Повышение эффективности модели доступа к удаленной памяти MPI для систем с распределенной памятью путем реализации односторонней рассылки

М. Абуэльсауд[1], А. А. Пазников[2]

Санкт-Петербургский электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

[1]welleamy@gmail.com, [2]apaznikov@gmail.com

Аннотация. **В настоящее время обработка больших объемов расширяющихся данных эффективно и последовательно представляет собой значительное вызов. Традиционные высокопроизводительные компьютеры с распределенной памятью (HPC), основанные на модели передачи сообщений, сталкиваются с встроенными трудностями синхронизации, что ограничивает их способность следовать за темпом развития. Удаленный доступ к памяти (RMA, также известный как односторонние обмены MPI) позволяет процессу напрямую читать из памяти другого процесса или записывать в нее, обходя необходимость в обмене сообщениями. К сожалению, в текущем стандарте MPI RMA отсутствует интерфейс коллективных операций. Тем не менее, RMA имеет потенциал снижать затраты на синхронизацию, позволяя одновременный доступ к общим структурам данных, распределенным среди памяти процессов MPI. Существующие стандарты односторонних обменов MPI предлагают только линейный интерфейс, который затрудняет параллелизацию и далек от эффективности. Для заполнения этой пробела мы предлагаем алгоритмическое проектирование для эффективных коллективных (параллелизуемых) операций в парадигме RMA. Наше исследование в первую очередь рассматривает выгоды коллективных операций, используя в качестве примера алгоритм рассылки. Наши реализации превосходят традиционные методы, демонстрируя многообещающий потенциал этой техники, как показывают дальнейшие тесты производительности.**

Ключевые слова: MPI, высокопроизводительные вычисления, параллельное программирование, удаленный доступ к памяти, MPI RMA, модель общей памяти MPI

I. Введение

В высокопроизводительных вычислениях (HPC) множество параллельных приложений зависят от коллективных операций, осуществляемых с помощью интерфейса передачи сообщений (MPI) для распределенных коммуникационных задач, таких как рассылка и сокращение обмена сообщениями. Исследования, проведенные в операционных условиях HPC, демонстрируют, что коллективные операции MPI могут потреблять более двух третей времени на коммуникацию в приложениях MPI, часто становясь основными источниками узких мест в производительности [1, 2].

В рамках MPI удаленный доступ к памяти (RMA) предлагает эффективную среду для обмена данными между процессами, обходя прямое вовлечение процессора. Хотя этот метод доказывает свою эффективность во множестве сценариев, он сталкивается с трудностями при обработке операций рассылки, когда один процесс распространяет данные на несколько получателей. Традиционные реализации MPI обычно требуют двусторонней связи для каждого процесса, участвующего в рассылке, что может привести к неэффективности и увеличенной задержке, особенно в обширных развертываниях [3]. Более того, интеграция операций RMA приводит к значительному снижению затрат, поскольку она устраняет расходы, связанные с сопоставлением тегов, управлением предварительным прибытием сообщений и сложностями, присущими буферизации в системах передачи сообщений точка-точка [4, 5].

Выполнение программ часто требует множественных идентичных операций удаленного доступа к памяти (RMA) на различных сегментах памяти [5–8] Однако текущий стандарт MPI [9] ограничивает пользователей простыми линейными (последовательными) алгоритмами, которые далеки от оптимальных, когда семантика программы позволяет параллельный доступ к данным. Таким образом, основной целью этого исследовательского проекта является разработка алгоритмов для коллективных операций (collectives) в рамках модели RMA. Ожидается, что эти алгоритмы значительно превзойдут линейные алгоритмы, что приведет к сокращению времени выполнения и энергопотребления в программах MPI. Для иллюстрации концепций и принципов мы сосредотачиваемся на алгоритме рассылки (one-to-all), который широко используется в схемах коммуникации. Выбор этого алгоритма основан на его относительной простоте в понимании и реализации. В этом подходе один процесс (корневой) передает данные всем остальным процессам, участвующим в коллективной операции. Определившись с простым алгоритмом, мы стремимся облегчить анализ и отладку, что позволит легче выявлять потенциальные проблемы или узкие места. Изучение сложностей проектирования коллективных операций в рамках модели MPI RMA представляет уникальные вызовы по сравнению с традиционными интерфейсами передачи

сообщений. Среди этих вызовов – необходимость синхронизации данных, где процессы должны координировать и синхронизировать свой доступ к общим данным в удаленной памяти во время коллективных операций на основе RMA. Обеспечение правильной синхронизации и предотвращение конфликтов данных становится сложным, когда несколько процессов одновременно получают доступ и изменяют общие данные. Решение этих проблем требует тщательного анализа и разработки эффективных стратегий для достижения эффективной и конфликтно-свободной синхронизации данных в контексте коллективных операций в рамках модели MPI RMA.

Данное исследование направлено на изучение модели MPI для коллективных операций, ее особенностей и интеграции в распределенные системы памяти. Мы исследуем основные концепции, лежащие в основе этого подхода, рассмотрим его потенциальные преимущества и недостатки, и обсудим стратегии преодоления его вызовов. Кроме того, мы исследуем будущие направления модели MPI RMA для коллективных операций в контексте улучшения парадигм HPC и предоставим примеры реального использования [10].

II. Рассылка в общей памяти RMA

Этот раздел исследует эффективный способ реализации алгоритма рассылки с использованием модели удаленного доступа к памяти (RMA). Рассылка выбрана, поскольку она является как фундаментальной, так и часто используемой в параллельном программировании. В операции рассылки данные изначально находятся в памяти одного процесса. Затем операция реплицирует эти данные для всех процессов, участвующих в том же MPI-коммуникаторе. В рамках модели RMA все участвующие процессы должны разделять одиночный объект окна RMA, названный 'win', с выделенной достаточной памятью для хранения передаваемых данных [4]. Спецификация MPI позволяет использовать возможности сетевого оборудования для односторонней связи. При использовании удаленного доступа к памяти (RMA) в MPI группы процессов, инкапсулированные внутри MPI-коммуникатора, связываются с MPI-окнами. Эти окна представляют собой области памяти, из которых могут быть считаны или в которые могут быть записаны удаленные узлы. Другими словами, операции RMA выполняются в рамках MPI-окна, облегчая передачу данных между локальным процессом MPI (исходным) и удаленным процессом MPI (целью). Интерфейс RMA включает поддержку операций чтения (MPI_Get()), записи (MPI_Put()) и атомарных операций с памятью (MPI_Accumulate()) для данных, предоставленных внутри MPI-окна, показанных на рисунке 1 [11].

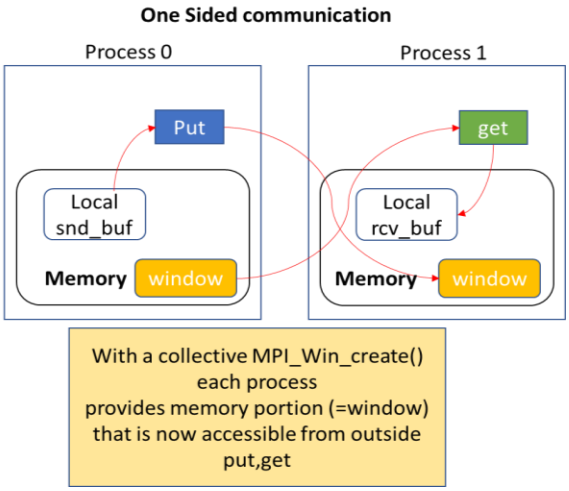

Рис. 1. Идет операция RMA Put. Процесс 0 записывает данные в память процесса 1

В рамках фреймворка Message Passing Interface (MPI), коммуникация через общую память облегчает обмен данными между процессами, находящимися на одном вычислительном узле, путем прямого доступа к области общей памяти. Этот подход заметно повышает эффективность коммуникации, особенно когда процессы находятся на одной физической машине, как показано на рис. 2, благодаря врожденной низкой задержке и высокой пропускной способности, связанных с архитектурами общей памяти [12].

- Выделение памяти: MPI предоставляет процедуры для выделения областей общей памяти, доступных для нескольких процессов. Например, можно использовать MPI_Win_allocate shared для выделения окна общей памяти.
- Доступ к памяти: После выделения области общей памяти, процессы могут напрямую читать из этого пространства памяти и записывать в него. Этот прямой доступ исключает необходимость в явной передаче сообщений через сеть, что снижает накладные расходы на коммуникацию.
- Освобождение памяти: Когда область общей памяти больше не нужна, MPI предоставляет процедуры для освобождения памяти. Например, можно использовать MPI_Win_free для освобождения окна общей памяти.

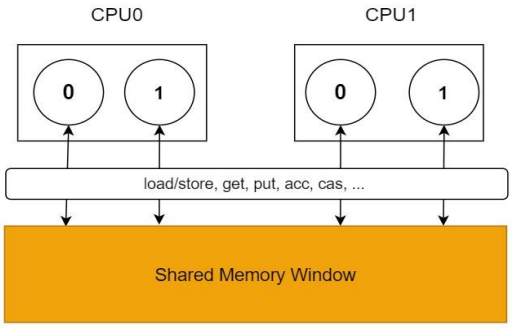

Рис. 2. Расширение общей памяти между процессами с использованием MPI RMA и окно общей памяти выделяется в каждом узле

A. Бинарное дерево

Алгоритм бинарного дерева обеспечивает эффективный метод трансляции данных в средах Message Passing Interface (MPI). В этом разделе описывается его реализация и анализируется его временная сложность. В начальном раунде, изображенном на рис. 4, процесс 0 сохраняет буфер b в памяти процессов 1 и 2. Затем, во втором раунде, процесс 1 передает буфер в память процессов 3 и 4. Во время третьего раунда процесс 2 помещает сообщение в буфер процессов 5 и 6. Наконец, в четвертом раунде процесс 3 перемещает сообщение в память процесса 7 [13].

С каждым раундом количество активно транслирующих узлов удваивается. Этот экспоненциальный рост приводит к значительному сокращению числа этапов передачи 2. В результате, последний процесс, получающий данные (например, ранг 7 на рис. 3), требует только $\log_2(p)$ раундов коммуникации для завершения трансляции. В алгоритме RMA Bcast Binary 1:

Функция сначала переупорядочивает ранги процессов, представляя структуру бинарного дерева, где корневой процесс имеет ранг 0.

- Он вычисляет новый ранг (rank), вычитая ранг корневого процесса root из текущего ранга процесса (my_rank), а затем берет модуль от общего числа процессов (nproc). Это гарантирует, что новый ранг находится в диапазоне рангов процессов на строке (1).
- Он вычисляет ранги дочерних процессов (child1 и child2) в структуре бинарного дерева на основе текущего ранга процесса на строках (4,5).
- Для каждого дочернего процесса (child1 и child2) функция проверяет, меньше ли ранг дочернего процесса общего числа процессов (p) на строках (6,12).
- Если существует дочерний процесс, он отображает виртуальный ранг дочернего процесса на реальный ранг, добавляя ранг корневого процесса и беря модуль от общего числа процессов на строках (7,13).
- Блокировка окна MPI, связанного с дочерним процессом, на строках (8,14).
- Затем вызывается операция MPI_Put для отправки данных из src_buf (9,15).
- Разблокировка окна MPI, связанного с дочерним процессом, на строке (10,16).

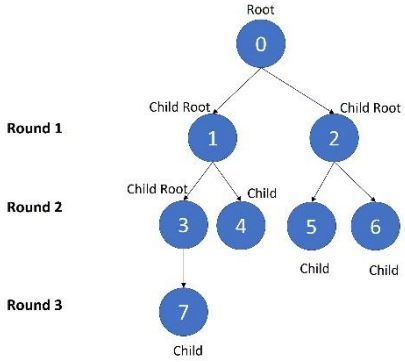

Рис. 3. Иллюстрация передачи данных бинарного дерева между 8 процессами

Input data:	*src_buf* − буфер происхождения *win* − окно RMA *comm* − коммуникатор
1	*srank* = comp_srank(*myrank, root, p*)
2	**for** *r* = 0 **to** *p* **do**
3	*rank* = **comp_rank**(*rank, root, p*)
4	*child1* = 2 ∗ *rank* + 1
5	*child2* = 2 ∗ *rank* + 2
6	**if** *child1* < *p* **then**
7	*child1* = comp_rank(*rank, root, p*)
8	MPI_Win_lock(MPI_LOCK_SHARED,win);
9	MPI_Put(*src_buf, length, datatype, child1*, win);
10	MPI_Win_unlock(*child1, win*);
11	**end if**
12	**if** *child2* < *p* **then**
13	*child2* = comp_rank(*rank, root, p*)
14	MPI_Win_lock(MPI_LOCK_SHARED,win);
15	MPI_Put(*src_buf, length, datatype, child2*, win);
16	MPI_Win_unlock(*child2, win*);
17	**end if**
18	**end for**

Алгоритм 1 RMA Bcast binary

Ограничение такое же, как у алгоритма биномиального дерева, упомянутого в подразделе II-B

B. Биномиальное дерево

В этом разделе подробно описывается реализация алгоритма биномиального дерева для трансляции данных в среде Message Passing Interface (MPI), используя возможности Remote Memory Access (RMA). Алгоритм использует итерационные раунды, число которых определяется логарифмом по основанию 2 от общего числа процессов (p). Каждый раунд включает в себя специфические операции обмена данными на основе рангов процессов [14], [15], [16], [17]. Рассмотрим сценарий с восемью процессами (p = 8). Алгоритм разворачивается на протяжении трех раундов:

- **Раунд 1:** Процесс 0 (корень) использует операции MPI RMA для размещения буфера данных в область общей памяти, доступную для процесса 1.
- **Раунд 2:** Процесс 0 снова использует MPI RMA для размещения буфера данных в области общей памяти как процесса 1, так и процесса 2.
- Раунд 3:

  – Процесс 0 использует MPI RMA для размещения буфера данных в области общей памяти процессов 1, 2 и 4.

  – Процесс 1, получивший данные в предыдущих раундах, использует MPI RMA для

размещения буфера данных в области общей памяти процессов 3 и 5.

– Аналогично, процесс 2 использует MPI RMA для размещения буфера данных в области общей памяти процесса 6.

– Наконец, процесс 6, получивший данные в предыдущем раунде, использует MPI RMA для размещения буфера данных в области общей памяти процесса 7.

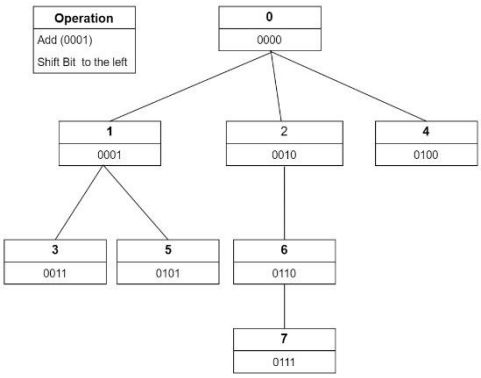

Рис. 4. Визуализация биномиального дерева с 8 процессами, использующими битовый сдвиг для эффективной коммуникации.

Временная сложность алгоритма биномиального дерева с использованием MPI RMA является алгоритмом putloop 2 Анализ сложности:

- На строке 3 цикл выполняется, пока mask < N, и mask удваивается на каждой итерации, тогда количество итераций $\log_2(N)$.
- На строке 9 внутри цикла выполняется операция MPI_Put для переноса данных из src_buf, которая выполняет операцию копирования размером с длину сообщения
- Пусть M представляет длину сообщения, тогда общая сложность составляет: O(Mlog(N)).

Input data:	*src_buf* − буфер происхождения *win* − окно RMA *comm* − коммуникатор
1	*srank* = comp_srank(*myrank, root, p*)
2	*mask* = 1
3	**while** *mask* < *p* **do**
4	**if** (*srank*&*mask*) == 0 **then**
5	*rank* = *srank* | *mask*
6	**if** *rank* < *p* **then**
7	*rank* = comp_rank(*rank, root, p*)
8	MPI_Win_lock(MPI_LOCK_SHARED,win);
9	MPI_Put(*src_buf, length, datatype, rank,* win);
10	MPI_Win_unlock(*child2, win*);
11	**else**
12	**break**
13	**end if**
14	*mask* = *mask* << 1
15	**end if**
16	**end while**

Алгоритм 2 put loop

Анализ сложности алгоритма RMA_Bcast_binomial:

- В алгоритме 3 на строке 1 цикл выполняется n раз, при каждой итерации вызывается функция putloop.
- Каждый вызов sendloop имеет сложность *O(Mlog(N)).*
- Полная сложность составляет $O(N \times M \times log(N))$.

$$T_{loop} = O(N \times M \times log(N)) \quad (1)$$

$$T_{total} = N \times T_{loop} \quad (2)$$

Из (1) и (2)

$$T_{total} = N \times O(N \times M \times log(N)) \quad (3)$$

Input data:	*src_buf* − буфер происхождения *win* − окно RMA *comm* − коммуникатор
1	**for** *r* = 0 **to** *p* **do**
2	*root* = *r*
3	put_loop(*src_buf, dest_buf, rank,root, win, comm*)
4	**end for**

Алгоритм 3 RMA Bcast binomial

Ограничения алгоритма биномиального дерева:

- Производительность алгоритма может ухудшиться при большом количестве процессов из-за увеличения накладных расходов на коммуникацию и синхронизацию.
- Этот подход может быть эффективным для небольшого и среднего количества процессов, но может столкнуться с увеличением задержек и накладными расходами при большом количестве процессов.
- Алгоритм зависит от наличия достаточного объема памяти и пропускной способности сети для обмена данными.

III. Экспериментальная оценка

В рамках данного исследования эксперименты проводились на вычислительном кластере, размещенном на платформе Yandex Cloud. Кластер использовал процессоры Intel Ice Lake, конкретно Intel® Xeon® Gold 6338 с тактовой частотой 2.00 ГГц. Каждый экземпляр виртуальной машины в составе кластера был сконфигурирован с 4 виртуальными центральными процессорами (vCPU), 8 ГБ оперативной памяти (RAM) и 20 ГБ дискового пространства.

Для создания бенчмарка были использованы вычислительные нагрузки, соответствующие нашим исследовательским контекстам. Мы провели тесты на сервере, включающие обработку 1000 сообщений, распределенных по восьми отдельным пакетам данных. Размеры этих пакетов были увеличены последовательно в двоичном формате, начиная с минимального значения 16 байт и удваиваясь на каждом шаге, пока не был достигнут пиковый размер в 33 мегабайта.

Согласно графическому представлению, показанному на рис. 5, алгоритм бинарного дерева демонстрирует превосходную производительность, за которым следует алгоритм биномиального дерева, а затем линейный алгоритм. Например, использование алгоритма бинарного дерева позволяет передавать данные двум процессам примерно за 0,01 миллисекунды, в то время как использование линейного алгоритма требует примерно 0,1 миллисекунды для выполнения той же задачи.

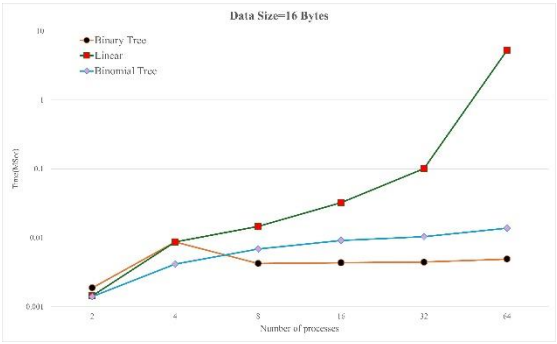

Рис. 5. Время рассылки в зависимости от выбора алгоритма (16 байт).

Как показано на рис. 6, алгоритм бинарного дерева демонстрирует превосходную производительность, за которым следует алгоритм биномиального дерева, а затем линейный алгоритм. Например, передача данных 32 процессам при использовании алгоритма бинарного дерева требует примерно 1 миллисекунды, в то время как использование линейного алгоритма для той же задачи требует примерно 10 миллисекунд.

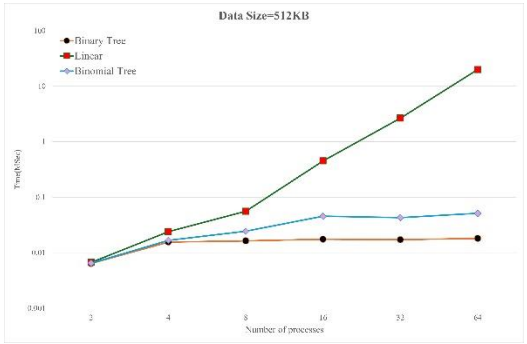

Рис. 6. Время рассылки в зависимости от выбора алгоритма (512 КБ данных)

Исходя из рис. 7, алгоритм бинарного дерева показывает наивысшую производительность, за которым следует алгоритм биномиального дерева, а затем линейный алгоритм. Например, передача данных 16 процессам с использованием алгоритма бинарного дерева занимает примерно 10 миллисекунд, в то время как для выполнения той же задачи с использованием линейного алгоритма требуется примерно 100 миллисекунд.

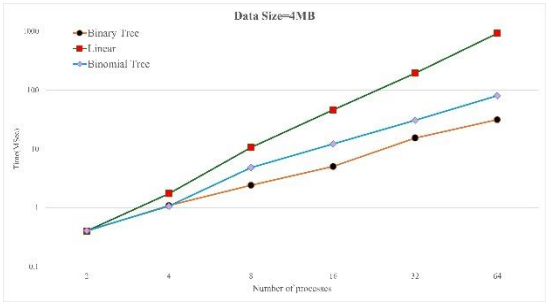

Рис. 7. Время рассылки в зависимости от выбора алгоритма (4 МБ данных)

Рис. 8 показывает время (измеренное в миллисекундах), необходимое для рассылки 32 МБ данных всем процессам с использованием трех различных алгоритмов: бинарного дерева, линейного и биномиального дерева. На оси x представлено количество процессоров, а на оси y – время (в миллисекундах) для рассылки данных.

Исходя из данных на рис. 8, алгоритм бинарного дерева демонстрирует наиболее эффективную производительность, за которым следует алгоритм биномиального дерева, и, наконец, линейный алгоритм. Например, рассылка данных 64 процессам с использованием алгоритма бинарного дерева занимает примерно 100 миллисекунд, в то время как для выполнения той же задачи с использованием линейного алгоритма требуется примерно 1000 миллисекунд.

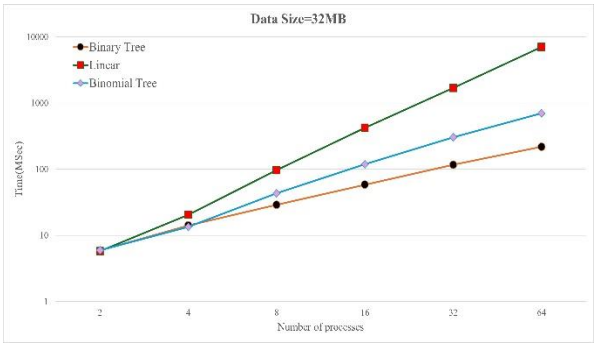

Рис. 8. Производительность рассылки с большим объемом данных (32 МБ): Бинарное дерево против Линейного против Биномиального дерева.

IV. Заключение

Исследование прояснило эффективность нескольких алгоритмов рассылки при улучшении модели удаленного доступа к памяти MPI для распределенных систем памяти, с особым упором на реализацию односторонней рассылки. После тщательного анализа были сделаны несколько важных выводов:

Бинарное дерево: Оно выделяется как наиболее эффективный алгоритм, демонстрируя логарифмическое увеличение времени рассылки при увеличении числа процессов, что указывает на масштабируемость и эффективность. Кроме того, биномиальное дерево: Расположенное между бинарным деревом и линейными алгоритмами, оно предлагает сбалансированный

профиль производительности, обеспечивая компромисс между эффективностью и сложностью. Линейный: Несмотря на свою простоту, он демонстрирует наименее эффективное время рассылки среди тройки, показывая линейное увеличение времени рассылки с увеличением числа процессов, что подчеркивает ограничения масштабируемости и эффективности.

Более того, необходимо признать потенциальные вызовы и ограничения, связанные с предложенным подходом:

1. Деградация производительности: Эффективность алгоритма может уменьшиться с увеличением количества процессов из-за роста накладных расходов на коммуникацию и синхронизацию. По мере масштабирования системы ее врожденные ограничения могут препятствовать эффективности.

2. Зависимость от количества процессов: Хотя алгоритм эффективен для небольшого и среднего числа процессов, увеличение числа процессов может привести к увеличению накладных расходов и задержек. Это подчеркивает необходимость тщательного рассмотрения масштабируемости в реальных распределенных системах.

3. Зависимость от ресурсов: Эффективность алгоритма зависит от доступной памяти и пропускной способности сети для безпроблемной коммуникации. Недостаточные ресурсы могут затруднить оптимальную производительность, что требует стратегий выделения и оптимизации ресурсов.

В заключение, хотя предложенный алгоритм проявляет обнадеживающие черты производительности, его эффективность зависит от того, насколько тщательно учитываются масштабируемость, доступность ресурсов и динамика системы. Будет крайне важно учитывать эти факторы, чтобы полностью использовать метод и применять его к распределенным системам памяти.

Список литературы

[1] Chunduri, S., Parker, S., Balaji, P., Harms, K., & Kumaran, K. (2018). Characterization of MPI usage on a production supercomputer. In Proceedings - International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018. doi: 10.1109/SC.2018.00033.

[2] Hoefler, T., Siebert, C., & Rehm, W. (2007). A practically constant-time MPI broadcast algorithm for large-scale InfiniBand clusters with multicast. In Proceedings - 21st International Parallel and Distributed Processing Symposium, IPDPS 2007; Abstracts and CD-ROM. doi: 10.1109/IPDPS.2007.370475.

[3] Hoefler, T., et al. (2015). Remote memory access programming in MPI-3. ACM Transactions on Parallel Computing, 2(2). doi: 10.1145/2780584.

[4] Abuelsoud, M. M., Kogutenko, A. A., & Naveen. (2024). Enhancing MPI remote memory access model for distributed-memory systems through one-sided broadcast implementation. J Phys Conf Ser, 2697(1), 012035. doi: 10.1088/1742-6596/2697/1/012035.

[5] Tipparaju, V., Nieplocha, J., & Panda, D. (2003). Fast collective operations using shared and remote memory access protocols on clusters. Proceedings - International Parallel and Distributed Processing Symposium, IPDPS 2003. doi: 10.1109/IPDPS.2003.1213188.

[6] Petrović, D., Shahmirzadi, O., Ropars, T., & Schiper, A. (2012). High-performance RMA-based broadcast on the Intel SCC. In Annual ACM Symposium on Parallelism in Algorithms and Architectures. doi: 10.1145/2312005.2312029.

[7] Tipparaju, V., Krishnan, M., Nieplocha, J., Santhanaraman, G., & Panda, D. (2003). Exploiting non-blocking remote memory access communication in scientific benchmarks. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2913. doi: 10.1007/978-3-540-24596-4_27.

[8] Sur, S., Bondhugula, U. K. R., Mamidala, A., Jin, H. W., & Panda, D. K. (2005). High performance RDMA based all-to-all broadcast for InfiniBand clusters. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). doi: 10.1007/11602569_19.

[9] Lusk, E., Huss, S., Saphir, B., & Snir, M. (2009). MPI: A message-passing interface standard Version 3.0. International Journal of Supercomputer Applications, 8(3/4).

[10] Gropp, W., & Lusk, E. (1999). Reproducible measurements of MPI performance characteristics. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). doi: 10.1007/3-540-48158-3_2.

[11] Hjelm, N., Dosanjh, M. G. F., Grant, R. E., Groves, T., Bridges, P., & Arnold, D. (2018). Improving MPI multi-threaded RMA communication performance. In ACM International Conference Proceeding Series. doi: 10.1145/3225058.3225114.

[12] Gropp, W., Lusk, E., & Skjellum, A. (2019). Using MPI - 2nd Edition Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation).

[13] Nuriyev, E., Rico-Gallego, J. A., & Lastovetsky, A. (2022). Model-based selection of optimal MPI broadcast algorithms for multi-core clusters. J Parallel Distrib Comput, 165. doi: 10.1016/j.jpdc.2022.03.012.

[14] Wadsworth, D. M., & Chen, Z. (2008). Performance of MPI broadcast algorithms. In IPDPS Miami 2008 - Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, Program and CD-ROM. doi: 10.1109/IPDPS.2008.4536478.

[15] Tu, B., Fan, J., Zhan, J., & Zhao, X. (2012). Performance analysis and optimization of MPI collective operations on multi-core clusters. Journal of Supercomputing, 60(1). doi: 10.1007/s11227-009-0296-3.

[16] Bruck, J., Ho, C. T., Kipnis, S., Upfal, E., & Weathersby, D. (1997). Efficient algorithms for all-to-all communications in multiport message-passing systems. IEEE Transactions on Parallel and Distributed Systems, 8(11). doi: 10.1109/71.642949.

[17] Hoefler T., & Lumsdaine A. (2008). Optimizing non-blocking collective operations for InfiniBand. In IPDPS Miami 2008 - Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, Program and CD-ROM. doi: 10.1109/IPDPS.2008.4536138.