



Московский государственный университет имени М.В. Ломоносова  
Факультет вычислительной математики и кибернетики  
Кафедра суперкомпьютеров и квантовой информатики

Худолеева Анна Александровна

**Моделирование влияния агента системы мониторинга  
производительности на работу пользовательских задач**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**Научный руководитель:**  
к.ф.-м.н., с.н.с. НИВЦ МГУ  
Стефанов К.С.

Москва, 2020

## Оглавление

Аннотация.....	3
Введение.....	4
Постановка задачи.....	7
Обзор предметной области, к которой относится рассматриваемая задача.....	8
NWPerf.....	8
LDMS.....	9
Исследования в области обнаружения шума ОС.....	10
Исследование на ASQI Q.....	10
Влияние шума ОС на коллективные операции.....	11
Построение решения поставленной задачи.....	13
Описание средства, детектирующего присутствие шума.....	14
Описание средства, моделирующего шум агента системы мониторинга.....	16
Описание практической части.....	18
Описание проведенных экспериментов.....	18
Результаты тестирования детектора шума.....	19
Тестирование чувствительности детектора шума.....	23
Заключение.....	26
Список цитируемой литературы.....	27

## **Аннотация**

Изучение параллельных программ с помощью средств мониторинга производительности — распространенная практика. Агент системы мониторинга для сбора данных о работе приложения периодически активируется во время счета этого приложения, внося помехи и занимая ресурсы. Однако вопрос об уровне влияния этих помех является слабо изученным, разработчики систем мониторинга зачастую не проводят исследования в этом направлении. В данной работе рассматривается подход к изучению влияния системы мониторинга производительности суперкомпьютера на пользовательские приложения. В качестве инструмента для измерения влияния агента системы мониторинга предлагается использовать коллективные MPI операции. Время выполнения коллективных MPI операций изучается в присутствии программного средства, моделирующего работу агента системы мониторинга. Оценивается уровень шума, который каждая из рассматриваемых коллективных операций в выбранной конфигурации запуска способна зафиксировать. В работе приводятся данные запусков инструмента с коллективными MPI операциями All-to-All, All-Reduce, Barrier. Найдено, что хорошей стабильностью и чувствительностью обладают операции All-to-All и Barrier.

## **Введение**

Решение фундаментальных проблем науки является неотъемлемой частью прогресса. Один из основных способов нахождения решения этих задач — компьютерное моделирование.

Для соответствия растущим потребностям научного сообщества область высокопроизводительных вычислений постоянно развивается. Основным показателем развития в этой сфере компьютерных наук является увеличение производительности вычислительной машины. Из способов наращивания мощности вычислительного устройства можно выделить горизонтальное и вертикальное масштабирование. Горизонтальное масштабирование — увеличение производительности системы с помощью добавления компонент в эту систему. Вертикальное масштабирование — увеличение производительности системы посредством увеличения производительности каждого компонента этой системы.

Современные суперкомпьютеры — это очень сложные комплексы, состоящие из множества компонентов. Согласно редакции от ноября 2019 года списка Top500 [9] в кластерах, занимающих последние места рейтинга, насчитываются десятки тысяч процессоров. В четверку лидеров входят системы с миллионами ядер. Первые две строчки в списке Top500 занимают суперкомпьютеры Summit [10] и Sierra [11], которые проектировались параллельно тесно сотрудничающими командами специалистов. Разработчики данных систем добились превосходства в показаниях пиковой производительности за счет использования графических ускорителей совместно с мощными центральными процессорами. Для достижения новых значений производительности используются гибридные техники наращивания мощности: увеличение количества узлов и усовершенствование их архитектуры.

Рост сложности системы, под которую разрабатывается приложение, усложняет структуру приложения и затрудняет его разработку. Это ведет к повышению сложности оптимизации приложения под целевую платформу. Однако достижение максимальной производительности и наиболее полное использование предоставленных ресурсов — важная задача, так как невозможность полностью использовать средства суперкомпьютера ведет к замедлению работы приложения и, следовательно, к задержке проведения научного исследования. Для решения данной проблемы оптимизации приложения существует несколько подходов.

Один из способов — подбор оптимальной конфигурации системы, состоящий в многократном запуске приложения с разными настройками системы и сравнение результатов запусков. Метод не подходит для больших приложений, так как приводит к значительным накладным расходам. Также возможно создание аналитической модели запуска приложения, в которой параметрами являются оценки значений различных ресурсов системы. Способ позволяет дать грубый прогноз производительности приложения.

Метод, о котором в дальнейшем пойдет речь — использование специального программного продукта — средства мониторинга производительности.

Средство мониторинга может включать два функционала. Первое назначение — измерение различных характеристик суперкомпьютера для задачи администрирования кластера. Второе назначение — измерение пользовательских приложений, запускаемых на суперкомпьютере. В рамках данной работы интерес представляют средства, обладающие второй возможностью.

При проектировании средства мониторинга производительности разработчики преследуют цель создания масштабируемого, эффективного инструмента, который будет предоставлять достаточное количество информации об используемых пользовательским процессом ресурсах для анализа и возможность хранения собранных данных. Данные могут характеризовать взаимодействие приложения с различными частями системы: с процессором, сетью, памятью. Также возможным требованием к системе мониторинга является простота в использовании и обеспечение инструментария для визуализации данных.

Агент системы мониторинга запускается совместно с исследуемым приложением на тех же узлах с определенным периодом. Период запусков является либо настраиваемым, либо встроенным параметром системы мониторинга. Обычно активация системы мониторинга производительности происходит раз в несколько секунд. Так как средство мониторинга разделяет ресурсы с приложением, оно вносит помехи в его работу и негативно влияет на производительность.

К данному моменту разработано и исследовано достаточное количество различных систем мониторинга. Однако авторы исследований часто утверждают, что влияние средств мониторинга незначительно, не приводя методов, позволяющих это влияние измерить.

Можно проводить запуск приложения без системы мониторинга и совместно с ней. Дальнейшее сравнение измеренных величин, собранных в двух случаях, покажет степень

влияния средства мониторинга. Но этот метод является невыгодным, так как значительное замедление программы будет видно только при использовании большого количества вычислительных узлов суперкомпьютера. Вопросу обнаружения влияния системы мониторинга при использовании небольшого числа ресурсов вычислительного кластера и установления уровня этого влияния посвящена данная работа.

## **Постановка задачи**

- Разработать инструмент для обнаружения влияния системы мониторинга производительности СК — детектор шума.
- Исследовать стабильность работы разработанного инструмента.
- Исследовать детектор шума совместно с источником искусственного шума — программным средством, моделирующим присутствие системы мониторинга производительности.

## **Обзор предметной области, к которой относится рассматриваемая задача**

За последние 20 лет было уделено много внимания созданию различных средств мониторинга производительности: Supermon [1], NWPerf [2], HPCToolkit [3], Performance Co-Pilot [4], LIKWID [5,6], LDMS [7]. Авторы статей об инструментах HPCToolkit, Performance Co-Pilot, LIKWID говорят о том, что эффект, оказываемый этими средствами мониторинга на пользовательские приложения, является незначительным, однако не приводят методов для получения такого вывода.

### ***NWPerf***

Агентом системы мониторинга NWPerf [2] является модуль ядра Linux. Интервал сбора данных для данного средства — одна минута. При создании этого средства мониторинга ставилась задача достичь уровня вносимых помех в работу пользовательского процесса менее одного процента. Для достижения этой задачи были выделены следующие возможности оптимизации: сокращение времени работы средства на всех узлах, уменьшение времени, в течение которого агент запускается и отправляет данные, синхронный запуск процессов мониторинга на всех узлах.

Было проведено исследование влияния средства мониторинга на коллективные операции All-to-All и All-Reduce. На 128 узлах, на 256 вычислительных ядрах запускался цикл из 10,000 коллективных операций с и без средства мониторинга. При частоте сбора метрик 1 раз в 6 секунд, в десять раз превышающих стандартный режим, время выполнения операции All-to-All увеличилось на 27%, а операции All-Reduce на 9.46%. Но при стандартной частоте съема данных прочие помехи в системе перекрывали шум от NWPerf, что свидетельствует о приемлемости частоты работы агента раз в минуту.

### ***LDMS***

Достаточно подробно было исследовано влияние средства мониторинга LDMS [7] на работу различных компонент высокопроизводительной системы. Проводилось измерение запусков бенчмарков и реальных приложений с различной интенсивностью мониторинга: без мониторинга, низкая частота мониторинга — 1 раз в 20 секунд, высокая — 1 раз в 1 секунду. Изменение уровня вносимых помех позволило отличить шум операционной системы от шума системы мониторинга.



Средой проведения экспериментов выступали две большие вычислительные системы. Тестирование с использованием бенчмарков показало, что влияние инструмента LDMS при высокой интенсивности мониторинга имеет незначительный эффект на сетевое соединение и производительность MPI операций. На основе результатов одного из тестов был сделан вывод о том, что синхронизированный запуск агента системы мониторинга на узлах уменьшает его влияние. Совместный запуск LDMS с большими приложениями, чувствительными к состоянию узлов, сети, системы ввода/вывода, также не демонстрирует существенного уменьшения производительности этих приложений и влияния агента мониторинга на них.

Результаты исследований вопроса влияния средств мониторинга производительности NWPerf [2] и LDMS [7] на пользовательские приложения сложно обобщить: для средства NWPerf при частоте сбора данных 1 раз в 6 секунд было получено заметное увеличение времени выполнения коллективных операций, для средства LDMS при частоте работы агента мониторинга 1 раз в секунду увеличение времени является статистически незаметным. Исследования перечисленных систем мониторинга проводились на разных системах при использовании разных тестов, что может быть причиной получения отличий в степени изменения производительности коллективных операций. Тем не менее вопрос влияния системы мониторинга на пользовательское приложение является недостаточно подробно изученным.

### ***Исследования в области обнаружения шума ОС***

Хорошо исследованной является область обнаружения уровня влияния шума операционной системы на работу пользовательских программ. Методы, применяемые в этой области, можно распространить на изучение шума, создаваемого системой мониторинга.

### ***Исследование на ASCI Q***

В статье [8], описывающей масштабную работу по оптимизации приложения SAGE на суперкомпьютере ASCI Q, состоящего из 8,192 процессоров, были выделены несколько причин снижения производительности приложения и даны рекомендации для избежания неэффективного использования ресурсов суперкомпьютера. Для уменьшения шума ОС предлагается рассмотреть вариант нестандартного расположения программы на вычислительных узлах. Например, запустить программу не на всех ядрах узла. Выделение трех процессоров под задачу, вместо доступных четырех на узле, ведет к увеличению производительности приложения, так как в системе остается место для «развертки»

коллективных операций на четвертом процессоре. При выделении четырех процессоров время выполнения операции All-Reduce росло линейно с увеличением числа узлов, в то время как при использовании трех, двух или одного оно оставалось постоянным. Также внимание обращается на номера узлов, которым назначается выполнение приложения. В кластере из 32 узлов некоторым из них отведена роль в управлении ресурсами вычислительного устройства, поэтому на этих узлах больше шума, и, для оптимизации работы приложения, их можно исключить. В лучшем случае, минимизировав уровень шума ОС, удалось добиться ускорения в 2,21 раза.

Авторы приводят рассуждения и результаты экспериментов, свидетельствующие о том, что на хорошо синхронизированные приложения, небольшой, но частый синхронный шум, влияет сильнее, чем долгая задержка, возникшая на одном узле. Они вводят понятие «резонирования» программы с системным шумом. Исключив резонанс системного шума со структурой выполнения приложения, можно достичь лучшей производительности работы приложения.

### **Влияние шума ОС на коллективные операции**

В работе [12] представлено обширное исследование влияния помех, вносимых операционной системой в производительность коллективных MPI операций при большом числе используемых вычислительных узлов.

Авторы реализовали бенчмарк, с помощью которого можно измерить минимальное значение помех, вносимых процессами, связанными с деятельностью операционной системы. После того, как было установлено, что общий уровень шума на целевой архитектуре является низким, в систему были внедрены искусственные помехи. Их наличие позволило смоделировать запуск приложения на более шумных, чем в действительности, ядрах и получить данные о поведении различных операций синхронизации в таких условиях.

Была измерена производительность коллективных MPI операций Barrier, All-Reduce и All-to-All в присутствии разного уровня шума. Для всех трех вариантов коммуникации было установлено, что на время завершения операции более сильное влияние оказывает несинхронизированный шум, нежели синхронизированный. По сравнению с операциями Barrier и All-Reduce, на операцию All-to-All синхронизированный шум оказывает почти незаметный эффект. Авторы статьи объясняют этот факт тем, что выполнение операции All-to-All является слабо согласованным, то есть во время коммуникации часть процессов

реализует операцию, другая часть простаивает. Поэтому шум, который попадает на простаивающие процессы, не вносит вклад в замедление операции.

Влияние синхронизировано возникающего в системе шума оказывается незначительным на работу слабо синхронизированного приложения. Однако, в отличие от авторов статьи [8], авторы исследования [12] считают, что на синхронизированное приложение несинхронизированный шум оказывает большее влияние, чем равные по продолжительности помехи, возникающие в системе одновременно. Результаты запуска операции Barrier подтверждают это (Рисунки 1, 2).

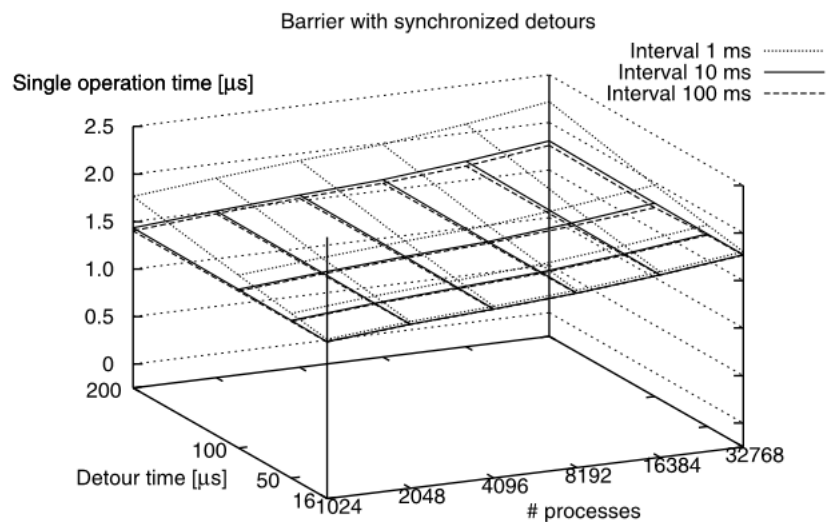


Рисунок 1: Результаты тестирования Barrier с искусственным синхронизированным шумом ОС в исследовании [12].

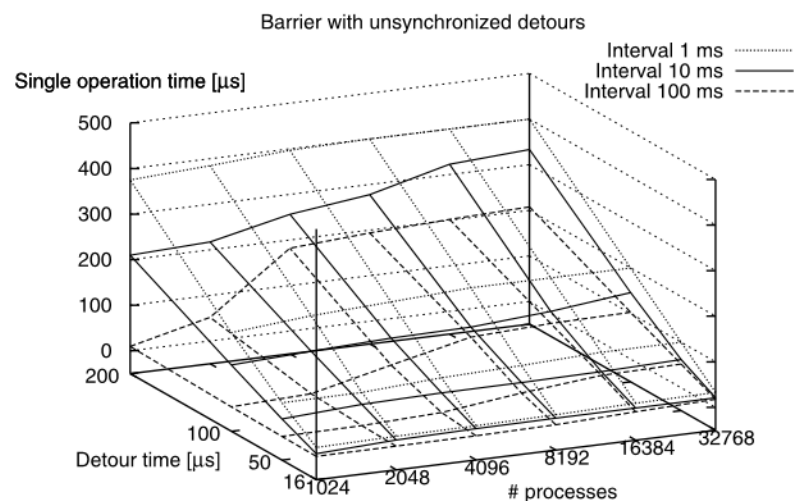


Рисунок 2: Результаты тестирования Barrier с искусственным несинхронизированным шумом ОС в исследовании [12].

## Построение решения поставленной задачи

Одним из самых популярных средств при программировании приложений для вычислительных кластеров является MPI — интерфейс, поддерживающий работу параллельных процессов с помощью передачи сообщений между ними. В исследовании [13] проводился анализ 110 параллельных приложений с открытым исходным кодом на предмет использования интерфейса MPI. Было обнаружено, что во всех 110-ти исследованных приложениях имеются вызовы коллективных операций.

Параллельное приложение может выполняться несимметрично, то есть в зависимости от ранга, процессам назначаются роли, реализованные разными фрагментами кода. На большом вычислительном комплексе нельзя гарантировать бесперебойность работы каждого узла, симметрию выполнения приложения на каждом ядре. Это может быть связано как с аномалиями в поведении аппаратуры, так и с активацией различных системных процессов, прерывающих выполнение программы. Поэтому даже в приложениях, где процессы выполняют один и тот же фрагмент кода, стартовав синхронно, время выполнения последовательной части программы будет разным для разных процессов. Так как при запуске коллективной операции все процессы должны одновременно выполнить пересылку сообщений или синхронизацию, то часть процессов будет простаивать, ожидая готовности к выполнению операции «опаздывающие» процессы. Поэтому время работы параллельного приложения измеряется по самому медленному из процессов. На рисунке 3 продемонстрировано влияние задержек, возникающих на отдельных процессах, на операции синхронизации и время их выполнения.

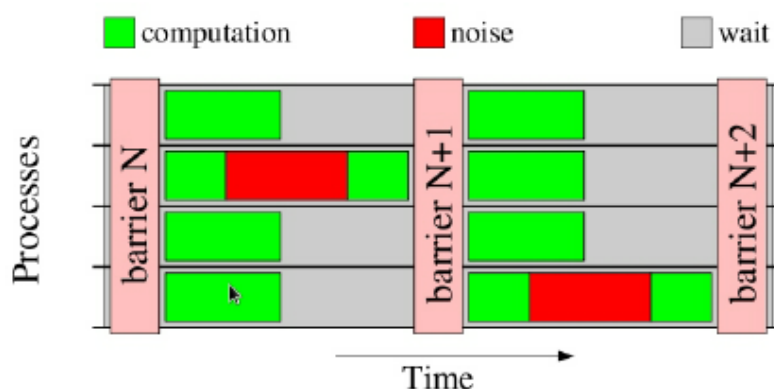


Рисунок 3: Выполнение операции Barrier при возникновении дополнительной нагрузки на одном процессе. Источник изображения - статья [16].

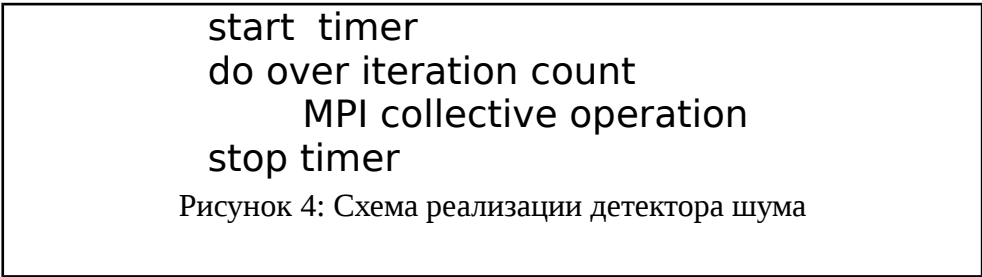
В силу тех фактов, что коллективные MPI операции используются во многих параллельных приложениях и могут являться источником замедления работы программы, представляет интерес изучение степени влияния системы мониторинга на их производительность. С другой стороны, коллективные операции можно использовать в качестве инструмента для обнаружения и исследования шума, вносимого системой мониторинга. Такой подход использовался в исследованиях [2, 7]. Однако однозначного ответа на вопрос о влиянии агента системы мониторинга на коллективные операции не было найдено.

Для устранения этого пробела предлагается создать инструмент, с помощью которого можно обнаружить влияние системы мониторинга на пользовательские приложения.

### ***Описание средства, детектирующего присутствие шума***

Программное средство, предназначенное для обнаружения присутствия и определения степени влияния агента системы мониторинга в дальнейшем будем называть детектором шума.

На Рисунке 4 представлена схема реализации детектора шума, в основе которого лежит цикл из коллективных MPI операций. Измеряемой величиной при запуске детектора является время выполнения цикла из *count* числа коллективных MPI операций. Рассматриваемыми коллективными MPI операциями в рамках данной работы являются: Barrier, All-Reduce, All-to-All.



```
start timer
do over iteration count
    MPI collective operation
stop timer
```

Рисунок 4: Схема реализации детектора шума

Для данного программного средства существует несколько свободных параметров, выбрав которые, можно определить конфигурацию детектора шума. Конфигурация инструмента считается подходящей, если при запуске в стандартных условиях, в отсутствии постороннего шума, эксперимент является воспроизводимым, то есть от запуска к запуску время выполнения цикла из коллективных операций изменяется в пределах допустимой погрешности. Вторым требованием к конфигурации детектора шума

является ее способность фиксировать присутствие процессов, связанных с деятельностью агента системы мониторинга, чем точнее чувствительность детектора, тем лучше.

Свободные параметры включают в себя:

- Выбор коллективной MPI операции;
- Количество узлов, на которых запущено программное средство;
- Количество коллективных операций;
- Для операций All-Reduce и All-to-All длина пересылаемого сообщения.

### **Описание средства, моделирующего шум агента системы мониторинга**

Агент системы мониторинга производительности запускается на отдельных ядрах узла, где вычисляется исследуемое приложение, и собирает данные с аппаратных датчиков, период сбора данных составляет единицы секунды. Более подробное описание работы системы мониторинга приведено в исследовании [18].

Для исследования чувствительности детектора шума было решено построить программное средство, моделирующее присутствие системы мониторинга производительности — источник шума. На Рисунке 5 представлен фрагмент кода источника шума.

В основе работы этого средства лежит цикл, в котором вычисляется сумма (значение суммы хранится в переменной *sum*) массива *\*a* объемом 250 000 слов (значение константы *SIZE*) типа *long long*. В *WORK\_TIME* хранится доля секунды, которую работает нагрузка — суммирование массива. В параметр *tw* процедуры *nanosleep()* записана доля секунды, в течение которой сумма массива не вычисляется. Это значение дополняет *WORK\_TIME* до 1 секунды. В дальнейшем под уровнем шума в процентах мы будем понимать значение  $100 \times \text{WORK\_TIME}$ . Например, если источник шума производит нагрузку, уровень которой равен 1%, значит цикл суммирования работает 0,01 секунду, оставшуюся часть секунды бездействует.

Во фрагменте кода, реализующем источник шума, отсутствуют операции синхронизации. С большой вероятностью выполнение такой программы будет асинхронным, что приведет к усилению влияния искусственного шума, как было отмечено при изучении шума операционной системы [9].

```
start_time = MPI_Wtime();
do
{
    t_1 = MPI_Wtime();
    do
    {
        for (int i = 0; i < SIZE; ++i)
        {
            sum += a[i];
        }
        t_2 = MPI_Wtime();
    }
    while ((t_2 - t_1) < WORK_TIME);
}
while ((t_2 - start_time) < BREAK_TIME);
```

Рисунок 5: Фрагмент реализации программного средства, моделирующего влияние агента системы мониторинга — источника шума.

## **Описание практической части**

Исследование разработанных программных инструментов было выполнено на суперкомпьютере Ломоносов-2 [14]. Запуски проводились на тестовом разделе суперкомпьютера. Так удалось избежать ожидания в загруженной очереди основного раздела. На каждом узле СК установлен процессор Intel Haswell-EP E5-2697v3, имеющий 14 ядер. Лимит времени выполнения заданий на тестовом разделе составляет 15 минут. Этим фактом обусловлен выбор некоторых из параметров тестируемых программных средств.

При запуске на суперкомпьютере использовалась библиотека OpenMPI [15].

## **Описание проведенных экспериментов**

Исследование разработанных программных инструментов проходило в два этапа.

Во-первых, необходимо было оценить стабильность работы разных вариантов конфигурации детектора шума. В случае существенного разброса времени выполнения коллективных операций на системе Ломоносов-2, нужно отказаться от такой конфигурации, так как она не подходит для многократного использования с целью измерения влияния шума. Причиной этого является отсутствие возможности получить стабильное значение времени работы операции без шума и, следовательно, отличить его от времени работы в присутствии сторонней нагрузки.

Во-вторых, выбранные варианты детектора шума запускались совместно с источником шума. Полученная разница во времени выполнения коллективных операций в этих двух случаях сравнивалась с помощью статистических критериев, описанных в работе [17].

В силу большого разнообразия возможностей проведения запусков была выбрана следующая конфигурация: программные средства запускались на 4 узлах суперкомпьютера Ломоносов-2, по 2 процесса на ядро (один процесс на виртуальное ядро, Hyper Threading включен). Для программы с операцией All-to-All также проводились тесты на 8 узлах, 2 процесса на ядро.

Модель агента системы мониторинга запускалась совместно с программным средством на тех же узлах, но на одном процессе одного ядра.

## **Результаты тестирования детектора шума**

Для сравнения времени работы детектора шума без источника шума и с ним, следовательно, создания возможности обнаружить присутствие искусственно внедренного



шума, необходимо определить базу стандартного времени работы различных вариантов конфигурации детектора, которое должно быть достаточно близким при проведении нескольких измерений. Будем считать, что конфигурация детектора является стабильной, если 99% доверительный интервал среднего значения «чистых» (без источника шума) запусков конфигурации детектора является достаточно узким.

Также конфигурация детектора шума должна быть способна обнаружить присутствие шума с уровнем 1%. При таком уровне шума источника искусственная нагрузка выполняется 1% времени от времени работы детектора. Будем считать, что детектор способен обнаружить шум, если 99% доверительные интервалы среднего значения времени «чистых» запусков детектора и запусков с источником шума не пересекаются и разделены несколькими секундами.

На рисунках 6, 7 ниже представлены результаты экспериментов, проведенных для обнаружения удовлетворяющих требованиям вариантов конфигурации детектора шума. Синим отмечены значения времени для «чистых» запусков детектора, красным — для запусков совместно с источником с уровнем шума 1%.

В таблицах 1, 2 представлены результаты измерений для каждой исследуемой конфигурации детектора шума.

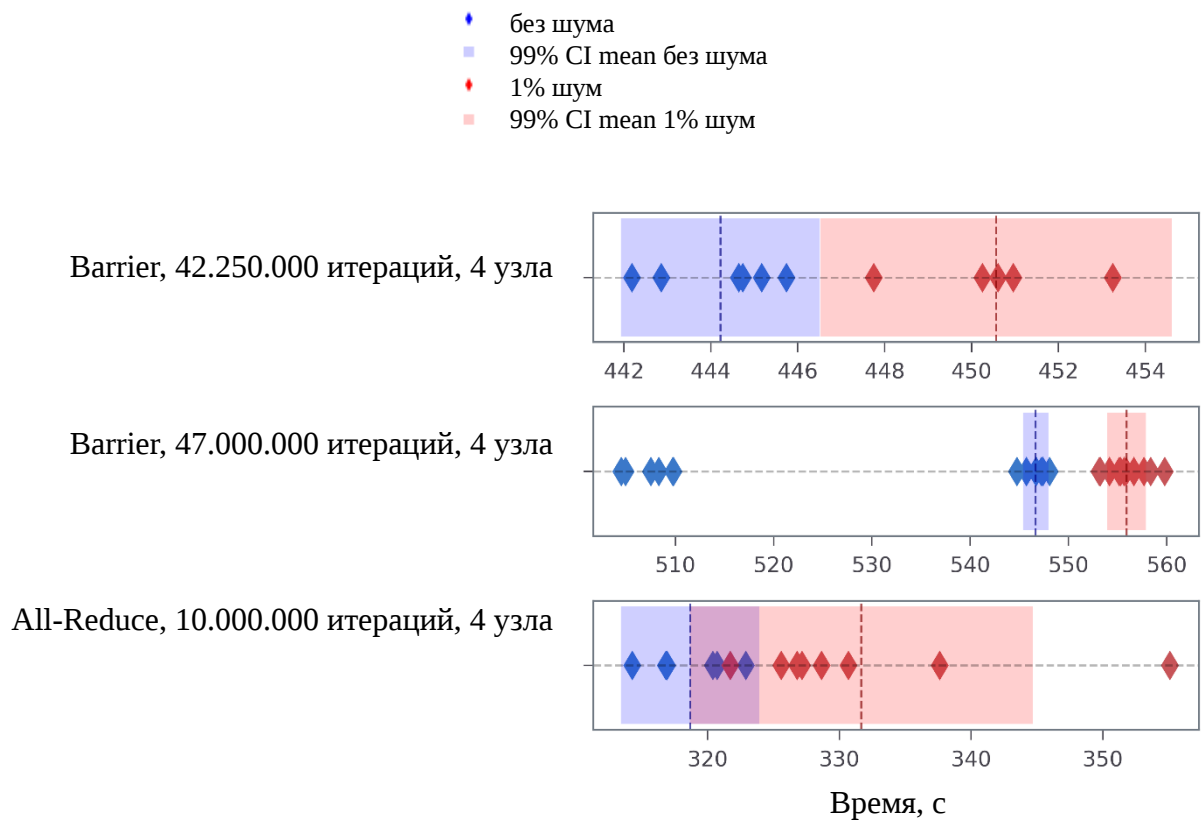


Рисунок 6: Распределение времени для операций Barrier, All-Reduce.

Таблица 1: Результаты запусков для

I: Barrier, 42.250.000 итераций, 4узла;

II: Barrier, 49.000.000 итераций, 4 узла;

III: All-Reduce, 10.000.000 итераций, 2КВ, 4 узла.

		I	II	III
Чистые Запуски	Среднее время, с	444,22	546,66	318,67
	Стандартное Отклонение, с	1,39	1,03	3,19
	Количество Запусков	6	8	6
Запуски с шумом 1%	Среднее время, с	450,56	555,91	331,66
	Стандартное Отклонение, с	1,96	2,09	4,95
	Количество Запусков	5	11	8

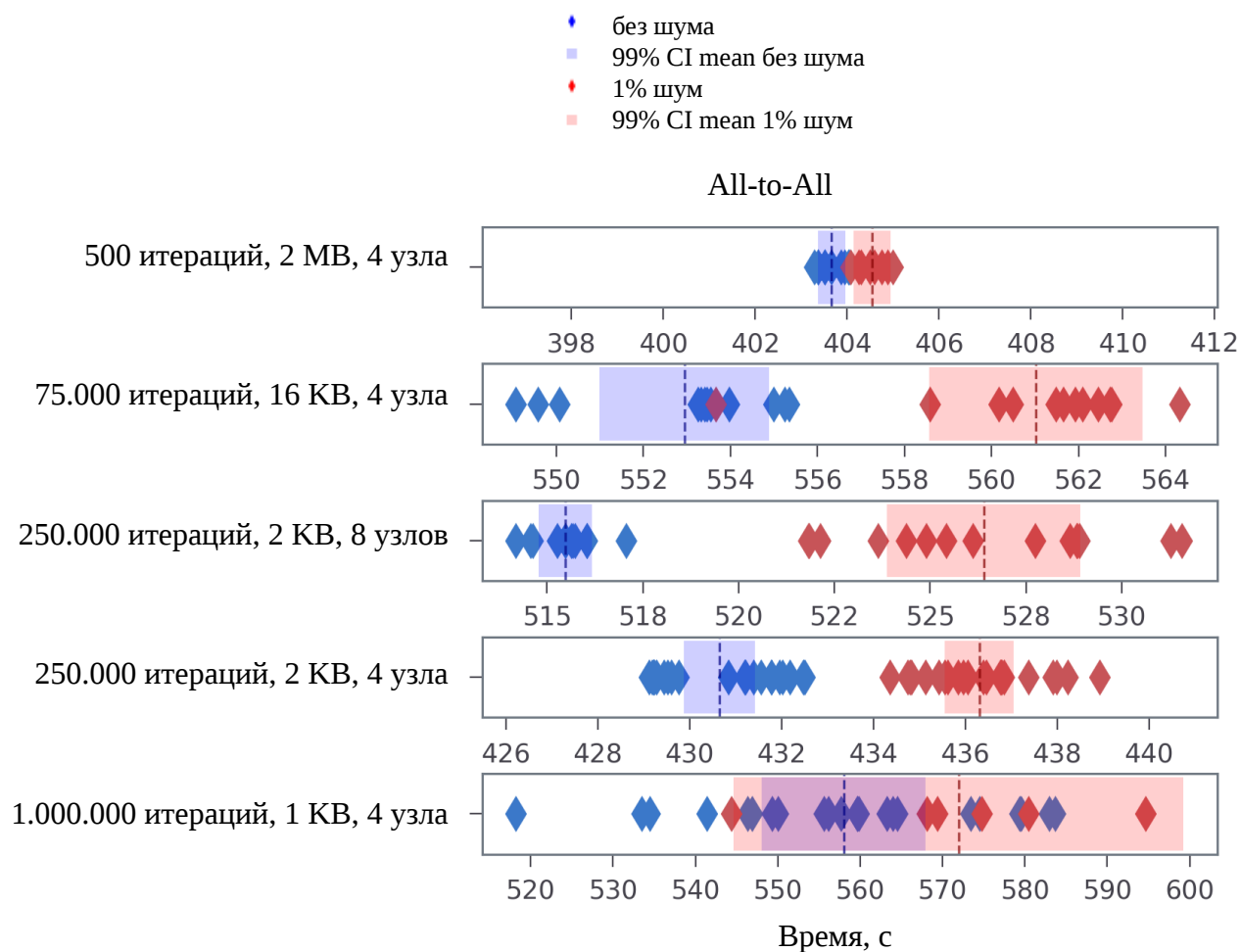


Рисунок 7: Распределение времени для операции All-to-All.

Таблица 2: Результаты запусков для

I: All-to-All, 500 итераций, 2 МВ, 4 узла;

II: All-to-All, 75.000 итераций, 16КВ, 4 узла;

III: All-to-All, 250.000 итераций, 2КВ, 8 узла;

IV: All-to-All, 500.000 итераций, 2 КВ, 4 узла;

V: All-to-All, 1.000.000 итераций, 1 КВ, 4 узла.

		I	II	III	IV	V
Чистые Запуски	Среднее время, с	403,62	552,94	515,88	430,65	557,991304
	Стандартное Отклонение, с	0,29	2,16	1,63	1,28	14,34
	Количество Запусков	10	12	13	22	23
Запуски с шумом 1%	Среднее время, с	404,55	561,04	526,39	436,31	571,96
	Стандартное Отклонение, с	0,32	2,74	3,28	1,24	16,59
	Количество Запусков	8	12	15	22	6

Для 49.000.000 итераций операции Barrier, 4 узла на рисунке 6 видно, что значения времени детектора сгруппированы около двух значений — 509 и 546 секунд. При построении таблицы 1 и 99% доверительного интервала использовалась вторая группа значений времени. Наличие двух групп может быть объяснено тем, что задачи ставились в разные дни. Возможно, на время работы детектора влияют свойства узлов СК, на которых запускается детектор в этой конфигурации, процессы запущенные на них. Тем не менее, значения времени для шума 1% превышают доверительный интервал времени без шума обеих групп, поэтому шум 1% обнаруживается обеими группами. Детектор с 42.250.000 итераций операции Barrier будет пригоден для использования, если, увеличив число измерений, получится сузить доверительный интервал.

All-Reduce с 1.000.000 операций (рисунок 6, таблица 1) не обнаруживает шум 1% — 99% доверительные интервалы для времени с шумом и без пересекаются. Не исключено, что операция All-Reduce может оказаться пригодной в качестве детектора шума при должном подборе параметров запуска.

Для All-to-All с 500 итераций, 2 МВ, 4 узла на рисунке 7 видно, что время с шумом 1% слабо отличимо от «чистого» времени — расстояние между доверительными интервалами составляет доли секунды. Поэтому в дальнейшем эта конфигурация детектора с операцией All-to-All не использовалась.

75.000 операций All-to-All, 16 КВ, 4 узла (рисунок 7, таблица 2) можно считать пригодным инструментом для обнаружения шума, который удовлетворяет требованиям к детектору шума.

Детекторы All-to-All, 250.000 итераций, 8 узлов и All-to-All, 500.000 итераций, 4 узла (рисунок 7, таблица 2) с размером сообщения 2 КВ в обоих случаях показывают стабильный результат, с хорошей возможностью обнаружения шума 1%.

В случае All-to-All 1.000.000 итераций, 1 КВ, 4 узла время «чистых» запусков оказалось сильно разбросанным. При таких параметрах детектора уловить разницу между «чистыми» запусками и запусками с шумом 1% не получилось.

Непредсказуемые значения — выбросы — в ходе проведения исследования наблюдались для всех выбранных коллективных операций. Возможно, на появление разницы во времени между запусками влияют конкретные узлы, которые выделяются на суперкомпьютере для запуска задачи, задержки во взаимодействие между ними.

### ***Тестирование чувствительности детектора шума***

Выше были приведены результаты демонстрирующие способность детектора улавливать шум модели агента мониторинга производительности с уровнем 1%. Современные системы мониторинга стремятся сделать как можно менее «шумными», чтобы они вносили малые помехи в работу пользовательского приложения. Поэтому интерес представляет изучение способности детектора обнаруживать шум с уровнем меньшим, чем 1%. Также необходимо оценить границу чувствительности детектора — насколько малый шум различные конфигурации детектора способны обнаружить.

Ниже на рисунке 8 и в таблицах 3-6 приведены результаты экспериментов, в которых уровень шума постепенно снижался до порога чувствительности (до верхней границы доверительного интервала).

Для тестирования были выбраны конфигурации детектора, которые различают шум 1%, то есть 99% доверительные интервалы среднего значения времени без шума и с шумом 1% не пересекаются:

- All-to-All 500.000 итераций, 2 KB, 4 узла — таблица 3;
- All-to-All 250.000 итераций, 2 KB, 8 узла — таблица 4;
- All-to-All 75.000 итераций, 16 KB, 4 узла — таблица 5;
- Barrier 49.000.000 итераций, 4 узла — таблица 6.

На рисунке 8 и в таблицах 3-6 в качестве значения шума при каждом из указанных уровней нагрузки представлено среднее арифметическое времени по всем соответствующим запускам.

По графикам на рисунке 8 видно, что при нагрузке в 0,5% шума детекторы, использующие операции Barrier и All-to-All, различают время с шумом и без. При уровне шума 0,3% наблюдается достижение границы чувствительности трех из четырех рассмотренных конфигураций детектора. Так как границы доверительных интервалов с шумом 0,3% и без близки, для обнаружения агента системы мониторинга, который вносит дополнительную нагрузку 0,3% детекторы нужно использовать с осторожностью.

Таблица 3: Результаты тестирования чувствительности All-to-All 500.000 ит., 2 КВ, 4 узла

Уровень шума	Без шума	0,01%	0,05%	0,10%	0,30%	0,50%	0,70%	1,00%
Среднее значение	430,65	431,82	432,42	432,99	434,21	435,01	435,61	436,30
Стандартное отклонение	1,28	1,14	2,21	1,47	1,65	1,48	2,24	1,24
Количество запусков	22	10	3	10	7	8	10	22

Таблица 4: Результаты тестирования чувствительности All-to-All 250.000 ит., 2 КВ, 8 узлов.

Уровень шума	Без шума	0,01%	0,05%	0,10%	0,30%	0,50%	0,70%	1,00%
Среднее значение	515,88	517,35	522,65	519,23	519,30	526,46	528,95	526,40
Стандартное отклонение	1,63	3,43	0,73	4,29	1,12	0,89	2,66	3,28
Количество запусков	13	5	5	5	5	5	4	15

Таблица 5: Результаты тестирования чувствительности All-to-All 75.000 ит., 16 КВ, 4 узла.

Уровень шума	Без шума	0,50%	0,70%	1,00%
Среднее значение	552,94	560,13	561,15	561,04
Стандартное отклонение	2,16	0,35	0,66	2,74
Количество запусков	12	5	5	12

Таблица 6: Результаты тестирования чувствительности Barrier, 49.000.000 ит., 4 узла.

Уровень шума	Без шума	0,30%	0,50%	0,70%	1,00%
Среднее значение	546,66	551,58	552,80	555,19	555,90
Стандартное отклонение	1,03	1,66	0,75	0,85	2,09
Количество запусков	8,00	5,00	5,00	5,00	11

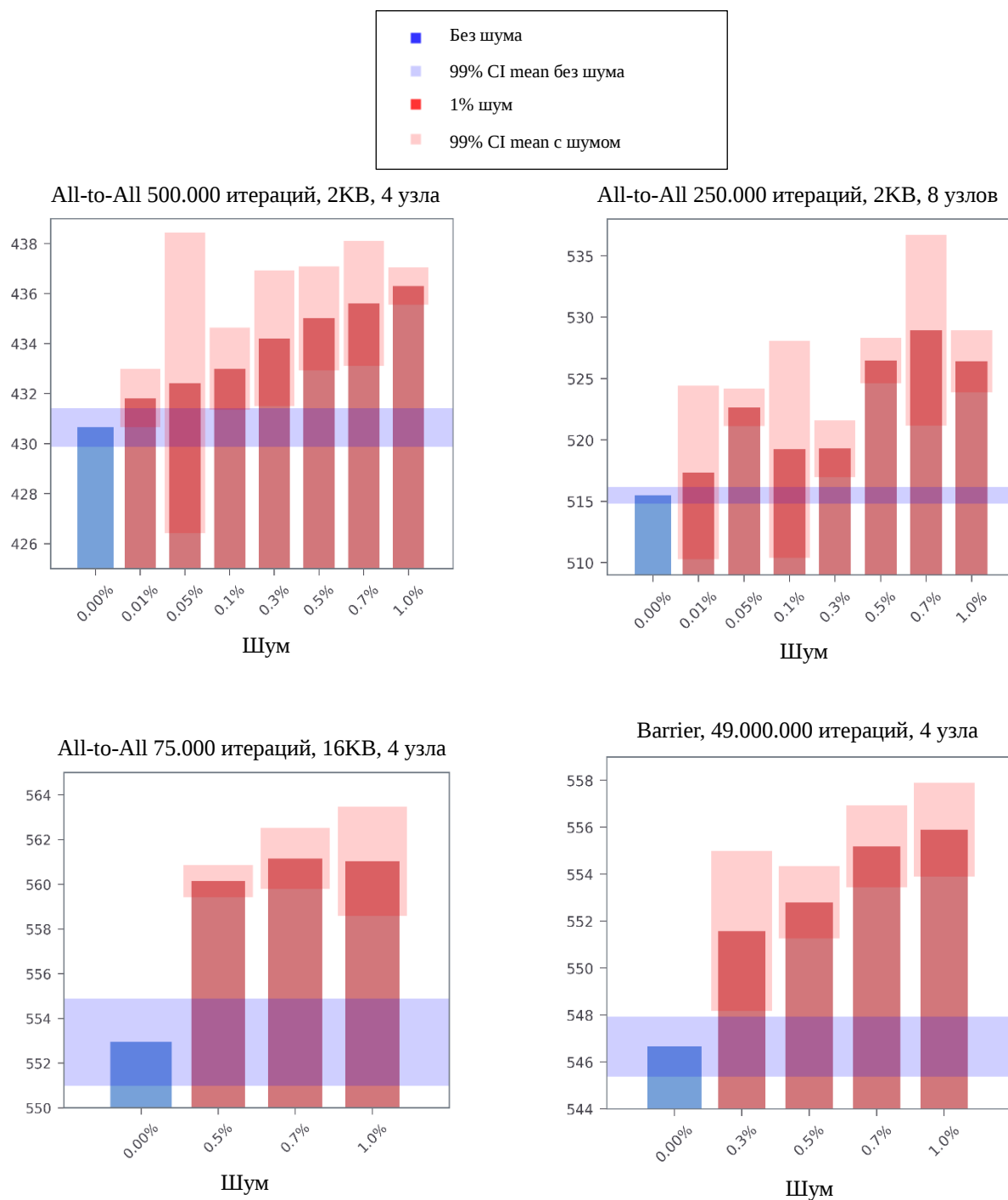


Рисунок 8: Тестирование чувствительности.

## **Заключение**

В результате работы были предложены два программных средства — источник шума, предназначенный для моделирования влияния работы агента системы мониторинга производительности, и детектор шума. В качестве детектора шума были опробованы программы с различными коллективными MPI операциями.

Для операции All-to-All проводились тесты с различной длиной сообщения. Оказалось, что наиболее стабильным из рассмотренных является детектор шума с длиной пересылаемого сообщения 2 КВ, протестированный на 4-х и 8-ми узлах суперкомпьютера. Детектор, использующий операцию Barrier, при большом числе итераций (время выполнения от 7 минут) также является чувствительным к шуму 1%.

Для операций Barrier и All-to-All обнаружены граница чувствительности к нагрузке, вносимой источником шума. Это значение составляет 0,3% шума. Эти коллективные операции являются подходящими для обнаружения влияния агента системы мониторинга производительности в указанных конфигурациях.



## Список цитируемой литературы

1. Sottile M.J., Minnich R.G. Supermon: a high-speed cluster monitoring system // Proceedings. IEEE International Conference on Cluster Computing. IEEE Comput. Soc, 2002. С. 39–46.
2. Mooney R., Schmidt K.P., Studham R.S. NWPerf: a system wide performance monitoring tool for large Linux clusters // 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No.04EX935). IEEE, 2004. С. 379–389.
3. Adhianto L. и др. HPCTOOLKIT: tools for performance analysis of optimized parallel programs // Concurrency and Computation: Practice and Experience. 2010. Т. 22, № 6. С. 685–701.
4. Performance Co-Pilot [Электронный ресурс]. URL: <http://pcp.io/> (дата обращения: 31.03.2020).
5. Treibig J., Hager G., Wellein G. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments // 2010 39th International Conference on Parallel Processing Workshops. IEEE, 2010. С. 207–216.
6. Rohl T. и др. LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance monitoring for the masses // 2017 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2017. Т. 2017–Sept. С. 781–784
7. Agelastos A. и др. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications // SC14: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2014. С. 154–165.
8. Petrini F., Kerbyson D.J., Pakin S. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q // ACM/IEEE SC 2003 Conference (SC'03). 2003.
9. Top500 [Электронный ресурс]. URL: <https://www.top500.org/> (дата обращения: 31.03.2020).
10. Top500 Summit [Электронный ресурс]. URL: <https://www.top500.org/system/179397> (дата обращения: 31.03.2020).
11. Top500 Sierra [Электронный ресурс]. URL: <https://www.top500.org/system/179398> (дата обращения: 31.03.2020).
12. Beckman P. и др. Benchmarking the effects of operating system interference on extreme-scale parallel machines // Cluster Computing. Springer US, 2008. Т. 11, № 1. С. 3–16.
13. A Large-Scale Study of MPI Usage in Open-Source HPC Applications
14. Vl. Voevodin, A. Antonov, D. Nikitenko, P. Shvets, S. Sobolev, I. Sidorov, K. Stefanov, Vad. Voevodin, S. Zhumatiy: Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community. In Journal: Supercomputing Frontiers and Innovations, Vol.6, No.2 (2019). pp.4–11. DOI:10.14529/jsfi190201
15. Open MPI [Электронный ресурс]. URL: <https://www.open-mpi.org/> (дата обращения: 31.03.2020)
16. D. Tsafir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick. System noise, os clock ticks, and fine-grained parallel applications. In ACM International Conference on Supercomputing, June 2005.
17. Hoefler, T., Belli, R.: Scientific benchmarking of parallel computing systems: Twelveways to tell the masses when reporting performance results. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, pp. 73:1–73:12. ACM, New York, NY, USA (2015). DOI:10.1145/2807591.2807644

18. Стефанов К.С. Система мониторинга производительности суперкомпьютеров // Вестник Пермского Национального исследовательского политехнического университета. Аэрокосмическая техника. No 39.2014. С. 17–34.