DOI: 10.55648/1998-6920-2024-18-1-29-39 УДК 004.451.23:004.451.36

# Система анализа оптимизации доступа к общей памяти для реализации стандарта PMIx *

К. Е. Крамаренко[1,2], А. Ю. Поляков[3], А. В. Ефимов[1,2]

[1] Институт физики полупроводников им. А. В. Ржанова СО РАН (ИФП СО РАН)
[2] Сибирский гос. унив. телекоммуникаций и информатики (СибГУТИ)
[3] NVIDIA

*Аннотация*: Рассмотрены алгоритмы синхронизации доступа к общей памяти с преобладанием операций чтения, которые характерны для реализаций PMIx. Проведен анализ схемы синхронизации *N*(mutex+signal), результаты показали увеличение пропускной способности примерно на 20 % в режиме «только чтение» и дальнейшее направление проведения оптимизации.

*Ключевые слова*: распределенные вычислительные системы, интерфейс управления процессами, PMI, PMIx, общая память, синхронизация доступа.

*Для цитирования*: Крамаренко К. Е., Поляков А. Ю., Ефимов А. В. Система анализа оптимизации доступа к общей памяти для реализации стандарта PMIx // Вестник СибГУТИ. 2024. Т. 18, № 1. С. 29–39. https://doi.org/10.55648/1998-6920-2024-18-1-29-39.



Статья поступила в редакцию 23.06.2023;
принята к публикации 03.12.2023.

## 1. Введение

В архитектуре распределенной вычислительной системы (РВС) используется несколько элементарных машин (ЭМ), которые соединены сетью [1]. Элементарная машина (ЭМ) является основным компонентом РВС и может быть процессорным ядром или конфигурацией вычислительных узлов с различными типами процессоров. В передовых РВС количество ЭМ может достигать десятков миллионов.

Основная цель РВС – обработка вычислительных задач, которые могут состоять из одной или нескольких параллельных программ (ПП). Для решения этих задач требуются ресурсы РВС, которые описываются метаданными. Менеджер ресурсов РВС назначает ЭМ каждой ветви ПП. Запуск задачи содержит несколько этапов, включая развертывание среды исполнения (СИ), запуск прикладных процессов и определение компонентов среды исполнения [2–5]. Для управления выполнением используется интерфейс управления процессами (PMI), а самая актуальная и стандартизированная версия этого интерфейса называется PMIx [6]. Информация о среде исполнения хранится в базе данных «ключ – значение» (KVDb), и для

* Работа выполнена в рамках государственного задания ИФП СО РАН (ГЗ FWGW-2022-0011).

доступа к ней используются примитивы чтения (Get), записи (Put) и синхронизации (Fence, Commit). Размещение KVDb в общей памяти вычислительного узла позволяет эффективно обращаться к ней без системных вызовов. В данной работе проведен анализ схемы синхронизации *N*(mutex+signal) и представлены описания существующих схем синхронизации доступа к общей памяти в реализациях стандарта PMIx. Работа является продолжением [21].

## 2. Синхронизация доступа к базе данных в стандарте PMIx

Компоненты стека системного программного обеспечения (ПО) в распределенной операционной системе, основанной на стандарте PMIx, включают следующие элементы: а) эталонная реализация PMIx (Open PMIx [7]); б) реализация среды исполнения (RTE); в) библиотека обмена сообщениями; и г) среда параллельного программирования (MPI [8], OpenSHMEM). На рис. 1 показано расположение этих компонентов на процессах распределенной операционной системы.

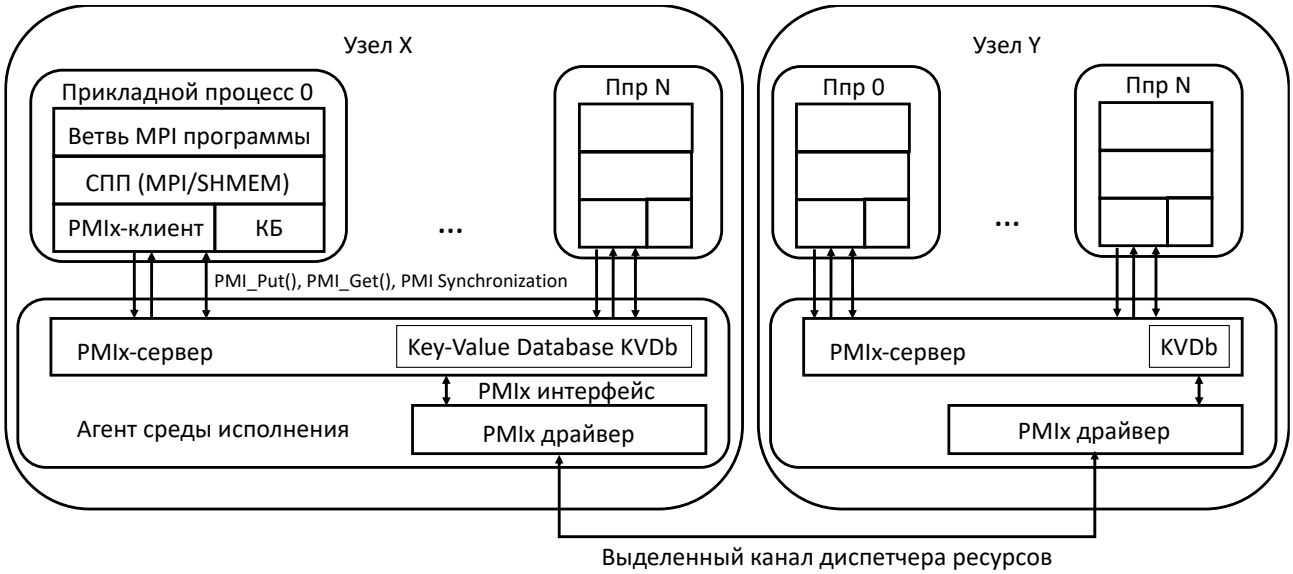



Рис. 1. Уровни системного программного обеспечения
(Ппр – прикладной процесс, СПП – среда параллельного программирования,
КБ – коммуникационная библиотека)

Прикладные процессы, составляющие задачу, взаимодействуют с системой СИ через интерфейс PMI [3, 4]. Это позволяет различным СИ совместимо работать с разными реализациями моделей и парадигм параллельного программирования. СИ может быть представлена как диспетчер ресурсов вычислительной системы [11], а также как специализированная библиотека [9, 10].

Агент СИ создается на каждом вычислительном узле, объединяющем несколько ЭМ общей оперативной памятью. Такой агент (рис. 1) включает в себя поток, выполняющий стандартные функции PMIx-сервера (реализован в библиотеке Open PMIx), и драйвер для взаимодействия с другими агентами, предоставляемый СИ. Прикладные процессы задачи включают реализацию клиентской части PMIx, предоставляемую Open PMIx.

Агент СИ обновляет базу данных PMI (KVDb), чтобы она всегда была актуальной. KVDb содержит как статическую, так и динамическую информацию о СИ. Статическая информация включает уникальный целочисленный идентификатор (ранг) процесса ПП, общее количество приложений и другие данные. Данная информация известна агенту СИ априори и не требует взаимодействия с агентами, расположенными на других узлах ВС.

Динамические параметры среды исполнения – это данные, которые специфичны для прикладных процессов. Эти данные могут включать информацию об адресах, получаемую от

коммуникационной подсистемы. Для работы динамического компонента необходима дополнительная синхронизация данных между агентами среды исполнения, которые могут располагаться на различных узлах вычислительной системы. Это может привести к накладным расходам на этапе инициализации вычислительной задачи.

PMIx отличается от предшественников PMI-1 и PMI-2 следующими важными особенностями [4]: а) значительное сокращение динамической информации о среде исполнения, за счет переноса большей ее часть в статический компонент, известный диспетчеру ресурсов; б) режим обмена данными по требованию.

Внутренний доступ к KVDb в рамках существующих реализаций PMI [12] является узким местом. Это прежде всего отражается на производительности операций Get и Put. Оптимизация доступа к KVDb путем его размещения в общей памяти является наиболее актуальным подходом, который дает следующие преимущества:

- Память KVDb масштабируется, что означает, что ее содержимое не дублируется на каждом процессе приложения.
- Общая память обеспечивает параллелизм, позволяя клиентам получать доступ к KVDb независимо от сервера.
- Доступ к операциям осуществляется с низкими накладными расходами, так как они выполняются без системных вызовов, которые требуют переключения контекста операционной системы.

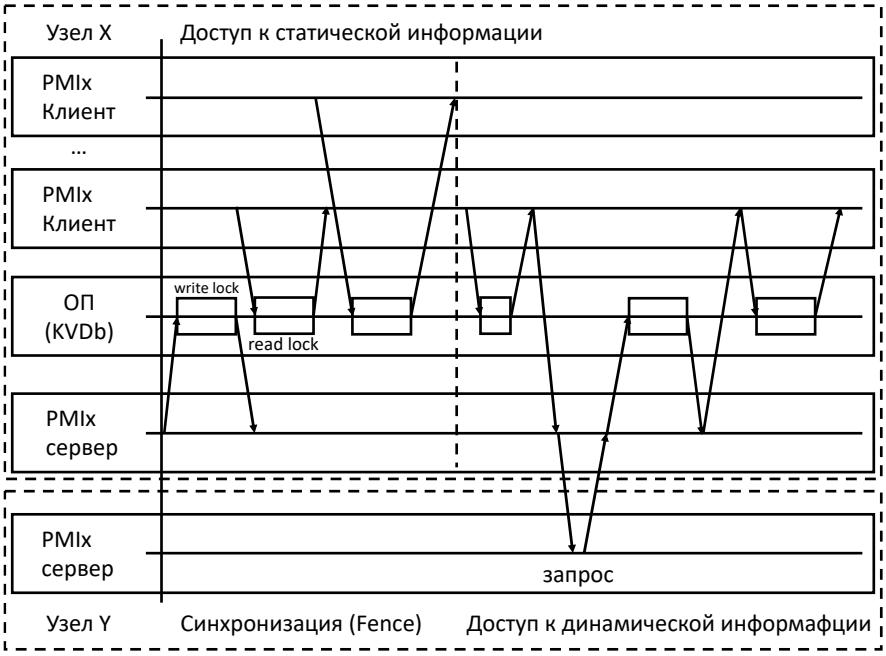



Рис. 2. Диаграмма доступа к KVDb

Для доступа к данным KVDb используются операции глобальной, локальной барьерной синхронизации процессов и операции чтения/записи. Во время выполнения синхронизации с помощью примитива "Fence" (см. рис. 2) поток "PMIx-сервер" блокирует общую память для записи (write lock) и помещает данные в KVDb. После завершения синхронизации приложения блокируют KVDb для чтения (read lock) и получают доступ к параметрам среды выполнения. Если запрашиваемый параметр отсутствует в локальной KVDb, блокировка на чтение снимается, и производится запрос к потоку СИ для получения данных, которые хранятся на других узлах ВС. Если агент СИ не выполняет операции записи в KVDb, приложения могут обращаться к ней одновременно. Для обеспечения такого доступа используются примитивы синхронизации [13–15].

## 3. Обзор существующих алгоритмов синхронизации доступа к оперативной памяти

Важной особенностью PMIx является параллельное и интенсивное чтение данных KVDb, выполняемое PMIx-клиентами, в то время как операция записи осуществляется относительно редко и только PMIx-сервером. Рассмотрим подробнее схемы синхронизации, обеспечивающие безопасный доступ к KVDb.

Первая схема (рис. 3) основана на примитиве «чтение – запись», предоставляемом стандартом POSIX [14]. Каждый клиент PMIx должен выполнить операцию захвата примитива синхронизации над одной и той же областью памяти для конкурентного доступа к KVDb при чтении. Это приводит к повышению нагрузки на подсистему обеспечения согласованности процессорных кэшей в узлах вычислительных систем, что приводит к увеличению накладных расходов [5].

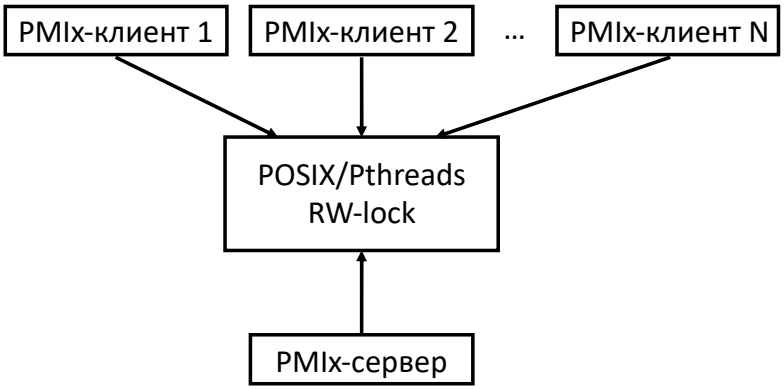



Рис. 3. Схема синхронизации на основе примитива «чтение – запись»

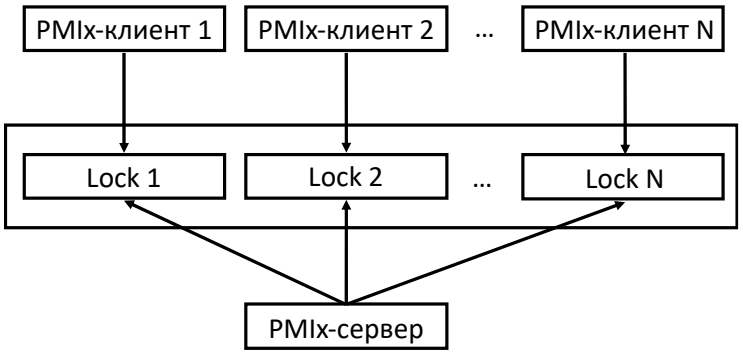



Рис. 4. Схема синхронизации 1*N*-mutex

В работе [15] предложена реализация схемы синхронизации 1*N*-mutex (рис. 4), которая не приводит к перегрузке подсистемы обеспечения когерентности процессорных кэшей. В схеме 1*N*-mutex для каждого клиента PMIx создается отдельный примитив синхронизации (mutex), при захвате которого гарантируется атомарный доступ к KVDb для чтения. Однако для записи требуется захват всех клиентских примитивов, что ограничивает масштабируемость операции в зависимости от количества клиентов PMIx.

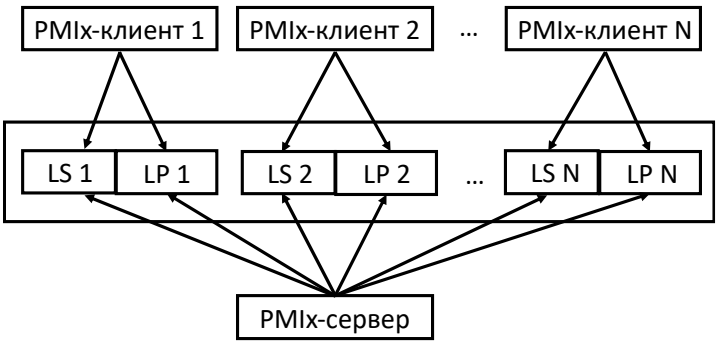



Рис. 5. Схема синхронизации 2*N*-mutex

В настоящее время в Open PMIx используется схема синхронизации 2*N*-mutex [5] (рис. 5). Каждому PMIx-клиенту соответствует пара синхронизационных примитивов (mutex). Первый примитив (LS*x*) служит сигнальным, а второй (LP*x*) – для доступа к оперативной памяти (ОП). При захвате примитива LP*x* PMIx-клиент освобождает примитив LSx. Для записи требуется захват всех сигнальных примитивов (LS1 – LS*N*), которые освобождаются PMIx-клиентами как можно быстрее. Затем сервер захватывает основные примитивы (LP1 – LP*N*), при этом благодаря блокировке сигнальных примитивов новые попытки доступа для чтения выполняться не будут.

## 4. Оптимизация алгоритмов синхронизации доступа к оперативной памяти

В данной работе предложена модификация схемы 2*N*-mutex, которая оптимизирует доступ к оперативной памяти в конкурентном режиме. Для снижения накладных расходов внесены изменения в способ сигнализации, который реализован в схеме 2*N*-mutex на основе примитива синхронизации LS*x*. В схеме синхронизации *N*(mutex+signal) (рис. 6) сигнальный примитив LS*x* (mutex) заменен переменной VS*x*, что позволяет сократить количество атомарных операций, необходимых для захвата примитива синхронизации при чтении, и повысить производительность.

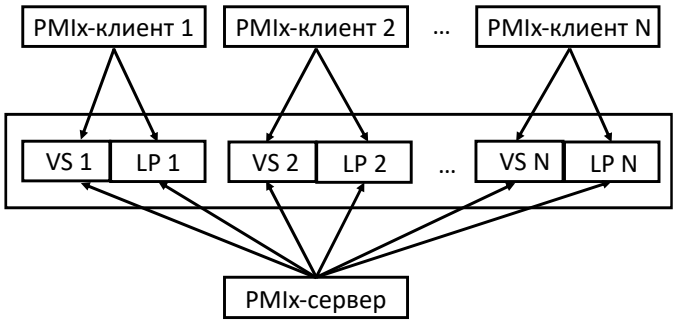



Рис. 6. Схема синхронизации *N*(mutex+signal)

На рис. 7 показан алгоритм блокировки примитивов на запись и чтение в схеме синхронизации *N*(mutex+signal). При доступе для записи (рис. 7а) все сигнальные переменные (VS1 – VS*N*) устанавливаются в значение «1». Затем PMIx-сервер последовательно ожидает освобождения и захватывает основные примитивы LP*i*. При захвате примитива синхронизации на чтение (рис. 7б) PMIx-клиент сначала проверяет состояние соответствующей сигнальной переменной (VS*x*). Если переменная установлена, то попытка захвата основного примитива синхронизации LP*x* не производится.

```
for i in 1 … N:
    VS[i] = 1
for i in 1 … N:
    lock(LP[i])
```

а

```
while(VS[x] = 1):
    noop();
lock(LP[x])
```

б

Рис. 7. Псевдокод захвата примитивов в схеме синхронизации *N*(mutex+signal) (PMIx-клиентом с номером x): (а) на запись; (б) на чтение

Стандарт POSIX Threads не предписывает соблюдать порядок доступа [16], поэтому использование сигнальных примитивов LS*x* в схеме 2*N*-mutex и сигнальных переменных VS*x* в схеме *N*(mutex+signal) и реализация примитива POSIX/mutex в библиотеке GNU C Library не обладает таким свойством.

## 5. Методика проведения испытаний

В данной работе разработан микробенчмарк [19], предназначенный для оценки эффективности схем синхронизации доступа к ОП. В рамках микробенчмарка реализованы функции проверки корректности схем синхронизации, а также нагрузочные тесты.

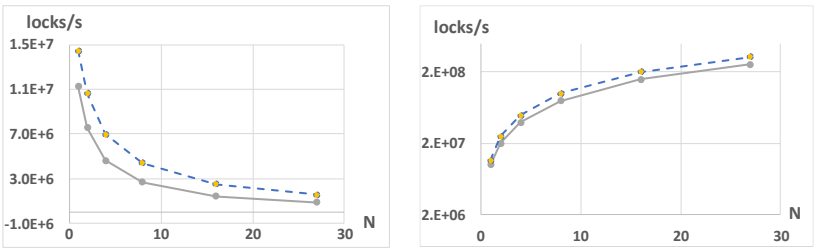
Для оценки эффективности схем синхронизации проводятся тесты в трех режимах: «только чтение», «только запись» и при конкурентном использовании. В последнем случае сервер выполняет заданное количество операций записи (в данной работе – 100 итераций), которые чередуются с ожиданием в 10 микросекунд. Во время работы сервера клиенты осуществляют непрерывный доступ для чтения, создавая помехи.

Эффективность схем синхронизации оценивается по следующим параметрам: пропускная способность в режиме «только чтение» (количество захваченных примитивов в секунду – кзп/с), пропускная способность в режиме «только запись» (кзп/с), пропускная способность в конкурентном режиме (кзп/с) и накладные расходы на конкурентный захват примитивов для записи (с).

Для проведения экспериментов были использованы две схемы синхронизации: 2*N*-mutex, которая используется в промышленной реализации стандарта PMIx, и *N*(mutex+signal), предложенная в данной работе. В качестве вычислительной базы использовались NUMA-узлы с процессорами Intel (2 x Xeon E5-2680 v4) с 28 ядрами. Тесты запускались с использованием команды *mpirun* и параметра "*--bind-to-core*", который гарантировал фиксированное назначение каждому потоку уникального вычислительного ядра.

Измерение эффективности схем синхронизации было проведено 10 раз для каждого значения *N*, после чего было вычислено среднее значение.

Результаты оценки производительности представлены на рис. 8.




а б

Рис. 8. Пропускная способность схем синхронизации:
а) в режиме «только чтение»; б) в режиме «только запись»;
— - 2*N*-mutex; - - - *N*(mutex+signal)

В предложенной схеме синхронизации удалось сократить количество атомарных операций, что привело к увеличению пропускной способности в режиме «только чтение» пример-

но на 20 %. В режиме «только запись» также наблюдается повышение пропускной способности на 3–5 %.

## 6. Реализация и анализ схемы синхронизации *N*(mutex+signal) в OpenPMIx

В работе была предложена схема синхронизации *N*(mutex+signal), которая была успешно реализована в Open-PMIx [20], соответствующей реализации стандарта PMI. Для анализа наиболее вычислительно сложных участков кода методов чтения и записи и последующей оптимизации был разработан программный инструмент. Оценка эффективности доступа к данным в KVDb производится следующим образом: функция чтения/записи данных в KVDb разбивается на несколько частей исходного кода в соответствии с задачами анализа. В начале и конце каждой части исходного кода вставляется функция, которая возвращает количество тактов, выполненных процессором с момента последней перезагрузки системы. Затем рассчитывается время выполнения каждой части и производится оценка вычислительной сложности.

Например, при анализе метода Get, функция Counter, возвращающая количество тактов, выполненных процессором, вставляется в соответствующие фрагменты исходного кода метода Get:

- При входе в метод Get (P1).
- Перед блокировкой примитива синхронизации (P2).
- После блокировки примитива синхронизации (P3).
- После нахождения ключа (P4).
- После распаковки ключа (P5).
- После разблокировки примитива синхронизации (P6).
- При выходе из метода Get (P7).

Чтобы получить результаты, нужно запустить тестовую программу, которая вызывает методы Get/Put из библиотеки OpenPMIx. После завершения программы значения количества тактов, затраченных на выполнение каждой части метода Get, записываются в лог-файл для последующего анализа.

## 7. Результаты

В данной работе был проведен анализ эффективности метода получения данных (Get) в KVDb для выявления наиболее вычислительно сложных участков кода.

В эксперименте использовались вычислительные узлы с процессорами Intel (2 x Xeon E5-2680 v4), имеющими 28 ядер. В качестве тестовой программы, вызывающей метод Get, была использована модифицированная утилита *pmix_intra_perf* (которая входит в состав исходного кода библиотеки OpenPMIx). Количество ключей, которые считывались из KVDb, составляло 100, при этом каждый ключ имел длину 50 символов. Количество клиентов *N*, считывающих ключи, варьировалось от 2 до 28.

Таблица 1. Результаты анализа метода Get

| | P7-P1 | P2-P1 | P3-P2 | P4-P3 | P5-P4 | P6-P5 | P7-P6 |
|---|---|---|---|---|---|---|---|
| $N = 2$ | | | | | | | |
| Среднее | 4182.085 | 931.845 | 284.815 | 1511.365 | 437.295 | 86.445 | 930.32 |
| Медиана | 3815.0 | 558.0 | 285.0 | 1569.0 | 330.0 | 75.0 | 961.0 |
| $N = 8$ | | | | | | | |
| Среднее | 4231.64 | 759.485 | 175.11 | 1832.575 | 404.16 | 113.68 | 946.62 |
| Медиана | 3673.5 | 636.0 | 126.0 | 1605.0 | 345.0 | 78.0 | 810.0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $N = 16$ | | | | | | | |
| Среднее | 4493.59 | 803.79 | 271.90 | 1769.84 | 435.91 | 209.24 | 1002.89 |
| Медиана | 4143.0 | 612.0 | 123.0 | 1674.0 | 351.0 | 84.0 | 822.0 |
| $N = 20$ | | | | | | | |
| Среднее | 4318.94 | 775.93 | 240.5 | 1735.19 | 441.96 | 143.48 | 981.85 |
| Медиана | 4068.0 | 600.0 | 123.0 | 1693.5 | 351.0 | 81.0 | 807.0 |

В табл. 1 представлены результаты анализа метода Get. В качестве значений указаны средние значения и медиана количества тактов процессора для 100 считанных ключей. Анализ полученных результатов показал, что около 50 % накладных расходов на выполнение метода Get приходится на инициализацию данных и их очистку. Для дальнейшей оптимизации необходимо сосредоточиться именно на этих участках кода.

## 8. Заключение

В работе проведен анализ схемы синхронизации *N*(mutex+signal). Полученные результаты показывают, что схема позволяет увеличить пропускную способность примерно на 20 % в режиме «только чтение» по сравнению со схемой 2*N*-mutex. Анализ метода Get при реализации схемы синхронизации *N*(mutex+signal) в OpenPMIx показал, что около 50 % накладных расходов приходится на инициализацию данных и их очистку. Поэтому дальнейшая оптимизация должна быть направлена именно на эти участки кода.

## Литература

1. *Хорошевский В. Г.* Архитектура вычислительных систем. М.: МГТУ им. Н. Э. Баумана, 2008. 520 с.
2. *Yu W., Wu J.,and Panda D. K.* Fast and Scalable Startup of MPI Programs in InfiniBand Clusters // Proc. 11th International Conference on High Performance Computing (HiPC), Bangalore, India, December 19–22, 2004. P. 440–449.
3. *Balaji P., Buntinas D., Goodell D., at. al.* PMI: A Scalable Parallel Processmanagement Interface for Extreme-scale Systems // Proc. 17th European MPI Users' Group Meeting Conference on Recent Advances in the Message Passing Interface (EuroMPI'10). Springer-Verlag, Berlin, Heidelberg. P. 31–41.
4. *Castain R. H., Solt D., Hursey J., and Bouteiller A.* PMIx: Process Management for Exascale Environments // Proc. 24th European MPI Users' Group Meeting (EuroMPI '17). ACM, New York, NY, USA, Article № 14, 10 p.
5. *Polyakov A., Karasev B., Hursey J. et. al.* A Performance Analysis and Optimization of PMIx-based HPC Software Stacks // Proc. 26th European MPI Users' Group Meeting, September 2019, Article № 9. P. 1–10.
6. PMIx Consortium. 2017–2020. Process Management Interface for Exascale (PMIx) Standard (version 3.1). [Электронный ресурс]. URL: `https://github.com/pmix/pmix-standard/releases/download/v3.1/pmix-standard-3.1.pdf` (дата обращения: 10.04.2023).
7. Open PMIx master repository [Электронный ресурс]. URL: `https://github.com/openpmix/openpmix` (дата обращения: 10.04.2023).
8. Message Passing Interface Forum. 2015. MPI-3.1: Official document. [Электронный ресурс]. URL: `http://www.mpi-forum.org/docs` (дата обращения: 10.04.2023).

9. Argonne National Laboratory. 2014. Hydra Process Management Framework. [Электронный ресурс]. URL: https://wiki.mpich.org/mpich/index.php/Hydra_Process_Management_Framework (дата обращения: 10.04.2023).
10. PMIx Reference RunTime Environment (PRRTE). [Электронный ресурс]. URL: https://github.com/openpmix/prrte (дата обращения: 10.04.2023).
11. SchedMD, LLC. 2017. SLURM Workload Manager. [Электронный ресурс]. URL: https://slurm.schedmd.com/ (дата обращения: 10.04.2023).
12. *Chakraborty S., Subramoni H., Perkins J., Panda D. K.* SHMEMPMI – Shared Memory Based PMI for Improved Performance and Scalability // Proc. 24th IEEE European MPI Users' Group Meeting (CCGrid). P. 60–69.
13. *Calciu I., Dice D., Lev Y., Luchangco V. et. al.* NUMA-aware reader-writer locks // Proc. 18th ACM SIGPLAN symposium on Principles and practice of parallel programming. ACM, New York, NY, USA, 2013. P. 157–166.
14. IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7. IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008) (Jan 2018). P. 1–3951.
15. *Hsieh W. C., Weihl W. E.* Scalable Reader-Writer Locks for Parallel Systems // Proc. Sixth IEEE International Parallel Processing Symposium, 1992. P. 656–659.
16. IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7. IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008) (Jan 2018). P. 1–3951. [Электронный ресурс]. URL: https://doi.org/10.1109/ IEEESTD.2018.8277153 (дата обращения: 01.09.2023).
17. *Mellor-Crummey J., Scott M.* Algorithms for scalable synchronization on shared-memory multiprocessors // ACM Trans. Comput. Syst. 1991. № 9. P. 21–65.
18. *Mellor-Crummey J.* Algorithms for Scalable Lock Synchronization on Shared-memory Multiprocessors. Department of Computer Science Rice University. Lecture 18. 17 March 2009. P. 1–49. [Электронный ресурс]. URL: https://cs.anu.edu.au/courses/comp8320/lectures/aux/comp422-Lecture18-HWLocks.pdf (дата обращения: 10.04.2023).
19. Микробенчмарк. [Электронный ресурс]. URL: https://bitly.su/Wtxe0Z (дата обращения: 05.02.2023).
20. OpenPMIx. [Электронный ресурс]. URL: https://shorturl.at/kwx14 (дата обращения: 16.04.2023).
21. *Ефимов А. В., Поляков А. Ю., Крамаренко К. Е., Бочкарев Б. В.* Оптимизация доступа к общей памяти для реализации стандарта PMIx // Вестник СибГУТИ. 2020. № 4. С. 28–38.

**Крамаренко Константин Евгеньевич**
инженер-программист ИФП СО РАН; ст. преподаватель СибГУТИ, тел. +7 383 2698 286, e-mail: kostya.kram@gmail.com, ORCID ID: 0000-0002-3844-5481.

**Поляков Артём Юрьевич**
старший архитектор программного обеспечения NVIDIA Corporation (Santa Clara, California, USA), e-mail: artemp@nvidia.com, ORCID ID: 0000-0001-6759-1448.

**Ефимов Александр Владимирович**
ведущий инженер-программист ИФП СО РАН (630090, Новосибирск, пр-кт. Ак. Лаврентьева, 13);
к.т.н., доцент СибГУТИ (630102, Новосибирск, ул. Кирова, 86), тел. +7 383 2698 293, e-mail: alexandr.v.efimov@gmail.com, ORCID ID: 0000-0002-8272-9924.

*Авторы прочитали и одобрили окончательный вариант рукописи.*
*Авторы заявляют об отсутствии конфликта интересов.*
Вклад соавторов: *Каждый автор внес равную долю участия как во все этапы проводимого теоретического исследования, так и при написании разделов данной статьи.*

## Shared Memory Access Optimization Analysis System for PMIx Standard Implementation

Konstantin E. Kramarenko, Artem Y. Polyakov, Alexander V. Efimov

[1] The Institute of Semiconductor Physics, Siberian Branch of the Russian Academy of Sciences (ISP SB RAS)
[2] Siberian State University of Telecommunications and Information Science (SibSUTIS)
[3] NVIDIA

*Abstract*: The results in the field of system software stacks optimization for distributed computing systems based on the process management Interface (PMI) are obtained. Algorithms for synchronizing access to shared memory with a predominance of reading operations which are typical for PMIx implementation,are considered. The locking schemes *N*(mutex+signal) are proposed.

*Keywords*: distributed computing systems, resource management, process management interface, PMI, PMIx, shared memory.

*For citation*: Kramarenko K. E., Polyakov A. Y., Efimov A. V. Shared memory access optimization analysis system for PMIx standard implementation (in Russian). *Vestnik SibGUTI*, 2024, vol. 18, no. 1, pp. 29-39. https://doi.org/10.55648/1998-6920-2024-18-1-29-39.



The article was submitted: 23.06.2023;
accepted for publication 03.12.2023.

### References

1. Khoroshevskii V. G. *Arkhitektura vychislitel'nykh sistem* [Architectures of computer systems]. Moscow, MGTU im. Baumana, 2008. 520 p.
2. Yu W., Wu J., Panda D. K. Fast and Scalable Startup of MPI Programs in InfiniBand Clusters. *Proc. 11th International Conference on High Performance Computing (HiPC)*, Bangalore, India, 19-22 December, 2004, pp. 440-449.
3. Balaji P., Buntinas D., Goodell D., at. al. PMI: A Scalable Parallel Processmanagement Interface for Extreme-scale Systems. *Proc. 17th European MPI Users' Group Meeting Conference on Recent Advances in the Message Passing Interface (EuroMPI'10)*, Springer-Verlag, Berlin, Heidelberg, pp. 31-41.
4. Castain R. H., Solt D., Hursey J., Bouteiller A. PMIx: Process Management for Exascale Environments. *Proc. 24th European MPI Users' Group Meeting (EuroMPI '17)*. ACM, New York, NY, USA, Article no. 14, 10 p.
5. Polyakov A., Karasev B., Hursey J. et. al. A Performance Analysis and Optimization of PMIx-based HPC Software Stacks. *EuroMPI '19: Proceedings of the 26th European MPI Users' Group Meeting*, September, 2019, article no. 9, pp. 1-10.
6. *PMIx Consortium. 2017–2020. Process Management Interface for Exascale (PMIx) Standard (version 3.1)*, available at: https://github.com/pmix/pmix-standard/releases/download/v3.1/pmix-standard-3.1.pdf (accessed: 10.04.2023).
7. *Open PMIx master repository*, available at: https://github.com/openpmix/openpmix (accessed: 10.04.2023).

8. *Message Passing Interface Forum. 2015. MPI-3.1: Official document*, available at: http://www.mpi-forum.org/docs (accessed: 10.04.2023).
9. *Argonne National Laboratory. 2014. Hydra Process Management Framework*, available at: https://wiki.mpich.org/mpich/index.php/Hydra_Process_Management_Framework (accessed: 10.04.2023).
10. *PMIx Reference RunTime Environment (PRRTE)*, available at: https://github.com/openpmix/prrte (accessed: 10.04.2023).
11. *SchedMD, LLC. 2017. SLURM Workload Manager*, available at: https://slurm.schedmd.com/ (accessed: 10.04.2023).
12. Chakraborty S., Subramoni H., Perkins J., Panda D. K. SHMEMPMI – Shared Memory Based PMI for Improved Performance and Scalability. *Proc. 24th IEEE European MPI Users' Group Meeting (CCGrid)*, pp. 60-69.
13. Calciu I., Dice D., Lev Y., Luchangco V. et. al. NUMA-aware reader-writer locks. *Proc. 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, ACM, New York, NY, USA, 2013, pp. 157-166.
14. IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, iss. 7. *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*, Jan, 2018, pp. 1-3951.
15. Hsieh W. C., Weihl W. E. Scalable Reader-Writer Locks for Parallel Systems. *Proc. Sixth IEEE International Parallel Processing Symposium*, 1992, pp. 656-659.
16. IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, iss. 7. *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*, Jan, 2018, pp. 1-3951, available at: https://doi.org/10.1109/ IEEESTD.2018.8277153 (accessed: 01.09.2023).
17. Mellor-Crummey J., Scott M. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Trans. Comput. Syst*, 1991, no. 9, pp. 21-65.
18. Mellor-Crummey J. *Algorithms for Scalable Lock Synchronization on Shared-memory Multiprocessors*. Department of Computer Science Rice University, Lecture 18. 17 March, 2009, pp. 1-49. available at: https://cs.anu.edu.au/courses/comp8320/lectures/aux/comp422-Lecture18-HWLocks.pdf (accessed: 10.04.2023).
19. *Mikrobenchmark* [Microbenchmark], available at: https://bitly.su/Wtxe0Z (accessed: 05.02.2023).
20. *OpenPMIx*, available at: https://shorturl.at/kwx14 (accessed: 16.04.2023).
21. Efimov A. V., Plyakov A. U., Kramarenko K. E., Bochkarev B. V. Optimizatsiya dostupa k obshchei pamyati dlya realizatsii standarta PMIx [Optimization of access to shared memory for the implementation of the PMIx standard]. *Vestnik SibGUTI*, 2020, no. 4, pp. 28-38.

**Konstantin E. Kramarenko**
Software engineer of ISP SB RAS;
Senior lecturer, SibSUTIS (Russia, 630102, Novosibirsk, Kirov St. 86), tel. + 7 383 2698 286, e-mail: kostya.kram@gmail.com, ORCID ID: 0000-0002-3844-5481.

**Artem Yu. Polyakov**
Senior software architect of NVIDIA Corporation (Santa Clara, California, USA), e-mail: artemp@nvidia.com, ORCID ID: 0000-0001-6759-1448.

**Alexander V. Efimov**
Senior software engineer ISP SB RAS (630090, Novosibirsk);
Ph.D., docent, SibSUTIS (Russia, 630102, Novosibirsk, Kirov St. 86**)**, tel. +7 383 2698 293, e-mail: alexandr.v.efimov@gmail.com, ORCID ID: 0000-0002-8272-9924.