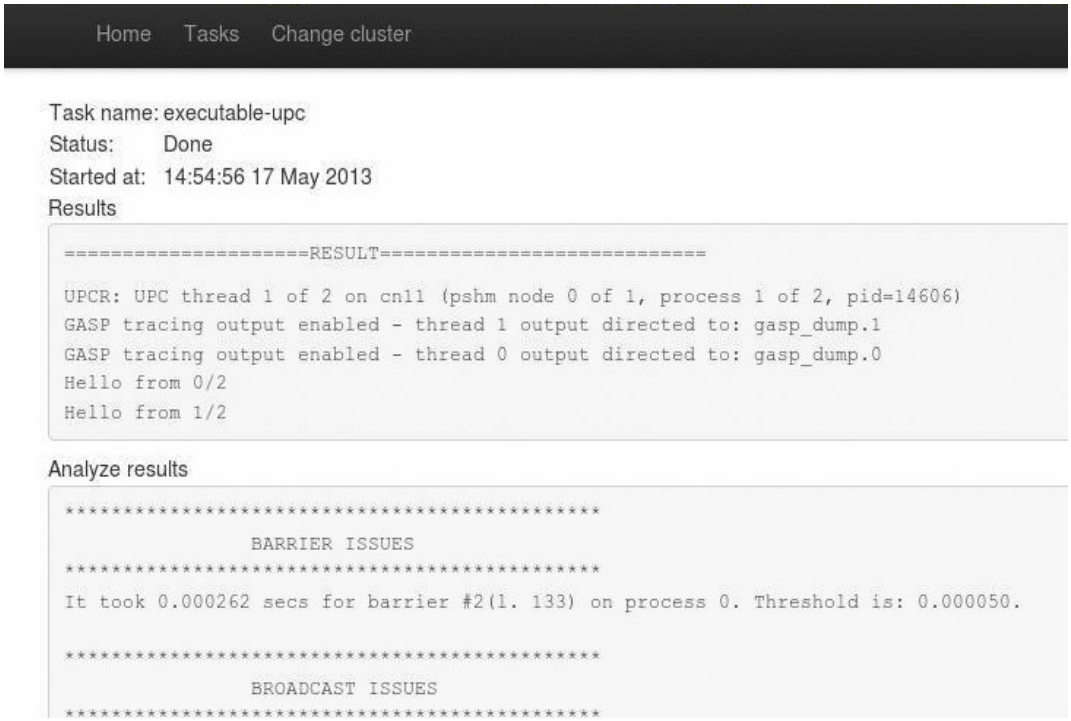

Рис.2. Пример результатов задачи и анализа

7. Примеры использования системы для оптимизации параллельных программ

Применение разработанных средств для оптимизации рассмотрим на следующем примере UPC-программы.

Было проведено исследование реализации операции Собеля для выделения границ в изображениях, которая предоставлена в качестве тестовой программы на сайте, посвященном языку UPC [8]. Данная программа была запущена на 18 процессорах (максимальное число доступных на тот момент ресурсов вычислительной системы). Анализ с помощью разработанных средств выделения шаблонов показал, что выполнение одной из барьерных синхронизаций заняло более 1 секунды, что при таком числе процессоров обычно свидетельствует о серьезной разбалансировке потоков. Далее, также с помощью разработанных средств было обнаружено, что несколько потоков обращаются в чужую память чаще остальных.

Это привело к появлению гипотезы о неравномерном распределении элементов общих массивов, что и подтвердилось при изучении исходного кода. Причина заключалась в выполнении цикла upc_forall(), где привязка (аффинити) задавалась с помощью общего массива edge с размером блока N/THREADS, где N=2k – размер массива. В таком случае при использовании 18 потоков на несколько первых потоков приходится большее число элементов, и, соответственно, большее число итераций цикла.

Решением этой проблемы может являться организация более равномерного распределения массива. Для этого программа была запущена с теми же параметрами, но на 16 процессорах. Это привело к снижению времени выполнения программы: 60 сек. против 54 сек. при N=4096.

Таким образом, полученная с помощью разработанных средств информация позволила сделать вывод о неоптимальном распределении данных, устранение которого привело к снижению времени выполнения, несмотря на использование меньшего числа процессоров.

8. Заключение

В статье рассмотрена разрабатываемая система автоматизированного поиска ошибок и неэффективностей в параллельных программах, кратко описаны алгоритмы, лежащие в основе производимого анализа, проиллюстрирована работа web-портала, реализующего пользовательский интерфейс к разрабатываемой системе.

Работа выполнена при финансовой поддержке Министерства образования и науки Российской Федерации, государственный контракт №14.514.11.4062.

ЛИТЕРАТУРА:

1. Антонов А.С., Воеводин Вад.В., Жуматий С.А., Никитенко Д.А., Стефанов К.С., Швец П.А. Автоматизация поиска ошибок и неэффективностей в параллельных программах// Вычислительные методы и программирование: Новые вычислительные технологии (Электронный научный журнал). 2013. Том 14, раздел 2ю С. 11-17.
2. К.Е. Афанасьев, А.Ю. Власенко. Семантические ошибки в параллельных программах для систем с распределенной памятью и методы их обнаружения современными средствами отладки// Вестник КемГУ. Вып. 2 (38).- Кемерово: Изд-во КемГУ. 2009. С. 13-20.
3. Adinets A.V., Bryzgalov P.A., Voevodin Vad.V., Zhumatii S.A., Nikitenko D.A., Stefanov K.S. Job Digest: an approach to dynamic analysis of job characteristics on supercomputers // Numerical methods and programming: Advanced Computing. 2012. V. 13. Section 2. Pages 160-166.
4. Язык программирования UPC. http://upc.lbl.gov/
5. Cantonnet F., El-Ghazawi T., Lorenz P., Gaber J. Fast Address Translation Techniques for Distributed Shared Memory Compilers // IPDPS conference. 2005.
6. Руководство по языку программирования UPC. http://upc.gwu.edu/downloads/Manual-1.2.pdf
7. Спецификация GASP интерфейса. http://gasp.hcs.ufl.edu/gasp-1.5-61606.pdf
8. Тестовый пример, реализующий операцию Собеля. http://upc.gwu.edu/download.html (раздел «Benchmarking/GWU_Examples»)