DOI: 10.55648/1998-6920-2023-17-3-57-69 УДК 004.021

Алгоритмы редукции и широковещательной передачи для вычислительных кластеров с многоуровневыми сетями *

М. Г. Курносов

Сибирский гос. унив. телекоммуникаций и информатики (СибГУТИ)

Аннотация: Разработаны алгоритмы коллективных операций Allreduce, Bcast, Reduce стандарта MPI, основанные на динамическом формировании групп процессов на всех уровнях иерархии памяти в пределах вычислительного узла и уровнях коммуникационной сети, заданных описанием во внешнем конфигурационном файле. Алгоритмы реализованы на базе библиотеки Open MPI. Эксперименты на вычислительном кластере с двухуровневой сетью стандарта InfiniBand EDR показали эффективность текущей версии для сообщений до 16 КБ.

Ключевые слова: коллективные операции, редукция, allreduce, broadcast, MPI.

Для цитирования: Курносов М. Г. Алгоритмы редукции и широковещательной передачи для вычислительных кластеров с многоуровневыми сетями // Вестник СибГУТИ. 2023. Т. 17, № 3. С. 57–69. https://doi.org/10.55648/1998-6920-2023-17-3-57-69.

Статья поступила в редакцию 03.03.2023;
принята к публикации 18.03.2023.

1. Введение

Подавляющее большинство высокопроизводительных вычислительных систем строятся на базе серийно выпускаемых многопроцессорных серверов и коммуникационных сетей на базе стандартизированных технологий (InfiniBand, 10X Gigabit Ethernet, HPC Ethernet). В структурной организации таких систем явно выделяются иерархические уровни, в пределах которых группируются процессорные ядра. В табл. 1 приведен пример для высокопроизводительного кластера, узлы которого объединены коммуникационной сетью на базе двух уровней коммутаторов (топология spine-leaf, fat tree). Как правило, процессорные ядра одного уровня имеют техническую возможность более быстрого взаимодействия, по сравнению с ядрами на вышележащем уровне.

В гибридных системах отдельные уровни может образовывать память графических процессоров и каналы связи между ними (например, NVIDIA NVLink, AMD XGMI).

Для обеспечения равномерной загрузки процессорных ядер и коммуникационных каналов высокопроизводительные системы имеют *симметричную* (однородную) конфигурацию — одинаковое количество процессорных ядер на одном NUMA-узле; одинаковое количество процессоров на всех вычислительных узлах; равное число узлов, подключенных к разным сетевым коммутаторам.

* Работа выполнена в рамках государственного задания № 071-03-2023-001 от 19.01.2023.

Таблица 1. Иерархические уровни высокопроизводительного кластера

Уровень	Функциональный модуль	Среда взаимодействия
1	NUMA-узел	Кеш-память верхнего уровня (last level cache — LLC) или контроллер памяти (системная память)
2	Многоядерный процессор	Кеш-память верхнего уровня или системная память
3	Вычислительный узел	Системная память
4	Сетевой коммутатор L1	Сетевой коммутатор L1 (leaf), к которому подключены соответствующие вычислительные узлы
5	Сетевой коммутатор L2	Группа сетевых коммутаторов L2 (spine), к которым подключены коммутаторы L1

При разработке коммуникационных библиотек для высокопроизводительных вычислений (Open MPI, MPICH, Open UCX, libfabric) и распределенного машинного обучения (NVIDIA NCCL, Horovod, Facebook Gloo) встает задача создания алгоритмов и коммуникационных протоколов, минимизирующих время выполнения основных схем информационных обменов в вычислительных системах. Значительные усилия сосредоточены на разработке алгоритмов реализации коллективных коммуникационных операций между процессами параллельных программ: one-to-all, all-to-one и all-to-all, которые требуют участия всех или значительной части процессов программы. Известные алгоритмы коллективных операций можно разделить на три группы. К первой относятся алгоритмы, которые не учитывают *топологию* системы — иерархию памяти в пределах узла и структуру сетевых соединений вычислительных узлов. Они логически выстраивают процессы в деревья (графы) различных видов, опираясь лишь на информацию о номерах процессов и их количестве [1, 2]. Обмены между процессами реализуются базовыми двусторонними операциями send/recv или односторонними примитивами прямого удаленного доступа к памяти put/get, которые предоставляет нижний уровень коммуникационной библиотеки или драйвер сетевого интерфейса. Такие алгоритмы легко переносятся между системами разных архитектур и конфигураций, но не обеспечивают предельной производительности.

Во второй группе находятся алгоритмы, которые используют операции send/recv, но учитывают часть свойств топологии системы. В работах [3, 4] алгоритмы учитывают два иерархических уровня — используют общую память ядер для обменов в пределах вычислительного узла и коммуникационную сеть для взаимодействий между процессами разных узлов. В [5, 6] учитывается внутриузловая группировка процессов по NUMA-узлам и процессорам.

К третьей группе относятся алгоритмы, разработанные для систем с фиксированной топологией: многомерного тора [7, 8], гиперкуба или «толстого дерева» (fat tree) [9]. Часть этапов выполнения коллективных операций в таких системах, как правило, имеют аппаратную реализацию.

В данной работе предложен подход к разработке алгоритмов коллективных операций стандарта MPI, основанный на динамическом формировании групп процессов на всех уровнях иерархии памяти в пределах вычислительного узла и уровнях коммуникационной сети, заданных описанием во внешнем конфигурационном файле. Разработаны алгоритмы операций Allreduce, Bcast и Reduce, которые не используют механизм MPI-коммуникаторов для группировки процессов, что сокращает время выполнннения коммуникационных операций. Реализация выполнена на базе библиотеки Open MPI. Внутриузловая топология определяется автоматически средствами библиотек Open MPIx и HWLOC. Описание иерархии коммутаторов сети межмашинных связей загружается из файла описания топологии сети. Данная работа развивает результаты [10].

2. Построение иерархии групп процессов

В стандарте MPI процессы логически группируются в коммуникаторы, которые задают областью выполнения коллективных операций. При запуске MPI-программы процессы получают уникальные номера {0, 1, ..., p - 1}, где p — число процессов в глобальном коммуникаторе. В ходе выполнения могут создаваться новые коммуникаторы, включающие все или часть процессов. При этом схема распределения номеров процессов (рангов) по процессорным ядрам заранее неизвестна и задается пользователем при запуске приложения. Например, при использовании политики распределения процессов `map-by core` процессы последовательно нумеруются от ядра к ядру; `map-by numa` — процессы нумеруются с шагом через NUMA-узел. В общем случае это приводит к тому, что два процесса со смежными номерами могут находиться на любом расстоянии в пределах вычислительной системы — на одном NUMA-узле или на разных узлах, подключенных к разным сетевым коммутаторам. Поэтому при создании нового коммуникатора выполняется анализ топологии системы и распределение процессов по ядрам. Для каждого процесса строятся группы, в которые они входят на каждом уровне топологии систем. Эта информация используется при вызове коллективной операции и выполнения обменов с учетом топологии.

2.1. Анализ топологии системы и распределения процессов по ядрам

Каждый MPI-процесс определяет число ядер, разделяющих кеш-память уровней L2, L3, число NUMA-узлов и процессоров на вычислительном узле. Вычисляется количество *L активных уровней* иерархии памяти – уровней, которые совместно используются двумя и более физическими ядрами: кеш-память L2, L3, NUMA-узел, процессор (socket, package). Для анализа иерархии памяти в пределах узла используется библиотека HWLOC, которая доступна на широком спектре архитектур.

Далее выполняется построение *таблицы размещения процессов* в пределах узла. Процесс определяет множество ядер, к которым он привязан (на каких ядрах планировщик операционной системы GNU/Linux может его выполнять). Через обращение к интерфейсу библиотеки Open PMIx определяется размещение всех P процессов коммуникатора в пределах узла (`PMIX_LOCALITY_STRING`) — для каждого процесса формируется вектор, содержащий: номер процессора, номер кеш-памяти L3, кеш-памяти L2, кеш-памяти L1, номер ядра, номер NUMA-узла. Для каждого активного уровня l иерархии памяти определяется номер функционального модуля $loc[l][i]$, который использует процессом i на уровне l. Вычислительная сложность шага – $O(PL)$. Для примера, на рис. 1 PMIx-строка с размещением процесса 18 имеет вид `(SK1:L31:L218:L118:CR18:NM1)`.

По умолчанию библиотека Open MPI привязывает процесс к процессорному ядру (`--bind-to core`), однако при запуске программы пользователь может изменить способ привязки – разрешить операционной системе динамически переносить процесс в пределах одного процессора или NUMA-узла (`--bind-to package | numa`) или отключить привязку. При отключенной привязке процессов к процессорным ядрам, алгоритм завершает свою работу, так как невозможно определить, какие уровни иерархии памяти процессы используют совместно.

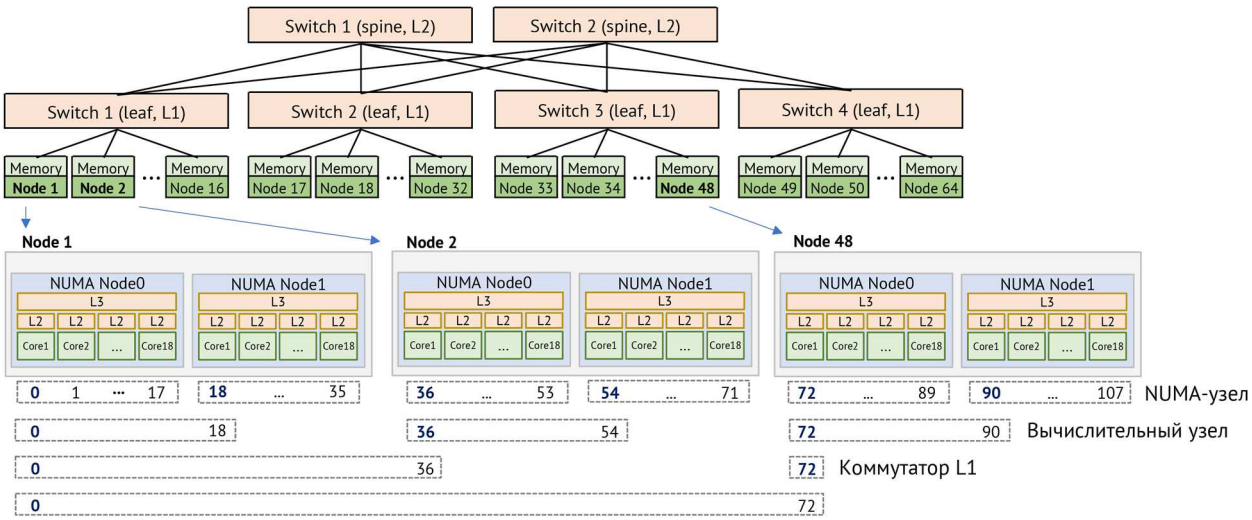

Рис. 1. Вычислительный кластер: 64 узла, 2 процессора на узле (NUMA-узлы), два уровня сетевых коммутаторов (16 узлов на коммутатор уровня 1), четыре уровня групп для 108 процессов на узлах 1, 2 и 48

Информация о сетевой топологии загружается из внешнего файла. Реализован минимально необходимый формат описания для систем на базе сетей с иерархией коммутаторов (топологии fat tree, spine-leaf). В файле для каждого вычислительного узла перечисляются идентификаторы коммутаторов, к которым они подключены. Этого достаточно для формирования групп узлов. На рис. 2 приведен пример для системы на рис. 1.

```
# <node-name> <switch-name>
node01 sw11
...
node16 sw11
node17 sw12
...
node32 sw12
...
node49 sw14
...
node64 sw14
```

Рис. 2. Пример описания сетевой топологии: распределение вычислительных узлов по сетевым коммутаторам уровня 1

2.2. Построение групп процессов

Каждый процесс для всех активных уровней иерархии памяти и сетевой топологии строит списки процессов, с которыми он разделяет один функциональный модуль (кеш-память L2, NUMA-узел, процессор, вычислительный узел, сетевой коммутатор). Построение начинается с низшего активного уровня. В каждой группе выбирается лидер – процесс с наименьшим номером. Лидеры групп становятся членами групп следующего активного уровня. Цикл повторяется для всех активных уровней. Вычислительная сложность построения групп – $O(L(C + P))$, где C – число ядер на вычислительном узле. На рис. 1 показана иерархия из четырех уровней групп процессов. Группа $G1$ — процессы одного NUMA-узла, группа $G4$ включает процессы на узлах, подключенных к разным коммутаторам уровня 2. Ниже приведены примеры групп:

- процесс 0: $G1(0, 1, \ldots, 17)$, $G2(0, 18)$, $G3(0, 36)$, $G4(0, 72)$;
- процесс 1: $G1(0, 1, \ldots, 17)$;
- процесс 36: $G1$ (36, 37, …, 53), $G2(36, 54)$, $G3(0, 36)$.

Реализована возможность динамического отключения группировки процессов на отдельных уровнях топологии. Это позволяет на целевой системе подобрать эффективное сочетание числа иерархических уровней для выполнения алгоритмов коллективных операций.

3. Алгоритм операции Reduce

В момент вызова операции `MPI_Reduce` каждому процессу известен свой исходный номер *rank* в коммуникаторе, адрес буфера отправки *sbuf*, содержащий *count* элементов типа *type*. Буфер приема *rbuf* присутствует только в адресном пространстве корневого процесса *root*. В буфере *rbuf* корневого процесса должен быть сформирован результат поэлементного применения ассоциативной бинарной операции *op* ко всем буферам отправки. На рис. 3 приведен псевдокод алгоритма. Для экономии места опущена логика обработки кодов возврата из функций и освобождения памяти, нумерация уровней топологии начинается с 0 (низший уровень). На каждом уровне иерархии топологии системы выполняется редукция между членами группы. Результат формируется в буфере лидера. Поэтому если процесс является лидером хотя бы одной группы, то он выделяет два временных буфера для приема промежуточных результатов. После редукции на уровне *level* выполняется обмен указателей *sptr* и *rptr*, чтобы буфер приема предыдущего уровня стал буфером отправки на следующем. В конце работы алгоритма глобальный лидер передает результат корневому процессу.

Функция *ReduceGroup* выполняет заданный пользователем «плоский», не учитывающий топологию системы, алгоритм редукции между членами группы. Результат формируется в буфере лидера группы. Реализована возможность выбора для каждого иерархического уровня своего алгоритма редукции на уровне групп: алгоритм плоского дерева (flat tree), алгоритм биномиального дерева (binomial tree) и комбинированный алгоритм ReduceScatter с Gather [1]. Алгоритм поддерживает только коммутативные операции *op*, так как результат промежуточных редукций зависит от распределения процессов по процессорным ядрам.

```
algorithm MPI_Reduce(sbuf, rbuf, count, type, op, root)
  if rank = groups[0].leader then
    tmpbuf[0] = AllocMem(count * sizeof(type))      // Выделение памяти
    tmpbuf[1] = AllocMem(count * sizeof(type))
  end if

  sptr = sbuf
  if rank = root and sbuf = MPI_IN_PLACE then
    sptr = rbuf                          // Данные для отправки уже в rbuf
  end if
  rptr = tmpbuf[1]

  for level = 0 to ngroups - 1 do
    // Редукция на уровне level между процессами группы
    ReduceGroup(sptr, rptr, count, type, op, 0, groups[level])

    if level < ngroups - 1 then
      p = sptr
      if level != 0 then p = tmpbuf[0]
      sptr = rptr
      rptr = p
      if level = ngroups - 2 and rank = root and root = 0 then
        // Лидер формирует конечный результат в rbuf
        rptr = rbuf
      end if
    end if
  end for
```

```
  if root != 0 then
    if rank = 0 then       // Лидер передает сообщение корневому процессу
      Send(rptr, count, type, root)
    else if rank = root then
      Recv(rbuf, count, type, 0)
    end if
  end if
end algorithm
```

Рис. 3. Псевдокод алгоритма операции Reduce

Время выполнения алгоритма линейно зависит от количества L активных уровней иерархии и от времени выполнения плоского алгоритма *ReduceGroup*.

4. Алгоритм операции Bcast

В операции `MPI_Bcast` каждому процессу известен адрес буфера *buf* для *count* элементов типа *type*. На стороне корневого процесса он является буфером отправки, а некорневые процессы выполняют прием данных в него. На рис. 4 показан псевдокод алгоритма. На каждом уровне иерархии топологии системы выполняется широковещательная рассылка сообщения из буфера *buf* лидера группы всем её членам. Функция *BcastGroup* выполняет заданный пользователем алгоритм широковещательной передачи в пределах группы. Для каждого иерархического уровня возможно задание конкретного алгоритма операции Bcast: алгоритм плоского дерева (flat tree, linear), алгоритм k-номиального дерева (k-nomial tree), комбинированный алгоритм Scatter и Allgather [1].

```
algorithm MPI_Bcast(buf, count, type, root)
  if root != 0 then
    if rank = root then       // Корень передает сообщение лидеру
      Send(buf, count, type, 0)
    else if rank = 0 then
      Recv(buf, count, type, root)
    end if
  end if

  for level = ngroups - 1 to 0 do
    // Bcast на уровне level между процессами группы
    BcastGroup(buf, count, type, 0, groups[level])
  end for
end algorithm
```

Рис. 4. Псевдокод алгоритма операции Bcast

5. Алгоритм операции Allreduce

В результате выполнения операции `MPI_Allreduce` в буфере *rbuf* каждого процесса должен быть сформирован результат поэлементного применения ассоциативной и коммутативной бинарной операции *op* ко всем буферам отправки *sbuf*. Разработано две версии алгоритма. Первая использует на отдельных этапах операции Reduce, Allreduce и Bcast. Во второй версии используются Reduce и Bcast.

5.1. Алгоритм ReduceAllreduceBcast

На первом шаге осуществляется выполнение редукции *ReduceGroup* на каждом уровне иерархии – от нулевого до предпоследнего. На втором шаге выполняется алгоритм *AllreduceGroup* (алгоритм рекурсивного удваивания) для процессов групп на последнем уровне иерархии топологии систем. Третий шаг — широковещательная рассылка *BcastGroup* результатов лидерами группы, начиная с предпоследнего уровня до нулевого. Если процесс не входит в группу на верхнем уровне иерархии, то он не участвует в операции *AllreduceGroup*.

```
algorithm MPI_Allreduce(sbuf, rbuf, count, type, op)
  if nlevels = ngroups then
    // Процесс входит в группу на самом верхнем уровне иерархии
    for level = 0 to ngroups – 2 do
      sptr = sbuf
      if level = 0 and sbuf = MPI_IN_PLACE then sptr = MPI_IN_PLACE
      ReduceGroup(sptr, rbuf, count, type, op, 0, groups[level])
    end for

    AllreduceGroup(MPI_IN_PLACE, rbuf, count, type, op,
                   groups[nlevels – 1])
    for level = ngroups – 2 to 0 do
      // Bcast на уровне level между процессами группы
      BcastGroup(rbuf, count, type, 0, groups[level])
    end for
  else
    // Процесс не входит в группу на самом верхнем уровне иерархии
    for level = 0 to ngroups – 1 do
      sptr = sbuf
      if level = 0 and sbuf = MPI_IN_PLACE then sptr = MPI_IN_PLACE
      ReduceGroup(sptr, rbuf, count, type, op, 0, groups[level])
    end for

    for level = ngroups – 1 to 0 do
      // Bcast на уровне level между процессами группы
      BcastGroup(rbuf, count, type, 0, groups[level])
    end for
  end if
end algorithm
```

Рис. 5. Псевдокод алгоритма ReduceAllreduceBcast операции Allreduce

5.2. Алгоритм ReduceBcast

На первом этапе выполняется вызов операции Reduce с иерархическим алгоритмом, на втором — результаты редукции из буфера глобального лидера рассылаются всем процессам иерархическим алгоритмом операции Bcast.

```
algorithm MPI_Allreduce(sbuf, rbuf, count, type, op)
  if sbuf = MPI_IN_PLACE then
    if rank = 0 then
      Reduce(MPI_IN_PLACE, rbuf, count, type, op, 0)
    else
      Reduce(rbuf, NULL, count, type, op, 0)
    end if
  else
    Reduce(sbuf, rbuf, count, type, op, 0)
```

```
  end if
  Bcast(rbuf, count, type, 0)
end algorithm
```

Рис. 6. Псевдокод алгоритма ReduceBcast операции Allreduce

6. Результаты экспериментов

Алгоритмы реализованы на базе библиотеки Open MPI 5.0.0rc9 в отдельном компоненте coll/tcl, что позволяет распространять их как динамическую библиотеку. Эксперименты проведены на вычислительном кластере из 10 узлов на базе двух процессоров Huawei Kunpeng 920 (64 ядра ARMv8, 2 NUMA-узла на процессор), 255 Гб оперативной памяти. Коммуникационная сеть InfiniBand EDR. Операционная система CentOS 7.6 (linux 4.14.0-115, aarch64), компилятор gcc 4.8.5.

В качестве теста эффективности реализаций алгоритмов MPI использован пакет OSU Micro-Benchmarks 5.8. Для каждого размера сообщения тест запускался 1000 раз. Отбрасывались наименьшее и наибольшее значения максимальной латентности, среди оставшихся значений отыскивалось среднее время работы операции [4]. Каждый процесс привязывался к процессорному ядру (`mpiexec --bind-to core`).

Выполнялось сравнение разработанных алгоритмов в компоненте coll/tcl с алгоритмами на базе операций send/recv библиотеки Open MPI, компонент coll/tuned. Для каждой операции разработанные алгоритмы запускались с различным количеством учитываемых топологических уровней вычислительного кластера:

- `NM-SK-ND-SW`: в группы объединялись процессы одного NUMA-узла (NM), одного процессора (SK, SOCKET), на одном вычислительном узле (ND) и подключенные к одному InfiniBand-коммутатору первого уровня;
- `NM-ND-SW`: в группы объединялись процессы общего NUMA-узла, на одном вычислительном узле и подключенные к одному InfiniBand-коммутатору первого уровня (отключалась группировка по процессору).

В табл. 2 приведены результаты выполнения операции Reduce. Разработанные алгоритмы coll/tcl на базе биномиального дерева и комбинации ReduceScatter с Allgather обеспечили в среднем на 60–80 % меньшее время выполнения для сообщений от 4 байт до 4 КБ, по сравнению с алгоритмами coll/tuned. Предложенные алгоритмы уступают coll/tuned на сообщениях больше 16 КБ, одна из причин — текущая версия coll/tcl не поддерживает алгоритмическую конвейеризацию передачи сообщений (pipelining, message segmentation).

Таблица 2. Нормализованное время теста OSU Reduce
(время нормализовано относительно алгоритма бинарного дерева)

Сообщение (байт)	Алгоритмы Open MPI coll/tuned						Алгоритмы coll/tcl					
							Уровни: NM-SK-ND-SW			Уровни: NM-ND-SW		
	Binary tree	Binomial tree	Chain	Linear	Pipeline	RedScat Allgather	Linear	Binomial	RedScat Allgather	Linear	Binomial	RedScat Allgather
4	1	0.627	1.113	1.141	1.897	1.08	0.743	1.072	0.213	0.824	0.465	0.281
8	1	0.762	0.452	0.53	1.412	0.618	0.119	0.204	0.05	0.087	0.041	0.046
16	1	1.046	1.782	2.634	6.77	2.333	1.387	0.445	3.334	2.1	0.217	0.506
32	1	0.124	0.497	0.621	1.63	0.618	0.424	0.047	0.109	0.466	0.043	0.705
64	1	0.942	1.825	2.351	6.477	2.311	0.897	0.193	0.462	1.529	2.641	1.769

128	1	0.363	1.631	2.065	5.079	2.064	0.646	0.131	0.621	0.709	0.649	0.667
256	1	3.605	2.063	3.94	7.638	3.561	0.739	0.212	0.515	0.48	0.259	0.474
512	1	0.293	0.91	26.827	6.341	6.475	0.267	0.091	0.204	0.153	0.101	0.206
1024	1	0.678	2.504	951.87	9.489	965.37	0.42	0.206	0.471	0.414	0.261	0.456
2048	1	0.871	0.832	278.594	3.2	1.08	0.12	0.069	0.145	0.123	0.086	0.141
4096	1	0.001	0.005	4.584	0.033	0.004	0.001	0.002	0.001	0.001	0.000	0.001
8192	1	0.002	0.006	7.406	0.03	0.002	0.001	0.000	0.001	0.001	0.000	0.001
16384	1	0.001	0.008	12.333	0.031	0.001	0.002	0.001	0.001	0.002	0.001	0.001
32768	1	0.001	0.012	14.913	0.055	0.001	0.002	0.003	0.001	0.003	0.001	0.001
65536	1	0.001	0.011	6.751	0.036	0.001	0.002	0.001	0.001	0.002	0.001	0.001
131072	1	0.001	0.016	6.223	0.067	0.001	0.057	0.002	0.002	0.025	0.002	0.001
262144	1	0.001	0.014	0.075	0.058	0.001	0.003	0.002	0.002	0.003	0.001	0.002
524288	1	0.002	0.021	0.095	0.077	0.002	0.005	0.003	0.003	0.006	0.003	0.003
1048576	1	0.019	0.163	0.729	0.634	0.021	0.044	0.027	0.028	0.043	0.028	0.024

Результаты для операции Bcast представлены в табл. 3. Алгоритм *k*-номиального дерева ($k = 2$) опережает по времени выполнения все алгоритмы coll/tuned в 2–10 раз на сообщениях от 4 байт до 512 КБ. При отключении группировки процессов по процессору (`NM-ND-SW`) повышается эффективность линейного алгоритма в связи с сокращением накладных расходов на прохождение иерархии групп процессов. На сообщениях больше 512 KB разработанный компонент демонстрирует сопоставимые с алгоритмом ScatterAllgatherRing из coll/tuned результаты.

Таблица 3. Результаты для алгоритмов операции Bcast
(время нормализовано относительно алгоритма плоского дерева)

Сообщение (байт)	Алгоритмы Open MPI/tuned							Алгоритмы coll/tcl					
								Уровни: NM-SK-ND-SW			Уровни: NM-ND-SW		
	Linear tree	Binary tree	Binomial tree	Chain	Pipeline	Scatter Allgather	Scatter Allgather Ring	Knomial	Linear	Scatter Allgather Ring	Knomial	Linear	Scatter Allgather Ring
4	1	2.06	1.85	3.05	5.10	0.97	1.41	0.01	0.02	0.02	0.01	0.03	0.03
8	1	0.27	0.19	0.71	1.85	0.98	0.93	0.01	0.02	0.04	0.01	0.14	0.03
16	1	0.22	0.19	1.59	1.66	1.07	0.95	0.01	0.04	0.03	0.01	0.03	0.05
32	1	0.30	0.19	1.45	1.68	0.99	1.02	0.01	0.02	0.09	0.01	0.03	0.11
64	1	0.32	0.19	0.47	1.69	1.00	0.99	0.01	0.05	0.13	0.01	0.03	0.1
128	1	0.23	0.19	0.73	1.69	1.01	1.01	0.01	0.03	0.18	0.01	0.03	0.17
256	1	0.20	0.17	0.63	1.62	1.00	0.99	0.01	0.03	0.23	0.01	0.03	0.15
512	1	0.17	0.15	0.43	1.50	1.00	1.08	0.01	0.05	0.13	0.03	0.05	0.12
1024	1	0.20	0.15	0.39	1.36	1.01	1.02	0.02	0.04	0.11	0.03	0.05	0.12
2048	1	0.16	0.15	0.34	1.26	1.01	0.90	0.02	0.05	0.09	0.03	0.05	0.1
4096	1	0.13	0.12	0.29	1.08	0.16	0.52	0.02	0.05	0.07	0.03	0.04	0.07
8192	1	0.15	0.16	0.32	1.19	0.17	0.44	0.02	0.08	0.07	0.02	0.04	0.07
16384	1	0.18	0.20	0.28	1.09	1.08	0.32	0.02	0.04	0.06	0.02	0.04	0.06
32768	1	0.19	0.23	0.25	0.93	13.55	0.21	0.02	0.03	0.04	0.02	0.03	0.04
65536	1	0.25	0.28	0.37	1.51	20.63	0.14	0.04	0.03	0.03	0.04	0.03	0.03
131072	1	0.41	0.38	0.26	1.06	17.94	0.08	0.03	0.04	0.03	0.03	0.02	0.03

262144	1	0.49	0.29	0.25	0.90	26.13	0.05	0.02	0.04	0.02	0.02	0.02	0.02
524288	1	0.38	0.32	0.17	0.75	30.58	0.03	0.02	0.03	0.02	0.02	0.01	0.02
1048576	1	0.22	0.27	0.16	0.70	12.79	0.02	0.02	0.02	0.02	0.02	0.01	0.02
2097152	1	0.53	0.26	0.18	0.77	6.81	0.01	0.02	0.02	0.02	0.02	0.01	0.03
4194304	1	0.42	0.25	0.22	0.83	3.28	0.02	0.02	0.02	0.02	0.02	0.01	0.03

В табл. 4 представлены результаты для операции Allreduce. Сравнение выполнено с алгоритмами компонента coll/tuned: *linear* (плоское дерево, его время бралось для нормализации всех значений), *ReduceBcast* (комбинация Reduce + Bcast), *RedScatAllgather* (алгоритм Р. Рабенсейфнера [1]), алгоритм рекурсивного удваивания, *ring* (кольцевой алгоритм), *segmented ring* (кольцевой алгоритм с сегментацией сообщений). При группировке процессов по всем уровням топологии (`NM:SK:ND:SW`) наилучшие результаты для сообщений от 4 байт до 2КБ демонстрирует алгоритм ReduceBcast. При отключении группировки процессов по процессорам алгоритмы coll/tcl уступают по производительности алгоритмам coll/tuned.

Таблица 4. Результаты для алгоритмов операции Allreduce
(время нормализовано относительно алгоритма плоского дерева)

Сообщение (байт)	Алгоритмы Open MPI coll/tuned						Алгоритмы coll/tcl			
							Уровни: NM:ND:SW		Уровни: NM:SK:ND:SW	
	Linear	ReduceBcast	RedScat Allgather	Recursive doubling	Ring	Segmented ring	ReduceBcast	ReduceAllreduce Bcast	ReduceBcast	ReduceAllreduce Bcast
4	1	0.4924	0.9389	0.7283	0.6485	0.6169	0.0465	0.1503	0.2084	0.0369
8	1	0.2168	1.0238	0.1828	0.2276	0.2309	0.0475	0.4385	0.1803	0.1904
16	1	0.1383	1.0771	0.1506	0.153	0.1493	0.3401	0.4471	0.1096	0.2385
32	1	0.1087	1.0355	0.1481	0.1466	0.1418	0.1904	0.4401	0.0904	0.2348
64	1	0.0354	0.6934	0.0478	0.0744	0.0726	0.0282	0.1379	0.0234	0.0736
128	1	0.0863	1.5587	0.1205	0.1065	0.0974	0.0764	0.3022	0.07	0.1612
256	1	0.0871	0.9809	0.1152	0.1194	0.1141	0.0779	0.2974	0.0751	0.1587
512	1	0.081	1.0236	0.1063	0.1164	0.1083	0.0802	0.2552	0.0744	0.1362
1024	1	0.0007	0.9807	0.001	0.001	0.001	0.0008	0.0021	0.0007	0.0011
2048	1	0.0008	0.0042	0.0741	0.0787	0.0933	0.0008	0.002	0.0008	0.0011
4096	1	0.0009	0.0013	0.2715	0.0075	0.0073	0.001	0.0019	0.001	0.001
8192	1	0.0004	0.001	0.1872	0.0025	0.0023	0.0005	0.0006	0.0004	0.0004
16384	1	0.002	0.0002	0.1142	0.001	0.001	0.0003	0.0003	0.0003	0.0002
32768	1	0.0003	0.0002	0.1395	0.0008	0.0009	0.0003	0.0003	0.0003	0.0003
65536	1	0.0004	0.0003	0.1468	0.0009	0.0009	0.0004	0.0004	0.0005	0.0004
131072	1	0.001	0.0005	0.2633	0.0012	0.0018	0.001	0.0008	0.001	0.0008
262144	1	0.0211	0.0106	2.1571	0.0108	0.0108	0.027	0.0203	0.024	0.0223
524288	1	0.0212	0.0149	1.9726	0.0108	0.0107	0.0276	0.0277	0.0294	0.0267
1048576	1	0.0259	0.0208	0.8288	0.0185	0.0183	0.0288	0.0255	0.0296	0.0282

7. Заключение

Разработанные алгоритмы для операций Bcast, Reduce и Allreduce учитывают многоуровневую топологию современных высокопроизводительных кластеров. Алгоритмы реализованы на базе библиотеки Open MPI в изолированном компоненте, который поддерживает описание сетевой топологии на базе составных коммутаторов. Внутриузловая топология определяется динамически средствами библиотек Open PMIx и HWLOC. Эксперименты показали эффективность текущей версии для сообщений до 16 КБ. Эффективная поддержка сообщений больших размеров требует создания версий алгоритмов с поддержкой сегментации сообщений. Следует отметить ограниченную поддержку в текущей версии алгоритмов асимметричного подключения сетевых адаптеров к процессорам, а также присутствия нескольких сетевых интерфейсов на одном узле.

Литература

1. *Thakur R., Rabenseifner R., Gropp W.* Optimization of collective communication operations in MPICH // Int. Journal of High Performance Computing Applications. 2005. V. 19 (1). P. 49-66.
2. *Balaji P., Buntinas D., Goodell D., Gropp W., Hoefler T., Kumar S., Lusk E., Thakur R., Traff J.* MPI on Millions of Cores // Parallel Processing Letters. 2011. V. 21, Is. 1. P. 45–60.
3. *Graham R.* Cheetah: A Framework for Scalable Hierarchical Collective Operations // Proc. IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2011. P. 73–83.
4. *Jain S., Kaleem R., Balmana M., Langer A., Durnov D., Sannikov A. and Garzaran M.* Framework for Scalable Intra-Node Collective Operations using Shared Memory // Proc. International Conference for High Performance Computing, Networking, Storage, and Analysis (SC-2018), 2018. P. 374–385.
5. *Zhu H., Goodell D., Gropp W., Thakur R.* Hierarchical Collectives in MPICH2 // Proc. EuroPVM/MPI, 2009. P. 325–326.
6. *Luo X.* HAN: a Hierarchical AutotuNed Collective Communication Framework // Proc. IEEE International Conference on Cluster Computing (CLUSTER), 2020. P. 23–34.
7. *Kurokawa M.* The K computer: 10 Peta-FLOPS supercomputer // Proc. 10th International Conference on Optical Internet (COIN2012), 2012. P. 1.
8. *Kumar S., Mamidala A., Heidelberger P., Chen D., Faraj D.* Optimization of MPI Collective Operations on the IBM Blue Gene/Q Supercomputer // Journal of High Performance Computing Applications. 2014. № 28 (4). P. 450–464.
9. *Venkata M., Bloch G., Shainer G., Graham R.* Accelerating OpenSHMEM Collectives Using In-Network Computing Approach // Proc. International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2019. P. 212–219.
10. *Курносов М. Г.* Иерархический алгоритм барьерной синхронизации для многопроцессорных систем с общей памятью // Вестник СибГТУИ. 2022. Т. 16, № 2 (58). С. 3–11.

Курносов Михаил Георгиевич

д.т.н., профессор кафедры вычислительных систем, Сибирский государственный университет телекоммуникаций и информатики (СибГУТИ, 630102, Новосибирск, ул. Кирова, 86), тел. +7 383 2698 382, e-mail: mkurnosov@sibguti.ru;

старший научный сотрудник, Институт физики полупроводников им. А. В. Ржанова Сибирского отделения Российской академии наук (630090, Новосибирск, просп. Ак. Лаврентьева, 13), тел. +7 383 3305 626, e-mail: mkurnosov@isp.nsc.ru, ORCID ID: 0000-0002-7808-1635.

Автор прочитал и одобрил окончательный вариант рукописи.
Автор заявляет об отсутствии конфликта интересов.

MPI Reduction and Broadcast Algorithms for Computer Clusters with Multistage Interconnection Networks

Mikhail G. Kurnosov

Siberian State University of Telecommunications and Information Science (SibSUTIS)

Abstract: Algorithms for MPI Allreduce, Bcast and Reduce collective operations have been developed. They are based on the dynamic construction of process groups at all levels of the memory hierarchy within the computing node and levels of the multistage interconnection network. The description of the network topology is loaded from an external file. Algorithms are implemented on the basis of the Open MPI library in an isolated collective component. Experiments on a cluster with a two-tier InfiniBand EDR network have shown the effectiveness of the algorithms for messages up to 16KB.

Keywords: collective communication, allreduce, broadcast, MPI.

For citation: Kurnosov M. G. MPI reduction and broadcast algorithms for computer clusters with multistage interconnection networks. (in Russian). *Vestnik SibGUTI*, 2023, vol. 17, no. 3, pp. 57-69. https://doi.org/10.55648/1998-6920-2023-17-3-57-69.

The article was submitted: 03.03.2023;
accepted for publication 18.03.2023.

References

1. Thakur R., Rabenseifner R., Gropp W. Optimization of collective communication operations in MPICH. *Int. Journal of High Performance Computing Applications*, 2005, vol. 19 (1), pp. 49-66.
2. Balaji P., Buntinas D., Goodell D., Gropp W., Hoefler T., Kumar S., Lusk E., Thakur R., Traff J. MPI on Millions of Cores. *Parallel Processing Letters*, 2011, vol. 21, iss. 1, pp. 45-60.
3. Graham R. Cheetah: A Framework for Scalable Hierarchical Collective Operations. *Proc. of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2011, pp. 73-83.
4. Jain S., Kaleem R., Balmana M., Langer A., Durnov D., Sannikov A. and Garzaran M. Framework for Scalable Intra-Node Collective Operations using Shared Memory. *Proc. of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC-2018)*, 2018, pp. 374-385.
5. Zhu H., Goodell D., Gropp W., Thakur R. Hierarchical Collectives in MPICH2. *Proc. of EuroPVM/MPI*, 2009, pp. 325–326.

6. Luo X. HAN: a Hierarchical AutotuNed Collective Communication Framework. *Proc. of the IEEE International Conference on Cluster Computing (CLUSTER)*, 2020, pp. 23-34.
7. Kurokawa M. The K computer: 10 Peta-FLOPS supercomputer. *Proc. of the 10th International Conference on Optical Internet (COIN2012)*, 2012, pp. 1.
8. Kumar S., Mamidala A., Heidelberger P., Chen D., Faraj D. Optimization of MPI Collective Operations on the IBM Blue Gene/Q Supercomputer. *Journal of High Performance Computing Applications*, 2014, no. 28(4), pp. 450-464.
9. Venkata M., Bloch G., Shainer G., Graham R. Accelerating OpenSHMEM Collectives Using In-Network Computing Approach. *Proc. of the International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2019, pp. 212-219.
10. Kurnosov M. G. Ierarhicheskij algoritm bar'ernoj sinhronizacii dlya mnogoprocessornyh sistem s obshchej pamyat'yu [Barrier synchronization hierarchical algorithm for multicore shared-memory systems]. *Vestnik SibGUTI*, 2022, vol. 16, no. 2(58), pp. 3-11.

Mikhail G. Kurnosov

Dr. of Sci. (Engineering), Professor; Professor of the Department of Computer Systems, Siberian State University of Telecommunications and Information Science (SibSUTIS, Russia, 630102, Novosibirsk, Kirov St. 86), phone: +7 383 2698 272, e-mail: mkurnosov@sibguti.ru; Senior Research Scientist, Rzhanov Institute of Semiconductor Physics Siberian Branch of Russian Academy of Sciences, (ISP SB RAS, 630090, Novosibirsk, Lavrenteva ave. 13), phone: +7 383 3305 626, e-mail: mkurnosov@isp.nsc.ru, ORCID ID: 0000-0002-7808-1635.