УДК 004.021

АВТОМАТИЗАЦИЯ ПОИСКА ОШИБОК И НЕЭФФЕКТИВНОСТЕЙ В ПАРАЛЛЕЛЬНЫХ ПРОГРАММАХ

А. С. Антонов[1], Вад. В. Воеводин[1], С. А. Жуматий[1], Д. А. Никитенко[1], К. С. Стефанов[1], П. А. Швец[1]

Вместе с ростом масштабов решаемых на суперкомпьютерных системах задач существенно более значимой становится проблема эффективного использования доступных ресурсов. Избыточные вычисления, вызванные неэффективной реализацией алгоритмов, неоправданное множество тестовых запусков, неучтенные особенности архитектуры вычислительной системы или используемого программного обеспечения — это и многое другое в совокупности приводит к неоправданным расходам вычислительных ресурсов, увеличению времени разработки, удорожанию получения результата. Существуют разные подходы для автоматизированной оценки эффективности и поиска ошибок в параллельных приложениях. В настоящей статье предложен комплексный подход к исследованию эффективности программ в ходе их выполнения. Работа выполнена при финансовой поддержке Министерства образования и науки РФ, государственный контракт № 14.514.11.4062.

Ключевые слова: суперкомпьютер, производительность, исследование эффективности, параллельные вычисления, параллельные программы, динамические характеристики программ, высокопроизводительные вычисления, профилирование, мониторинг, суперкомпьютерный центр.

1. Введение. Пристальное внимание к вопросам эффективного выполнения программ на суперкомпьютерных системах особенно явно прослеживается в последнее время. Можно утверждать, что повышение эффективности является одним из важнейших вопросов высокопроизводительных вычислений. Действительно, с ростом масштабов вычислительных систем, усложнением их архитектур и расширением возможностей повышаются и масштабы решаемых задач. Растущие возможности, в свою очередь, порождают еще больший спрос на вычислительные ресурсы. Даже самые крупные вычислительные центры, например Суперкомпьютерный центр МГУ имени М. В. Ломоносова, в состав которого входит крупнейшая из российских систем, включенных в мировой рейтинг Top500 (суперкомпьютер “Ломоносов” с пиковой производительностью 1.7 PFlop/s), характеризуются крайне высокой загрузкой [1]. Вследствие этого пользователям зачастую приходится ждать часами, пока их программа будет поставлена на счет. Принимая во внимание, что на практике не каждая задача дает корректный результат с одного запуска даже после отладки, вопрос эффективности выполнения программ естественным образом сводится к вопросу автоматизированного поиска потенциальных неэффективностей, логических и семантических ошибок. Наличие средств, осуществляющих такой поиск, должно привести к сокращению цикла разработки, увеличить эффективность использования оборудования и оказать помощь в оптимизации пользовательского приложения.

Существуют различные подходы к оценке эффективности и анализу динамических характеристик программ.

Некоторые из них основаны на данных системного мониторинга и позволяют увидеть ход выполнения программы с точки зрения аппаратуры [2–6]. Существенным требованием к таким методам является незначительное влияние на скорость работы основной программы и минимизация накладных расходов, что является достаточно сложной задачей, особенно когда речь идет о системах передового уровня производительности.

Другие подходы опираются на профилирование и инструментирование, они в большей мере ориентированы на исследование структур программ. Среди них можно выделить как относительно простые инструментарии для получения данных о программе, так и сложные пакеты, применяемые для подроб-

[1] Научно-исследовательский вычислительный центр, Московский государственный университет им. М. В. Ломоносова, Ленинские горы, д. 1., стр. 4, 119992, Москва; А. С. Антонов, вед. науч. сотр., e-mail: asa@parallel.ru; Вад. В. Воеводин, науч. сотр., e-mail: vadim@parallel.ru; С. А. Жуматий, вед. науч. сотр., e-mail: serg@parallel.ru; Д. А. Никитенко, науч. сотр., e-mail: dan@parallel.ru; К. С. Стефанов, ст. науч. сотр., e-mail: cstef@parallel.ru; П. А. Швец, программист, e-mail: shvets.pavel.srcc@gmail.com

© Научно-исследовательский вычислительный центр МГУ им. М. В. Ломоносова

ного анализа отдельно взятого приложения. В силу высокой сложности такие пакеты нельзя применять для оценки потока задач, поскольку слишком высокими оказываются накладные расходы.

Третья группа подходов ограничивается статическим анализом текста программ [7]. В таком подходе привлекательным видится то, что для статического анализа вообще не требуется самого запуска на целевом суперкомпьютере, многие шаблоны неэффективного поведения удается выявить просто по исходному коду, а каждое такое выявление ошибки или неэффективности уменьшает время работы реальной задачи.

В зависимости от того, какие цели преследуются, можно выбрать соответствующий подход к исследованию эффективности. Однако когда речь идет о функционировании целого суперкомпьютерного центра, целесообразно рассматривать весь комплекс задач, а потому крайне востребован комплексный инструментарий, позволяющий использовать различные подходы или их сочетания.

Отвечая этой востребованности, предлагается подход, объединяющий в себе возможности различных вышеозначенных вариантов. Идея его состоит в том, чтобы:

— где возможно (например, в UPC-программах), проводить статический анализ текста и на его основании еще до запуска программы выявлять те или иные аномалии;

— в параллельных программах и пакетах, использующих MPI (а таких, согласно статистике использования суперкомпьютерных систем, подавляющее большинство), опираясь на профилирование, выявлять наиболее типичные ошибки и неэффективное поведение;

— для всех задач, вне зависимости от используемых в них технологий параллельного программирования, собирать данные системного мониторинга и предоставлять пользователю отчет для дальнейшего изучения, что даст возможность лучше понять поведение программы в условиях реального счета на данной программно-аппаратной платформе.

2. Анализ по данным системного мониторинга. Следуя традиции измерять производительность суперкомпьютеров во флопсах (количестве производимых операций с плавающей точкой в секунду) и оценивать близость полученной реальной производительности к пиковой, критерием эффективности работы программы мы будем считать приближение реальной производительности, достигнутой исследуемой программой, к пиковой производительности части вычислительной системы, которая была выделена этой программе для работы. Это не единственный возможный критерий, но в данной работе мы сосредоточимся на таком способе оценки эффективности.

По данным системного мониторинга мы можем определить реальную производительность программы. Для этого во многих современных процессорах имеются встроенные счетчики выполненных операций разных классов, в частности таким образом можно подсчитать и число производимых операций с плавающей точкой.

При этом, исследуя данные других датчиков системного мониторинга, можно установить причины снижения реальной производительности программы. Повышение количества кэш-промахов укажет на низкую локальность использования данных программы, наличие большого трафика по сети, используемой для доступа к общей файловой системе, наличие интенсивного ввода-вывода, мешающего вычислениям, и др.

Рассмотрим несколько примеров.

На рис. 1 показаны данные системного мониторинга при выполнении теста Linpack.

По графикам видно, что загрузка процессора почти постоянно держится на уровне 100% и производительность составляет примерно 5 Гфлопс (пик на времени 10:30:00 — артефакт используемой системы получения данных, сейчас ведутся работы по устранению таких артефактов). Результат запуска самого теста Linpack показывает производительность 545.7 Гфлопс, он запускался на 64 вычислительных ядрах, т.е. производительность на ядро составляет примерно 8.52 Гфлопс. Разница показаний самого теста и данных системного мониторинга объясняется тем, что компилятор использует векторные команды процессора, которые тест считает за две операции с плавающей точкой, а аппаратные счетчики процессора — за одну. С учетом этой поправки данные системного мониторинга вполне можно учитывать при оценке эффективности работы программы.

На рис. 2 приведены данные программы, эффективность выполнения которой держится на достаточно хорошем уровне. При этом на графиках четко видны два этапа работы задачи. На первом этапе идет обсчет заранее загруженных данных (примерно до 22:30), при этом реальная производительность имеет относительно стабильный по времени характер. На втором этапе работы резко увеличивается количество кэш-промахов, при этом начинается активный обмен по сети Ethernet, к которой подключена общая файловая система. Эффективность работы разных процессоров получает большой разброс. Видимо, некоторые процессы в это время продолжают расчеты, а другие занимаются вводом-выводом, вызывая при этом дисбаланс загрузки процессоров, а временами и полную приостановку вычислений, что видно на

графиках.

На рис. 3 приведены данные по одной из программ, выполненных на суперкомпьютере “Ломоносов”. На графиках видны периоды повышенного уровня производительности, каждый из которых оканчивается всплеском кэш-промахов. Можно строить разные догадки о структуре данной программы и ее работе с памятью, однако в данном случае по данным мониторинга получить однозначный ответ вряд ли удается, что показывает необходимость привлечения других методов исследования эффективности работы программ.

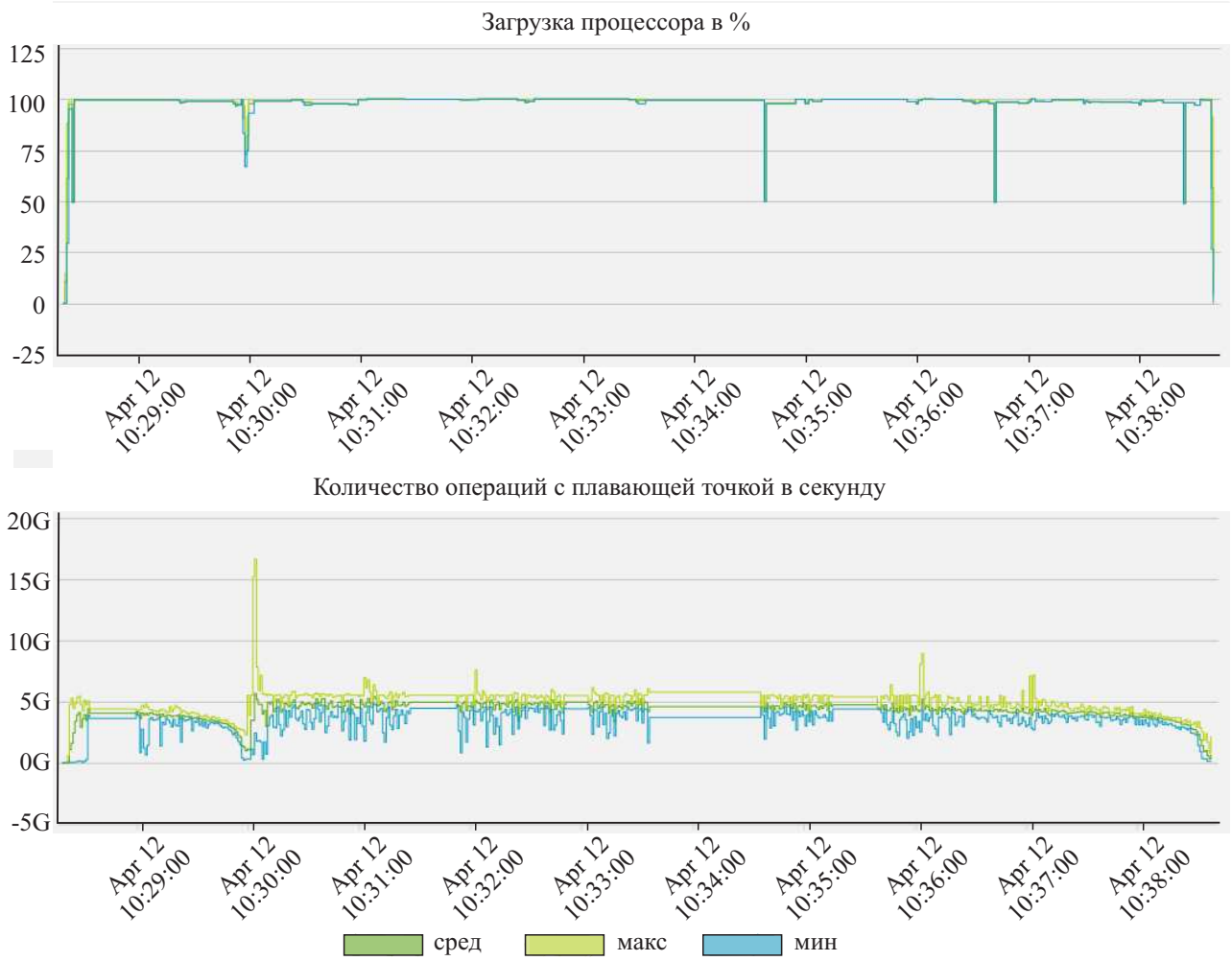

Рис. 1. Данные системного мониторинга при выполнении теста Linpack

3. Логические и семантические ошибки в программах. При написании программ могут возникать самые разные ошибки. Синтаксические ошибки, связанные с неправильным употреблением синтаксических конструкций языка программирования, обычно отлавливаются еще на этапе компиляции и не представляют проблемы для обнаружения и исправления. Сложнее дело обстоит в случае, если синтаксис языка программирования соблюден, но программа либо некорректно завершается, либо выполняет не совсем то, что от нее ожидал программист. Это означает, что в программе оказались ошибки, отличные от синтаксических. Обычно выделяют семантические или логические ошибки [8]. Разделение ошибок на семантические и логические является достаточно тонким, некоторые авторы не различают эти два типа, другие проводят довольно условное разграничение.

При разработке программ важное значение имеют не только ошибки, приводящие к зависанию программы или к неверному результату, но и такие ошибки (или неверные решения, принятые при разработке программы), по вине которых программа выполняется с низкой эффективностью.

Еще более усложняется ситуация при написании программ для параллельных компьютеров. В этом случае чаще всего к традиционному языку программирования добавляется какое-то расширение (реализуемое либо языковыми средствами, либо указаниями компилятору, либо библиотечными функциями).

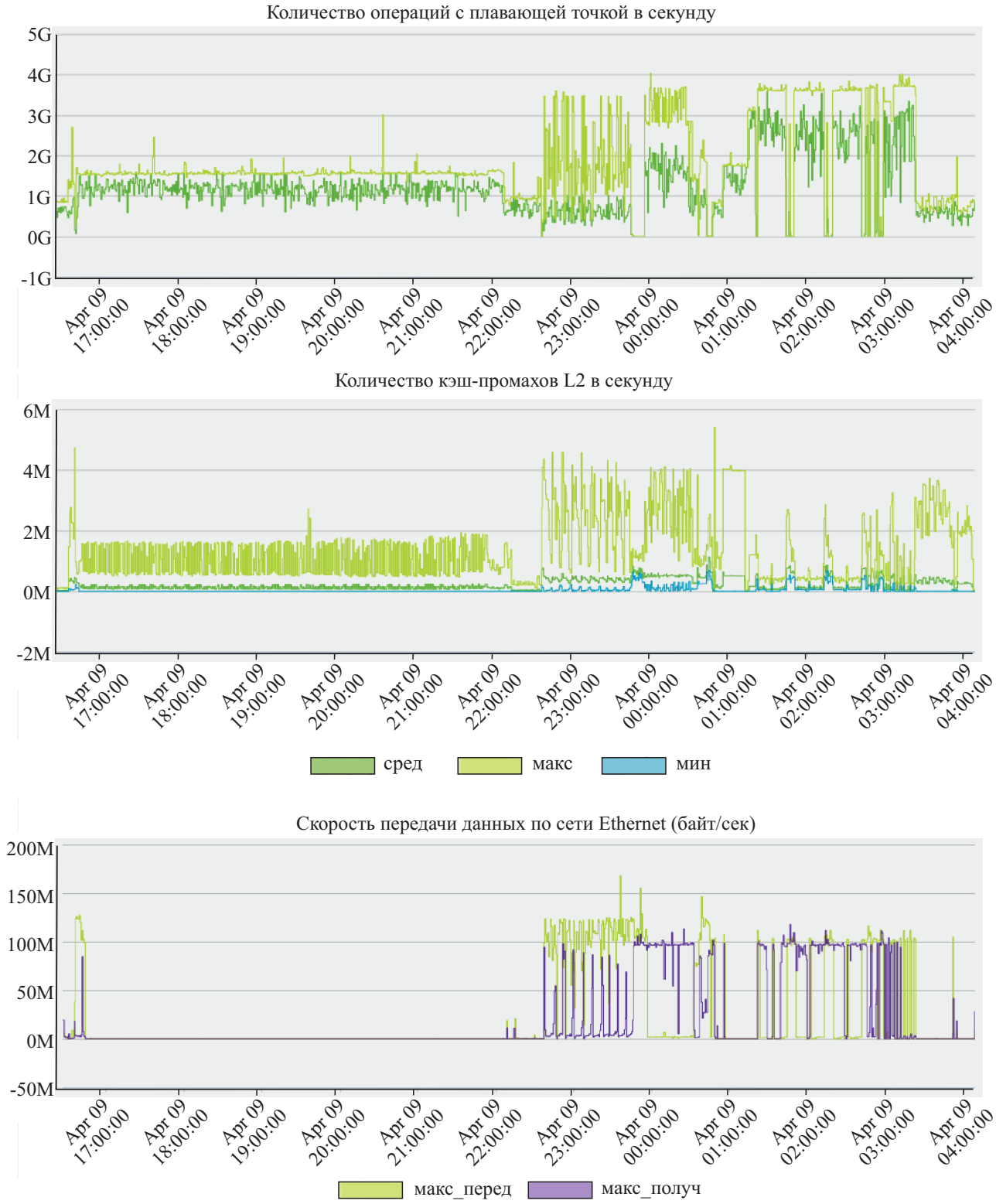

Рис. 2. Данные системного мониторинга программы с этапом активного ввода-вывода

Примерами таких расширений могут служить широко распространенные технологии MPI и OpenMP [9], а также другие технологии, например CUDA или UPC.

Использование технологии параллельного программирования приводит к тому, что к обычным ошибкам последовательной программы добавляются специфические ошибки, связанные с распараллеливанием.

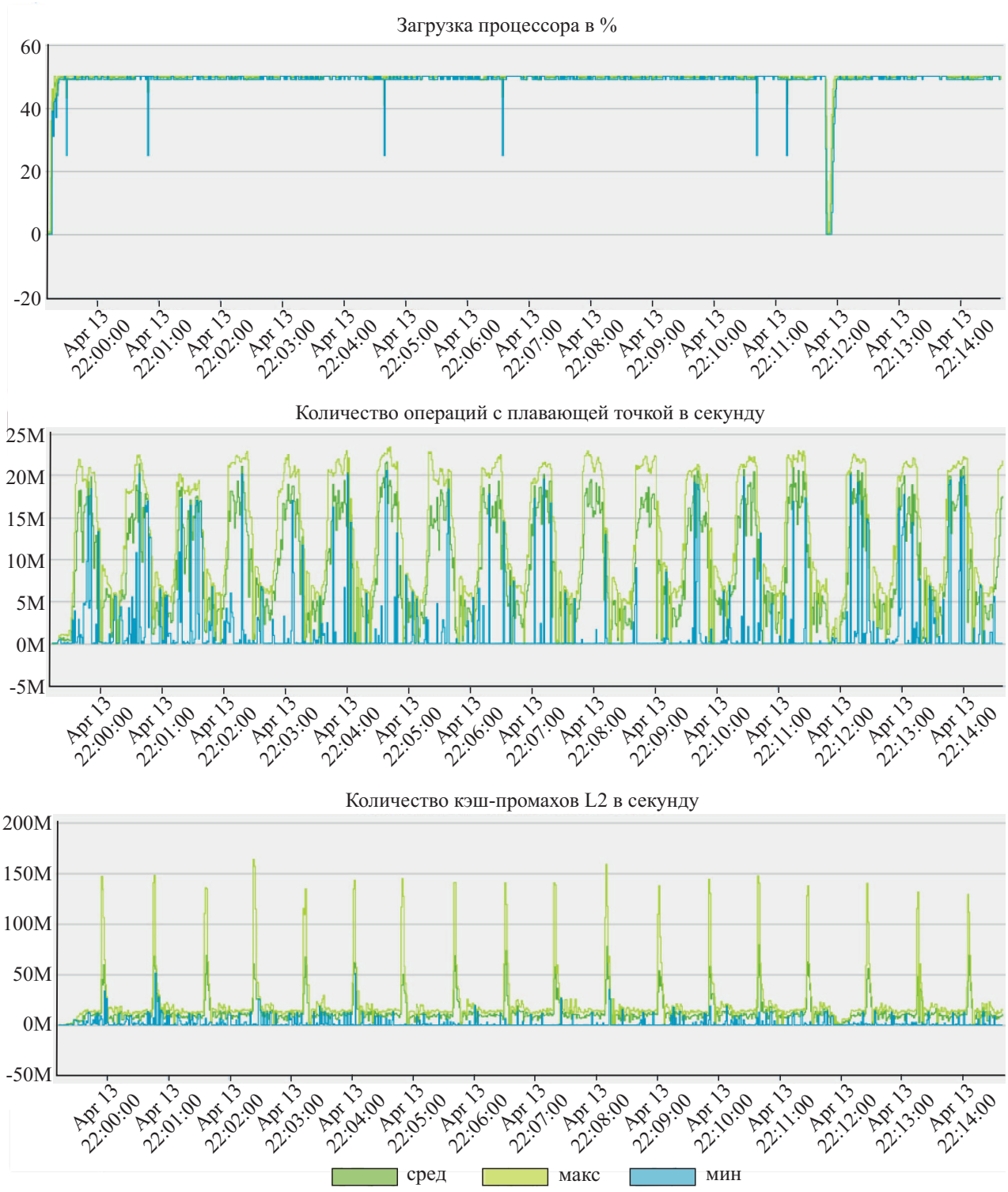

Рис. 3. Данные системного мониторинга, не имеющие однозначной трактовки

При этом вместо одного потока исполнения чаще всего возникает множество взаимодействующих асинхронных потоков, что значительно усложняет процесс поиска семантических или логических ошибок.

Наиболее известными примерами ошибок, присущих именно параллельным программам, являются тупиковые ситуации (дедлоки) и гонки данных. Тупиковая ситуация возникает, когда несколько потоков исполнения программы одновременно оказываются заблокированными на операции взаимодействия (например, все процессы MPI-программы ждут приема данных от других процессов). В этом случае процессы программы никогда не будут корректно завершены, и программу придется прерывать внешними средства-

ми. Еще хуже оказывается тот случай, когда возникновение тупиковой ситуации зависит от реализации или текущих возможностей вычислительной системы (например, от наличия внутренних буферов MPI, если все процессы пытаются послать сообщение). В этом случае ошибка может долгое время не проявляться и оказаться крайне трудно отлавливаемой.

Гонки данных возникают, 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 также может возникать или не возникать в зависимости от внешних условий, и поэтому крайне сложно отлавливается. Известно также большое количество других ошибок, присущих именно параллельным программам, обнаружение которых требует значительных усилий.

Существуют различные подходы к отладке параллельных программ. В отличие от последовательного случая, для параллельных программ не вполне годится пошаговое выполнение программы, поскольку проявление многих ошибок может зависеть от того, в какой стадии выполнения находятся в данный момент различные потоки исполнения. Поэтому зачастую анализируют собранные во время выполнения программы трассы, сравнивают отладочные выдачи с разных процессов и т.д. Однако до сих пор не предложено ни одного удобного инструмента, который полностью удовлетворял бы требования разработчиков параллельных программ.

4. Поиск ошибок в программах, использующих технологию MPI. Достаточно удобным базовым инструментом для получения исходных сведений о свойствах MPI-приложений является PMPI.

PMPI предоставляет стандартный интерфейс для профилирования MPI-программ. С использованием PMPI программист может создавать обертки вокруг нужных ему вызовов MPI и собирать всю необходимую для профилирования информацию.

PMPI обладает двумя значительными преимуществами перед специализированными библиотеками профилирования.

Первое — PMPI является частью стандарта MPI и поэтому является полностью переносимым и должен поддерживаться всеми полноценными реализациями MPI.

Второе — большинство библиотек MPI для разных языков программирования реализуют вызовы соответствующих процедур языка программирования Си. В связи с этим достаточно создать обертку для MPI-вызовов на языке Си, и ее можно будет использовать с другими языками программирования.

Рассмотрим методику сбора данных с использованием PMPI. Каждая стандартная MPI-функция может быть вызвана как с префиксом MPI, так и с префиксом PMPI. Таким образом, создав свою собственную функцию MPI_Send(), в которой мы будем вызывать PMPI_Send(), мы не нарушим логику работы профилируемой программы, зато сможем собирать и анализировать все параметры вызовов функции MPI_Send().

Таким образом, в частности, появляется возможность получить размер пересылаемых данных и идентификаторы процессов, которые участвовали в пересылке, а потом построить по этим данным профиль коммуникаций MPI-приложения.

5. Неэффективности в UPC. Помимо технологии MPI, ориентированной прежде всего на системы с физически распределенной памятью, широко используются и другие технологии. Одной из интересных и известных технологий, используемых при написании параллельных программ, является UPC.

UPC (Unified Parallel C) представляет собой расширение языка программирования Си, предназначенное для работы на многопроцессорных системах. В UPC используется модель PGAS (Partitioned Global Address Space), в рамках которой вся доступная память является адресуемой глобальной памятью и может быть адресована пользователем напрямую.

Основным достоинством языка UPC является относительная простота создания параллельных программ, при этом зачастую эффективность реализации сравнима с использованием технологии MPI. Однако несмотря на простоту, при создании UPC-программ пользователь может совершить ряд стандартных ошибок, которые приводят к снижению эффективности программы. В связи с этим представляется безусловно полезным попытаться выделить набор таких ошибок — шаблонов неэффективностей, а также научиться их определять в реальных программах.

Например, путем анализа кода программы в ряде случаев удается выявить неэффективное распределение общей памяти между процессами или некоторые случаи использования общей памяти вместо локальной, что зачастую приводит к снижению эффективности, и др. Для подобного анализа зачастую достаточны методы статического анализа.

6. Заключение. Проблема исследования эффективности и поиска ошибок в параллельных приложениях является задачей исключительно важной, одной из первостепенных для любого пользователя

высокопроизводительных компьютеров. Однако не существует решения, одинаково подходящего для всех пользователей и всех типов суперкомпьютеров. Создание программной системы, помогающей в решении данной проблемы, позволит не только повысить качество параллельных программ и увеличить эффективность использования суперкомпьютерной техники, но и снизить длительность цикла разработки параллельных приложений.

Ввиду значительной неоднородности используемых технологий и программно-аппаратных платформ, не существует единого подхода, пригодного для любой пользовательской программы и любого целевого суперкомпьютера. Поэтому представляется целесообразным использование комбинированных методов, применяемых в зависимости от конкретной ситуации.

Отдельные неэффективности в параллельных программах зачастую возможно обнаружить методами статического анализа. К таким методам относятся, например, исследования эффективности поведения UPC-программ на основе шаблонов их поведения.

С другой стороны, широкий класс программ, использующих технологию параллельного программирования MPI, может быть исследован на основе данных, собранных после анализа трасс, полученных через стандартное средство PMPI.

Вне зависимости от типа используемой технологии параллельного программирования, перспективным остается подход, основанный на данных системного мониторинга, полученных имеющимися программно-аппаратными средствами. Этот подход позволяет выявлять неэффективности и ошибки, проявляющиеся в ходе конкретного запуска, на конкретном сочетании приложения и выделенных ему ресурсов.

Есть все основания полагать, что предлагаемый подход и разрабатываемый комплекс средств, направленный на автоматизированное обнаружение ошибок и неэффективного поведения параллельных приложений, позволят оказать значительную помощь разработчикам параллельных программ и держателям суперкомпьютерных систем, существенно снизив стоимость и время разработки параллельных приложений, а также увеличив эффективность использования суперкомпьютера.

СПИСОК ЛИТЕРАТУРЫ

1. *Воеводин Вл.В., Жуматий С.А., Соболев С.И., Антонов А.С., Брызгалов П.А., Никитенко Д.А., Стефанов К.С., Воеводин Вад.В.* Практика суперкомпьютера “Ломоносов” // Открытые системы. 2012. № 7. 36–39.
2. *Адинец А.В., Брызгалов П.А., Воеводин Вад.В., Жуматий С.А., Никитенко Д.А., Стефанов К.С.* Job digest — подход к исследованию динамических свойств задач на суперкомпьютерных системах // Вычислительные методы и программирование. 2012. **13**. 160–166.
3. *Брызгалов П.А., Жуматий С.А., Никитенко Д.А., Адинец А.В.* Система визуализации параметров работы больших вычислительных систем // Сб. трудов Международной научной конференции “Параллельные вычислительные технологии 2012” (ПаВТ-2012). 2012. 714.
4. *Адинец А.В., Брызгалов П.А., Воеводин В.В., Жуматий С.А., Никитенко Д.А.* Мониторинг, анализ и визуализация потока заданий на кластерной системе // Материалы XI Всероссийской конференции “Высокопроизводительные параллельные вычисления на кластерных системах”. Нижний Новгород: Изд-во Нижегородского госуниверситета, 2011. 10–14.
5. *Адинец А.В., Брызгалов П.А., Воеводин Вад.В., Жуматий С.А., Никитенко Д.А.* Об одном подходе к мониторингу, анализу и визуализации потока заданий на кластерной системе // Вычислительные методы и программирование. 2011. **12**. 90–93.
6. *Никитенко Д.А., Стефанов К.С.* Исследование эффективности параллельных программ по данным мониторинга // Вычислительные методы и программирование. 2012. **13**. 97–102.
7. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002.
8. *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. Вып. 2. Кемерово: Изд-во КемГУ, 2009. 13–20.
9. *Антонов А.С.* Технологии параллельного программирования MPI и OpenMP. М.: Изд-во Моск. ун-та, 2012.

Поступила в редакцию
19.03.2013