УДК 004.272

АНАЛИЗ И ОПТИМИЗАЦИЯ АЛГОРИТМА ПАРАЛЛЕЛЬНЫХ ЦЕПОЧЕК ДЛЯ РЕАЛИЗАЦИИ КОРНЕВОЙ РЕДУКЦИИ НА РАСПРЕДЕЛЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

М. Г. Курносов[1]

В модели параллельных вычислений LogP построено аналитическое выражение времени выполнения алгоритма k параллельных цепочек для реализации корневой редукции на распределенных вычислительных системах (ВС). По построенной функциональной зависимости найдено оптимальное значение числа k параллельных цепочек, при котором алгоритм характеризуется минимальным в модели LogP временем выполнения. На основе этого создан алгоритм с оптимальным числом параллельных цепочек. Для сокращения времени ожидания корневым процессом результатов частичных редукций разработан алгоритм с адаптивным числом параллельных цепочек. Зависимость времени выполнения созданных алгоритмов от числа процессов имеет порядок роста $O(\sqrt{P})$, что эффективнее по сравнению с линейным $\Omega(P)$ временем выполнения исходного алгоритма. Алгоритмы реализованы в стандарте MPI и исследованы на вычислительных кластерах с сетями связи стандарта InfiniBand QDR.

Ключевые слова: корневая редукция, модель передачи сообщений, MPI, параллельное программирование, распределенные вычислительные системы.

1. Введение. Параллельные алгоритмы и программы для распределенных вычислительных систем (ВС) преимущественно разрабатываются в модели передачи сообщений (message-passing), в рамках которой ветви программы (процессы) взаимодействуют по средствам передачи сообщений по каналам межмашинных связей. Среди основных схем обменов значительное место по частоте использования и приходящемуся на них суммарному времени выполнения занимают коллективные операции обменов информацией (групповые, глобальные, collective communications) [1, 2]. Такие операции делятся на корневые и некорневые. В них участвуют все или подмножество всех ветвей параллельной программы (в стандарте MPI такие подмножества образуют группы процессов коммуникаторов). К *корневым* (rooted) относятся трансляционная передача ("один–всем", one-to-all) и коллекторный прием ("все–одному", all-to-one). Примерами *некорневого обмена* (unrooted) служат обмены типа "каждый–всем" (all-to-all). Для широкого класса параллельных алгоритмов время выполнения коллективных операций является критически важным и определяет их масштабируемость [3].

Работы, ориентированные на сокращение времени информационных обменов в распределенных ВС, ведутся по двум направлениям:

1) создание специализированных коммуникационных сетей, ориентированных на минимизацию времени реализации основных схем информационных обменов (Cray Gemini/Aries, IBM PERCS, Fujitsu Tofu, ТН Express-2, МВС-Экспресс, Ангара) [4–6];

2) разработка масштабируемых алгоритмов реализации коллективных операций на базе примитивов двухсторонних обменов (send/recv, в отрыве от конкретной коммуникационной технологии) [7–9].

В настоящей статье внимание уделено второму подходу, а именно оптимизации алгоритмов корневой редукции на базе примитивов двусторонних обменов.

Корневая редукция (reduction, reduce) — это коммуникационно-вычислительная коллективная операция, которая широко используется при разработке параллельных алгоритмов и программ для ВС с распределенной памятью [7, 10]. Данная операция реализует комбинирование операндов в памяти параллельных ветвей программы при помощи заданной бинарной ассоциативной и, факультативно, коммутативной операции. Если обозначить элемент в памяти ветви $i \in \{0, 1, \ldots, P-1\}$ через a_i, а через $\otimes$ — заданную бинарную операцию, то результат редукции в корневой ветви будет иметь вид $a_0 \otimes a_1 \otimes \ldots \otimes a_{P-1}$. Поддержка корневой редукции присутствует практически во всех системах параллельного программирования для ВС с распределенной памятью. Как правило, в коммуникационных библиотеках присутствуют

[1] Сибирский государственный университет телекоммуникаций и информатики, Центр параллельных вычислительных технологий, ул. Кирова, 86, 630102, Новосибирск; директор, e-mail: mkurnosov@gmail.com

соответствующие функции, а в языках параллельного программирования редукция реализуется на уровне библиотеки времени выполнения. Например, в стандарте MPI корневая редукция реализуется функцией `MPI_Reduce`, в OpenSHMEM — `shmem_int_sum_to_all`, в Coarray Fortran — операция `CO_REDUCE`, в стандартной библиотеке языка Unified Parallel C — функция `upc_all_reduce`.

Для реализации корневой редукции на распределенных ВС разработано значительное количество алгоритмов [7, 10, 11]. В их основе лежит логическая организация процессов программы в графы (деревья) различных видов и параллельное вычисление частичных результатов операции, которые передаются между ветвями посредством двусторонних обменов сообщениями (send/recv). Наибольшее распространение получили алгоритмы биномиального дерева (binomial tree), Р. Рабенсейфнера (R. Rabenseifner) [7], параллельных цепочек (k-chain), бинарного и плоского деревьев (binary, flat/linear tree).

В центре внимания нашей работы — алгоритм k параллельных цепочек [10], который характеризуется линейной зависимостью времени выполнения от числа процессов. Вопрос о выборе значения k является открытым и, как правило, решается эмпирическим путем [10]. В данной работе в модели параллельных вычислений LogP построено аналитическое выражение времени выполнения алгоритма как функция от числа процессов и количества цепочек. Построенное выражение позволило найти оптимальное значение числа k параллельных цепочек, при котором алгоритм характеризуется минимальным в модели LogP временем выполнения. На основе найденного значения k создан алгоритм с оптимальным числом параллельных цепочек. Для сокращения времени ожидания корневым процессом результатов частичных редукций разработан алгоритм с адаптивным числом параллельных цепочек. Зависимость времени выполнения созданных алгоритмов с оптимальным и адаптивным числом цепочек от числа процессов имеет порядок роста $O(\sqrt{P})$, что эффективнее по сравнению с линейным $\Omega(P)$ временем выполнения исходного алгоритма с фиксированным числом цепочек. Алгоритмы реализованы в стандарте MPI и исследованы на вычислительных кластерах с сетями связи стандарта InfiniBand QDR. Далее рассмотрение корневой редукции ведется на примере стандарта MPI.

2. Операция корневой редукции MPI_Reduce. Операция корневой редукции `MPI_Reduce` комбинирует при помощи заданной бинарной операции `op` элементы буферов отправки `sendbuf` всех процессов и записывает результат в буфер приема `recvbuf` корневого процесса `root`:

```
MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm)
```

Функция вызывается всеми процессами группы коммуникатора `comm` с указанием одинаковых значений параметров `count`, `datatype`, `op`, `root` и `comm`. Каждый процесс передает в функцию адрес буфера передачи `sendbuf`, содержащий `count` элементов типа `datatype`. Его содержимое после выполнения операции остается без изменения. Корневой процесс `root` записывает результат операции в свой локальный буфер приема `recvbuf`, тоже содержащий count элементов типа `datatype`. Буфер `recvbuf` является значимым только на стороне корневого процесса, остальные процессы могут игнорировать его и передавать в качестве адреса `NULL`.

3. Модель параллельных вычислений LogP. Модель LogP — это математическая модель ВС с распределенной памятью, в которой параллельные процессы взаимодействуют посредством двусторонних обменов короткими сообщениями фиксированного размера [12]. Модель не учитывает структуру коммуникационной сети ВС и отражает лишь показатели производительности каналов связи между процессорами.

Основные параметры модели:

- L — верхняя граница латентности канала связи (latency, delay) — время передачи сообщения фиксированного размера от одного процессора другому;
- o — промежуток времени (overhead), в течение которого процессор занят передачей или приемом сообщения (в течение этого времени процессор не может выполнять другие операции);
- g — минимальный интервал времени (gap) между последовательными передачами или приемами сообщений ($1/g$ — пропускная способность канала связи, доступная процессору);
- P — количество процессоров в системе.

Параметры L, o и g измеряются в тактах работы процессора, но без труда могут быть выражены в секундах. Неявным параметром модели является размер w сообщения. Предполагается, что сообщения имеют *небольшой размер*: одно или несколько машинных слов.

В модели LogP время t передачи сообщения от одного процессора другому выражается как $t = 2o + L$.

На рис. 1 показан пример последовательной передачи процессором 0 трех сообщений процессору 1. В нулевой момент времени процессор 0 инициирует передачу сообщения процессору 1. Для этого ему требуется o единиц времени на выдачу сообщения в сеть; далее, через L единиц времени сообщение достигает процессора 1. Процессор 1 затрачивает o единиц времени для получения сообщения из сети. Далее процессор 0 отправляет второе сообщение. Он не может начать передачу менее чем через $\max\{o, g\}$ единиц времени с момента начала передачи первого сообщения. В приведенном примере предполагается, что $g > o$. Таким образом, вторая и третья передачи сообщений процессором 0 начинаются в моменты времени g и $2g$ соответственно. Процессор 0 завершает работу в момент времени $2g + o$, а процессор 1 завершает прием данных за время $L + 2g + 2o$.

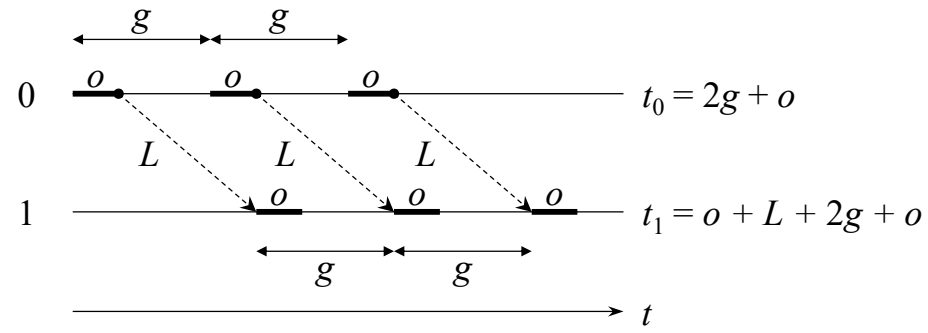


Рис. 1. Пространственно-временна́я диаграмма взаимодействия процессоров в модели LogP: процессор 0 выполняет три передачи сообщения процессору 1 ($g > o$)

Модель основана на ряде допущений. Подразумевается, что коммуникационная сеть имеет *конечную емкость* (finite capacity). Это означает, что в ней одновременно может находиться не более $\lceil L/g \rceil$ сообщений, передаваемых или принимаемых одним процессом.

Все процессоры функционируют *асинхронно*, и, как следствие, латентность (задержка) передачи сообщений между процессорами может варьироваться вплоть до значения L. Недетерминированный характер латентности может привести к тому, что сообщения, предназначенные для одного процессора, могут быть доставлены в порядке, отличном от исходного [12].

На практике некоторые параметры модели могут быть опущены. В частности, если процессоры относительно редко обмениваются сообщениями, то пропускную способность $1/g$ сети и ограничения на ее емкость можно игнорировать. В то же время, в некоторых системах накладные расходы o могут превышать значение g, что позволяет не учитывать последний параметр.

Для оценки издержек на выполнение локальной редукции процессором модель LogP расширяют параметром γ — время выполнения бинарной операции редукции, приходящееся на один байт сообщения.

Перед оценкой времени выполнения, как правило, фиксируют отношения между значениями L, o и g. Например, полагают, что имеют место неравенства $o \leqslant g < L$.

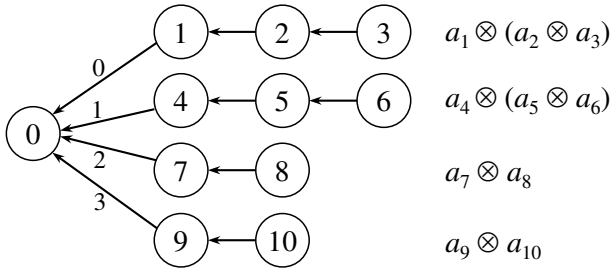


Рис. 2. Дерево обменов алгоритма параллельных цепочек: $P = 11$, $k = 4$, $root = 0$ (стрелками показаны направления передачи сообщений, справа результаты редукций в цепочках)

4. Алгоритм параллельных цепочек. Алгоритм k параллельных цепочек (k-chain algorithm) организует процессы в k цепочек (конвейеров), которые передают свои результаты корневому процессу с номером $root$, где k — это фиксированный параметр алгоритма. Например, в библиотеке Open MPI по умолчанию значение $k = 4$ (подсистема коллективных операций tuned) [10]. Если число процессов $P - 1$ не кратно значению k, то остаток $(P-1)\%k$ процессов распределяется по первым $(P-1)\%k$ цепочкам, в каждую добавляется по одному процессу. На рис. 2 первые две цепочки получили по одному дополнительному процессу.

Алгоритм применим только в том случае, если из $P - 1$ процессов можно организовать k непустых цепочек. Формально, должно выполняться условие $1 \leqslant k \leqslant P - 1$. При $k = 1$ алгоритм параллельных цепочек сводится к *конвейерному алгоритму* (pipeline reduce), а при $k = P - 1$ — к алгоритму *плоского дерева* (flat/liner tree).

Будем называть цепочку *короткой*, если она содержит $\lfloor (P - 1)/k \rfloor$ процессов. Аналогично, *длинной*

будем называть цепочку, если она содержит $\lfloor (P-1)/k \rfloor + 1$ процессов. Число длинных цепочек равно $(P-1) \,\%\, k$, количество коротких равно $k - (P-1) \,\%\, k$.

Корневой процесс выполняет k приемов результатов цепочек и столько же локальных редукций. Сначала выполняется прием результатов от длинных цепочек, а затем от коротких. Некорневые процессы по своему номеру определяют свое положение в цепочке — номера следующего и предшествующего процессов.

Выразим в модели LogP время выполнения алгоритма для сообщений размера m байт. Очевидно, что время выполнения корневого процесса определяет время работы всего алгоритма. На рис. 3 приведена пространственно-временна́я диаграмма выполнения в модели LogP алгоритма для $P = 11$, $k = 4$. В момент времени $\lfloor (P-1)/k \rfloor (2o + L + m\gamma)$ головные процессы длинных цепочек завершают выполнение локальной редукции и готовы передавать результат корневому процессу. На рис. 3 головной процесс 1 завершил выполнение локальной редукции в момент времени $4o + 2L + 2m\gamma$.

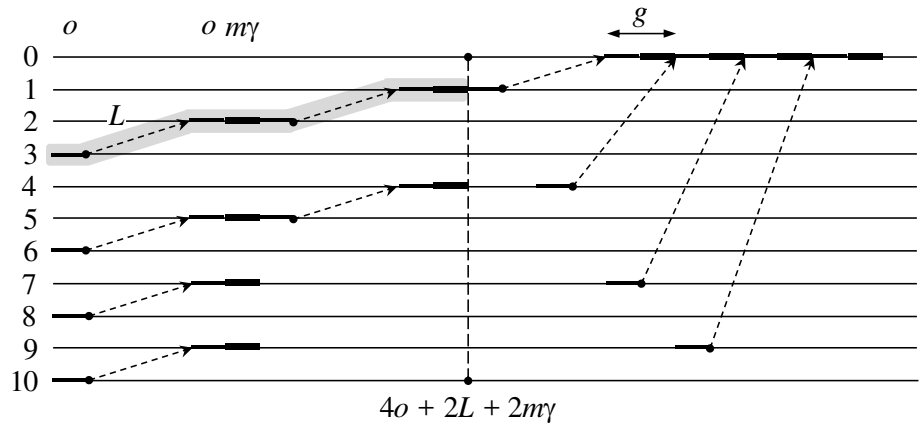


Рис. 3. Пространственно-временна́я диаграмма выполнения в модели LogP алгоритма параллельных цепочек ($P = 11$, $k = 4$, $root = 0$)

Корневой процесс получает результат первой длинной цепочки и завершает выполнение локальной редукции в момент времени $\lfloor (P-1)/k \rfloor (2o + L + m\gamma) + 2o + L + m\gamma$.

Оставшиеся $k - 1$ головных процессов к этому моменту уже готовы передавать свои результаты. Корневой процесс не может начать очередной прием ранее, чем через $\max\{o + m\gamma, g\}$ единиц времени. Поэтому суммарное время $t(m)$ выполнения корневого процесса имеет вид

$$t(m) = \lfloor (P-1)/k \rfloor (2o + L + m\gamma) + (k-2)\max\{o + m\gamma, g\} + 3o + L + 2m\gamma.$$

Заметим, если все цепочки будут иметь одинаковую длину, полученная оценка времени выполнения алгоритма останется справедливой.

Из полученного выражения $t(m)$ видно, что алгоритм параллельных цепочек с фиксированным значением k характеризуется *линейной зависимостью* времени выполнения от числа P процессов. Другими словами, скорость роста функции $t(m)$ имеет порядок $\Omega(P)$, что ограничивает применимость данного алгоритма в большемасштабных ВС.

5. Выбор оптимального числа цепочек. Найдем число k цепочек при котором алгоритм характеризуется наименьшим в модели LogP временем выполнения. Запишем время алгоритма параллельных цепочек как функцию от параметра k:

$$t(k) = \big((P-1)/k + 1\big)(2o + L + m\gamma) + (k-2)\max\{o + m\gamma, g\} + o + m\gamma.$$

Далее будем исходить из предположения, что $o + m\gamma > g$. Это допущение не повлияет на окончательный результат. Для краткости изложения обозначим $a = 2o + L + m\gamma$ и $b = o + m\gamma$. Тогда

$$t(k) = \big((P-1)/k + 1\big)a + (k-1)b. \tag{1}$$

Найдем значение аргумента k, при котором функция $t(k)$ принимает минимальное значение:

$$t'(k) = -a(P-1)/k^2 + b, \quad -a(P-1)/k^2 + b = 0, \quad k = \sqrt{a(P-1)/b}\,.$$

Последнее равенство справедливо, так как $a/b > 1$. Подставим найденное выражение для k в (1) и запишем время выполнения алгоритма с оптимальным числом цепочек как функцию от параметра P:

$$t(P) = \left(\frac{P-1}{\sqrt{a(P-1)/b}} + 1\right) a + \left(\sqrt{a(P-1)/b} - 1\right) b = 2\sqrt{ab} \cdot \sqrt{P-1} + a - b.$$

Если значения параметров o, L, γ заранее известны (предварительно измерены для заданной ВС), то оптимальное значение k определяется однозначно. Если значения параметров модели LogP не известны, то можно выработать рекомендации относительно выбора субоптимального значения параметра k.

Если сообщение имеет минимальный размер $m = 1$, то справедливо неравенство

$$1 < a/b = 1 + \frac{o+L}{o+m\gamma} = 1 + \frac{o+L}{o+\gamma}. \tag{2}$$

В этом случае верхняя граница значения k определяется величиной отношения L/γ. Заметим, что значение параметра γ может быть сколь угодно большим, так как стандарт MPI и другие системы параллельного программирования предоставляют возможность задавать пользовательские операции редукции.

При передаче сообщений значительных размеров имеет место неравенство $o + L < o + m\gamma$, отсюда

$$1 < a/b = 1 + \frac{o+L}{o+m\gamma} < 2. \tag{3}$$

Учитывая (2) и (3), можно рекомендовать на практике выбирать (суб)оптимальное значение k^* числа цепочек исходя из следующего неравенства: $k^* \geqslant \lceil \sqrt{P-1} \rceil$.

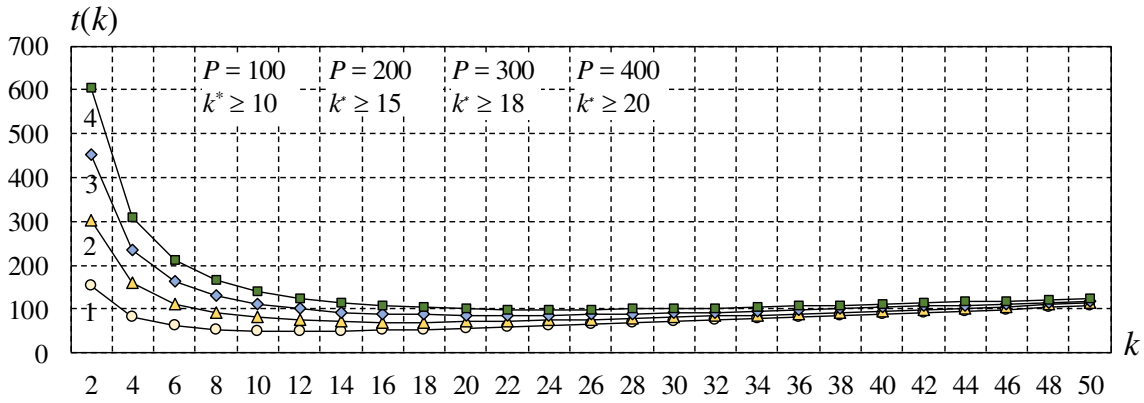


Рис. 4. Зависимость времени $t(k)$ выполнения алгоритма от числа k цепочек (модель LogP, $o + m\gamma > g$, $a = 3$, $b = 2$): 1) $P = 100$, $k^* = 10$; 2) $P = 200$, $k^* = 15$; 3) $P = 300$, $k^* = 18$; 4) $P = 400$, $k^* = 20$

На рис. 4 показана построенная в модели LogP зависимость времени $t(k)$ работы алгоритма от числа k цепочек. Нетрудно заметить, что кривые имеют экстремумы в окрестностях точки $k^* = \lceil \sqrt{P-1} \rceil$.

На рис. 5 показана зависимость времени $t(k)$ выполнения алгоритма от числа k параллельных цепочек на вычислительном кластере с сетью связи InfiniBand QDR (48 процессов, 6 двухпроцессорных узлов, коммутатор Mellanox InfiniScale IV IS5030 QDR, сетевые адаптеры Mellanox MT26428 InfiniBand QDR, операционная система GNU/Linux CentOS 6.5 x86-64, библиотека MVAPICH2 2.2a).

Для каждого значения k рассчитывалось среднее время выполнения алгоритма по результатам 10 запусков. Для устранения рассинхронизации процессов запуск операций в них выполнялся по показаниям глобальных часов. Данная методика описана в работах [13, 14]. Видно, что рекомендуемое значение $k^* = 7$ обеспечивает время выполнения алгоритма, близкое к оптимальному. Алгоритм реализован в стандарте MPI и находится в открытом доступе: `https://github.com/mkurnosov/reduce-kchain.git`.

6. Оптимизация алгоритма k параллельных цепочек. Из диаграммы на рис. 3 видно, что короткие цепочки (процессы 7 и 9) успевают вычислить результаты до момента завершения работы длинных цепочек. Время выполнения алгоритма k-цепочек можно сократить, если пересмотреть схему распределения процессов по цепочкам. Корневому процессу следует вначале принимать данные от коротких цепочек, а затем от длинных. Это позволит совместить передачу корню результатов некоторых коротких цепочек и передачу сообщений в длинных цепочках. На рис. 6 изображена диаграмма выполнения алгоритма, в котором первые две цепочки содержат по два процесса, а следующие длинные цепочки — по три.

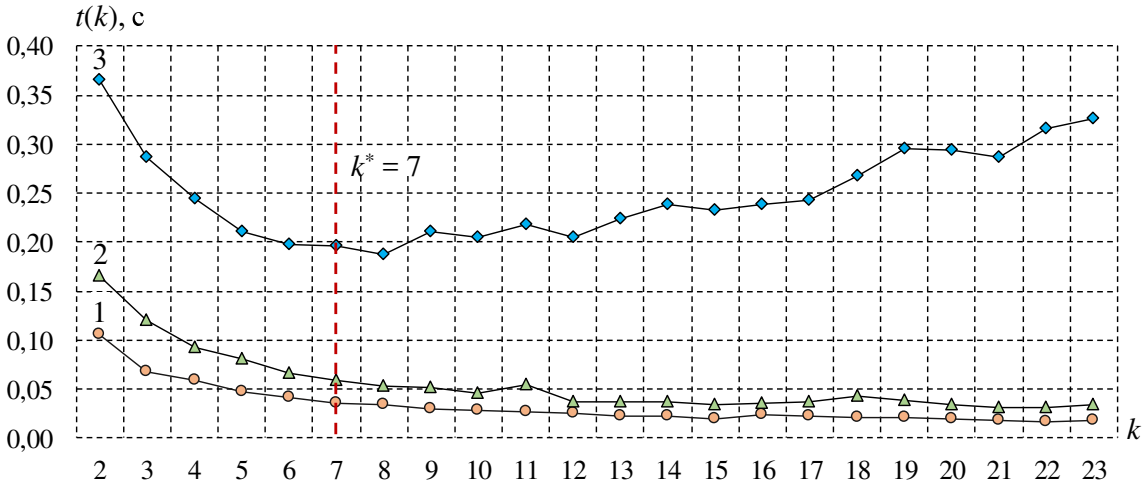


Рис. 5. Зависимость времени $t(k)$ выполнения алгоритма от числа k параллельных цепочек ($P = 48$, 6 двухпроцессорных узлов, сеть связи InfiniBand QDR, MVAPICH2 2.2a, суммирование элементов типа double): 1) $m = 1$; 2) $m = 1024$; 3) $m = 1\,048\,576$

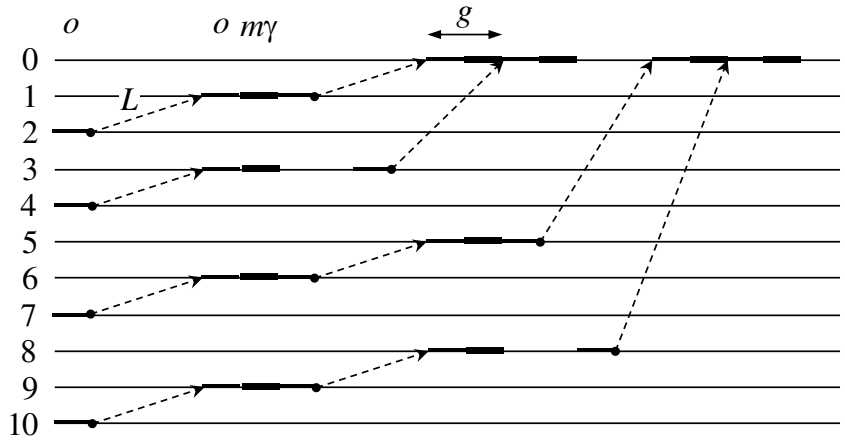


Рис. 6. Пространственно-временна́я диаграмма выполнения в модели LogP алгоритма параллельных цепочек ($P = 11$, $k = 4$, $root = 0$): порядок приема результатов корневым процессом — короткие цепочки, длинные цепочки

Корневой процесс получает результат первой короткой цепочки и завершает выполнение локальной редукции в момент времени $\lfloor (P-1)/k \rfloor (2o + L + m\gamma)$.

На рис. 6 результат первой короткой цепочки (процесс 1) достигает корня в момент времени $3o + 2L + m\gamma$, в это же время головной процесс первой длинной цепочки (процесс 5) получает сообщение от процесса 6. Начиная отсчет с этого момента времени, результат первой длинной цепочки будет вычислен и доставлен процессом 5 корню не менее чем через $2o + m\gamma + L$ единиц времени. В это же время корневой процесс выполняет прием результатов двух коротких цепочек. На это требуется $2\max\{o + m\gamma, g\}$ единиц времени. Корневой процесс после приема результата первой короткой цепочки получит результат первой длинной цепочки (процесс 5) не ранее чем через $\max\{2\max\{o + m\gamma, g\}, 2o + L + m\gamma\}$ единиц времени. На прием результатов длинных цепочек и их обработку требуется еще $\max\{o + m\gamma, g\} + o + m\gamma$ единиц времени. Учитывая все временны́е издержки, запишем время работы корневого процесса для примера на рис. 6:

$$t(m) = 3o + 2L + \max\{2\max\{o + m\gamma, g\}, 2o + L + m\gamma\} + \max\{o + m\gamma, g\} + o + m\gamma.$$

Если принять $o + m\gamma > g$, то время выполнения корневого процесса составит

$$t(m) = 7o + 2L + 4m\gamma + \max\{m\gamma, L\}.$$

Если же корневой процесс будет сначала принимать результаты длинных цепочек (этот случай рассмотрен ранее, рис. 3), то время выполнения корневого процесса будет $t(m) = 9o + 3L + 6m\gamma$. Это значение на

$2o + L + 2m\gamma - \max\{m\gamma, L\}$ единиц времени больше времени работы алгоритма с оптимизированным распределением процессов.

В общем случае алгоритм параллельных цепочек с оптимизированным распределением процессов выполняется за время

$$t(m) = (u-1)(2o + L + m\gamma) + \max\left\{s \cdot \max\{o + m\gamma, g\}, 2o + L + m\gamma\right\} + \\ + (v-1)\max\{o + m\gamma, g\} + 2o + m\gamma,$$

где $u = \lfloor (P-1)/k \rfloor$ — число процессов в короткой цепочке, а $s = k - (P-1) \,\%\, k$ и $v = (P-1) \,\%\, k$ — количество коротких и длинных цепочек соответственно.

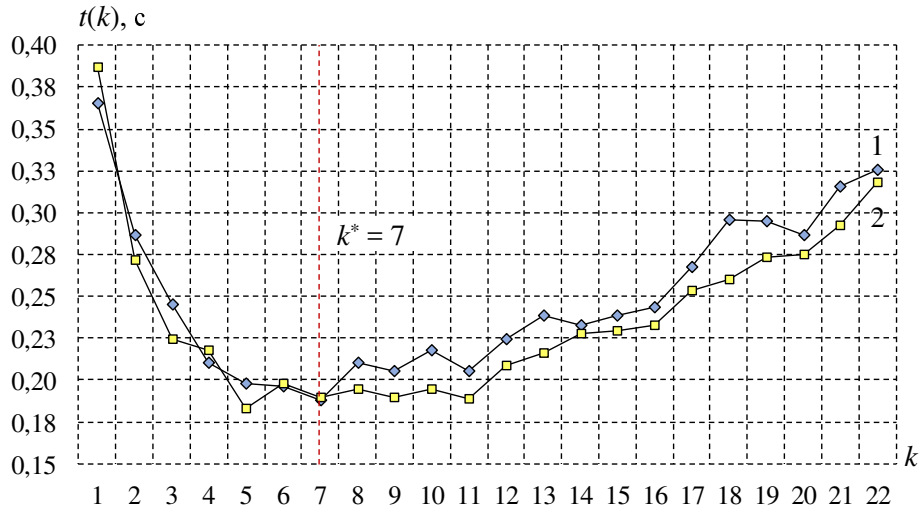


Рис. 7. Зависимость времени $t(k)$ выполнения алгоритма от числа k параллельных цепочек ($P = 48$, 6 двухпроцессорных узлов, сеть связи InfiniBand QDR, MVAPICH2 2.2a, бинарная операция — сложение MPI_SUM; сообщение — 1 048 576 элементов типа double): 1) порядок цепочек — длинные, короткие; 2) порядок цепочек — короткие, длинные

На рис. 7 показано время выполнения алгоритма с двумя вариантами получения результатов цепочек корневым процессом на кластере с сетью связи InfiniBand QDR.

Нетрудно заметить (рис. 6), что первая длинная цепочка отстает от первой короткой на одну операцию — на время $2o + L + m\gamma$. У корневого процесса имеется возможность использовать это время для получения результатов от коротких цепочек. Предложим алгоритм с монотонно возрастающей длиной цепочек, чтобы корневой процесс не ожидал поступления результатов, а цепочки не обгоняли по времени корневой процесс.

7. Алгоритм с адаптивным числом и длиной цепочек. Имеющиеся $P-1$ процессов распределяются по цепочкам так, чтобы первая цепочка содержала один процесс, вторая два, третья — три и т. д. (рис. 8).

Таким образом, длины цепочек образуют арифметическую прогрессию, в которой последняя цепочка k содержит k процессов: $P-1 = 1+2+\ldots+k$. Следовательно, $P-1 = (k^2+k)/2$. Решая относительно k последнее уравнение, получаем

$$k = \frac{1}{2}\left(\sqrt{8(P-1)+1} - 1\right). \tag{4}$$

На практике значение k можно округлять до ближайшего целого снизу: $k = \left\lfloor \frac{1}{2}\left(\sqrt{8(P-1)+1} - 1\right)\right\rfloor$.

Заметим, что значение $P-1$ может быть не представимо в виде суммы $P-1 = 1+2+\ldots+k$, поэтому оставшиеся $P-1-\left(k^2+k\right)/2$ процессы образуют дополнительную цепочку, которая передает свои результаты корневому процессу в последнюю очередь. Число таких процессов не превышает k.

В предложенном алгоритме с адаптивным числом цепочек корневой процесс имеет номер 0. Этого всегда можно добиться введением виртуальной нумерации процессов [7]. Процесс $r \in \{1, 2, \ldots, P-1\}$ находится в цепочке с номером d, где $d = \left\lceil \frac{1}{2}\left(\sqrt{8(P-1)+1} - 1\right)\right\rceil$.

Первым (головным) в цепочке d является процесс с номером $h = \big(d^2 - d\big)/2 + 1$, последний процесс цепочки имеет номер z, где $z = \begin{cases} \big(d^2 - d\big)/2, & \text{если } (d^2 - d)/2 < P, \\ P - 1, & \text{иначе.} \end{cases}$

Время выполнения адаптивного алгоритма корневой редукции на рис. 8 равно $t(m) = 2o + L + m\gamma + 3\max\{o + m\gamma, g\}$.

В общем случае время выполнения алгоритма с адаптивным числом цепочек равно $t(m) = 2o + L + m\gamma + (k-1)\max\{o + m\gamma, g\}$.

Учитывая дополнительную цепочку, запишем итоговое время работы алгоритма: $t(m) = 2o + L + m\gamma + k \cdot \max\{o + m\gamma, g\}$.

Подставив найденное значение (4) числа цепочек в последнее выражение $t(m)$, получим

$$t(m) = 2o + L + m\gamma + \frac{1}{2}\left(\sqrt{8(P-1)+1} - 1\right)\max\{o + m\gamma, g\}.$$

Как и ранее, если положить $o + m\gamma > g$, то

$$t(m) = \frac{1}{2}\left(\sqrt{8(P-1)+1} - 1\right)b + a.$$

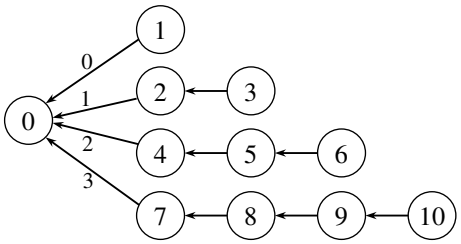


Рис. 8. Дерево обменов алгоритма с адаптивным числом параллельных цепочек: $P = 11$, $k = 4$, $root = 0$ (стрелками показаны направления передачи сообщений)

8. Анализ масштабируемости алгоритмов. Выше показано, что алгоритм с фиксированным числом k параллельных цепочек характеризуется *линейной* зависимостью времени $t(k, P)$ выполнения от числа P процессов, что ограничивает его применимость в большемасштабных вычислительных системах:

$$t(k, P) = (\lfloor (P-1)/k \rfloor + 1)\, a + (k-1)b.$$

Разработанные алгоритмы с оптимальным и адаптивным числом цепочек масштабируются эффективнее — время их выполнения имеет одинаковый, степенной порядок роста $O(\sqrt{P})$. Время выполнения алгоритма с оптимальным числом цепочек: $t(P) = 2\sqrt{ab} \cdot \sqrt{P-1} + a - b$.

Время выполнения алгоритма с адаптивным числом цепочек:

$$t(P) = \frac{1}{2}\left(\sqrt{8(P-1)+1} - 1\right)\, b + a = b\sqrt{2(P-1)+1/4} - b/2 + a.$$

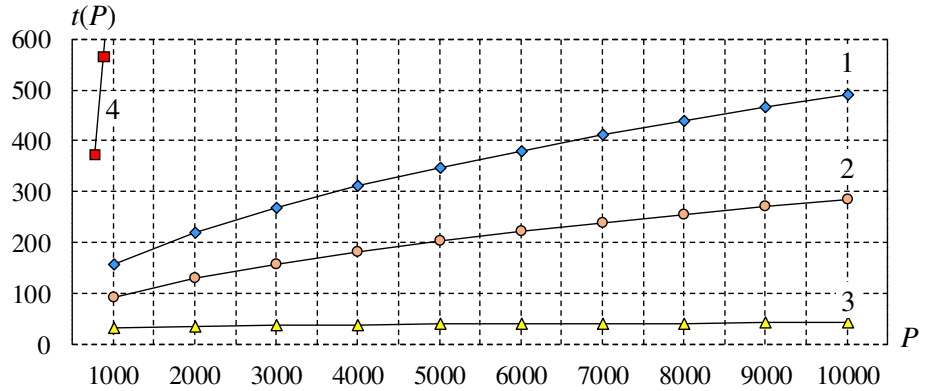


Рис. 9. Зависимость в модели LogP времени выполнения алгоритмов от числа P процессов ($a = 3$, $b = 2$): 1) алгоритм с оптимальным числом цепочек; 2) алгоритм с адаптивным числом цепочек; 3) алгоритм биномиального дерева; 4) алгоритм с фиксированным числом цепочек $k = 4$

На рис. 9 показана зависимость в модели LogP времени выполнения разработанных алгоритмов с оптимальным и адаптивным числом цепочек от числа процессов. Для сравнения приведено время выполнения известного алгоритма биномиального дерева (binomial tree), время выполнения которого $a\,\log_2 P + b$ логарифмически зависит от числа P процессов.

На рис. 10 и 11 приведены результаты выполнения алгоритмов с оптимальным и адаптивным числом цепочек на вычислительных кластерах с сетью связи стандарта InfiniBand QDR. Можно сделать вывод, что алгоритм с адаптивным числом цепочек целесообразно использовать при сообщениях значительных размеров.

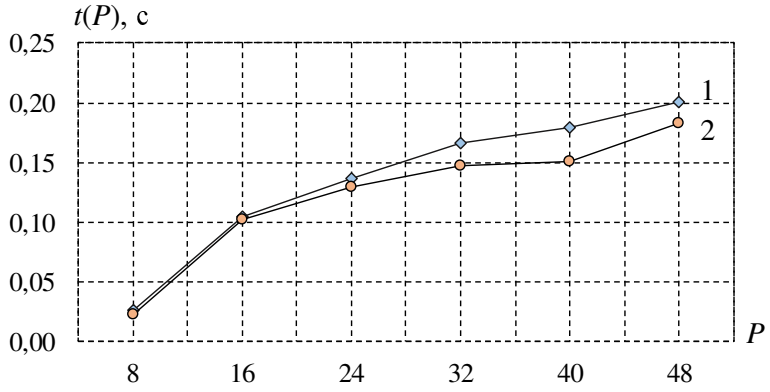
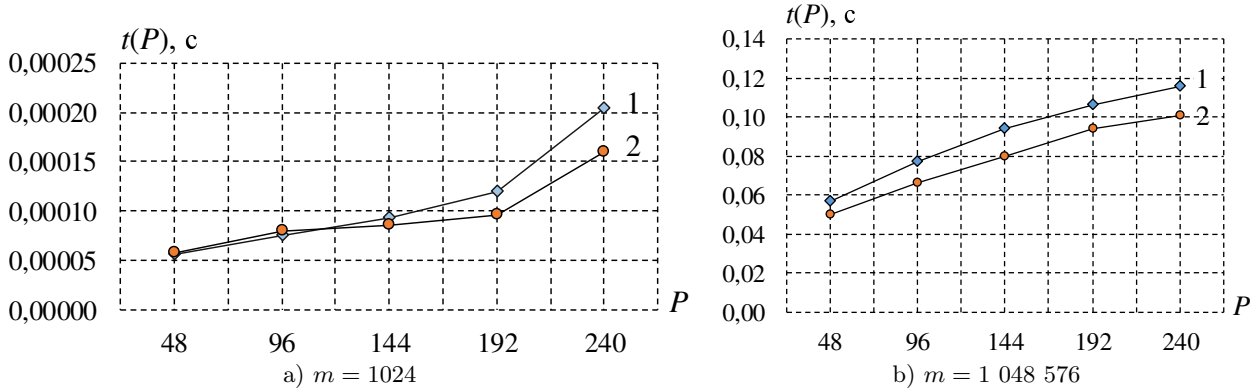


Рис. 10. Зависимость времени выполнения алгоритмов от числа P процессов (двухпроцессорные узлы — два процессора Intel Quad Xeon E5620, сеть связи InfiniBand QDR, библиотека MVAPICH2 2.2a, бинарная операция — сложение MPI_SUM; сообщение — 1 048 576 элементов типа double): 1) алгоритм с оптимальным числом цепочек; 2) алгоритм с адаптивным числом цепочек



a) $m = 1024$ b) $m = 1\ 048\ 576$

Рис. 11. Зависимость времени $t(P)$ выполнения алгоритмов от числа P процессов (двухпроцессорные узлы HP XL230a Gen9 — два 12-ядерных процессора Intel Xeon E5 2680, библиотека Intel MPI 5.0.1, GNU/Linux SUSE Linux Enterprise Server 11 SP4, суммирование элементов типа double): 1) алгоритм с оптимальным числом цепочек; 2) алгоритм с адаптивным числом цепочек

9. Заключение. Построенное в модели параллельных вычислений LogP аналитическое выражение времени выполнения известного алгоритма k параллельных цепочек позволило найти оптимальное число цепочек и разработать новый алгоритм, у которого зависимость времени выполнения от числа процессов имеет меньший порядок роста, а именно $O\left(\sqrt{P}\right)$, что предпочтительнее с позиций масштабируемости алгоритмов на распределенных ВС.

Для сокращения времени ожидания корневым процессом результатов частичных редукций предложен алгоритм с адаптивным числом и длиной цепочек.

Оба разработанных алгоритма реализованы на базе двусторонних операций стандарта MPI. Эксперименты на вычислительных кластерах с сетями связи стандарта InfiniBand QDR подтвердили результаты теоретического анализа.

Заметим, что интерес представляет сам подход, продемонстрированный на примере оптимизации алгоритма параллельных цепочек. Суть подхода заключается в построении аналитических выражений зависимости показателей эффективности алгоритмов в моделях параллельных вычислений (Дж. Хонки, LogP, LogGP, BSP и др.) и в применении математических методов для определения (суб)оптимальных параметров алгоритмов. Выбор модели обусловлен спецификой алгоритма и целевой ВС. Например, если алгоритм реализует группировку сообщений в пакеты больших размеров, то целесообразно использовать модель LogGP, которая в явном виде учитывает издержки на передачу сообщений больших размеров. Аналогичным образом возможны постановки задач о выборе оптимальных размеров m сообщения, числа P процессов, времени γ вычисления бинарной операции и т.д.

Работа выполнена при поддержке РФФИ (коды проектов 15–37–20113 и 15–07–00653).

СПИСОК ЛИТЕРАТУРЫ

1. *Хорошевский В.Г.* Распределенные вычислительные системы с программируемой структурой // Вестник СибГУТИ. 2010. **10**, № 2. 3–41.
2. *Hoefler T., Moor D.* Energy, memory, and runtime tradeoffs for implementing collective communication operations // Journal of Supercomputing Frontiers and Innovations. 2014. **1**, N 2. 58–75.
3. *Balaji P., Buntinas D., Goodell D., Gropp W., Hoefler T., Kumar S., Lusk E., Thakur R., Träff J.* MPI on millions of cores // Parallel Processing Letters. 2011. **21**, N 1. 45–60.
4. *Alverson R., Roweth D., Kaplan L.* The Gemini System Interconnect // Proc. 18th IEEE Symposium on High Performance Interconnects. Washington, DC: IEEE Press, 2010. 83–87.
5. *Chen D., Eisley N.A., Heidelberger P., Senger R., et al.* The IBM Blue Gene/Q interconnection network and message unit // Proc. 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. New York: ACM Press, 2011. doi 10.1145/2063384.2063419.
6. *Левин В.К., Четверушкин Б.Н., Елизаров Г.С., Горбунов В.С., Лацис А.О., Корнеев В.В., Соколов А.А., Андрюшин Д.В., Климов Ю.А.* Коммуникационная сеть МВС-Экспресс // Информационные технологии и вычислительные системы. 2014. № 1. 10–24.
7. *Thakur R., Rabenseifner R., Gropp W.* Optimization of collective communication operations in MPICH // Int. Journal of High Performance Computing Applications. 2005. **19**, N 1. 49–66.
8. *Курносов М.Г.* Алгоритмы трансляционно-циклических информационных обменов в иерархических распределенных вычислительных системах // Вестник компьютерных и информационных технологий. 2011. № 5. 27–34.
9. *Ma T., Bosilca G., Bouteiller A., Dongarra J.J.* Kernel-assisted and topology-aware MPI collective communications on Multicore/Many-Core Platforms // Journal of Parallel and Distributed Computing. 2013. **73**, N 7. 1000–1010.
10. *Fagg G., Pješivac-Grbović J., Bosilca G., Dongarra J., Jeannot E.* Flexible collective communication tuning architecture applied to Open MPI // Proc. of Euro PVM/MPI. Julich: Forschungszentrum, 2006. 1–10.
11. *Pješivac-Grbović J., Angskun T., Bosilca G., et al.* Performance analysis of MPI collective operations // Cluster Computing. 2007. **10**, N 2. 127–143.
12. *Culler D., Karp R., Patterson D., et al.* LogP: towards a realistic model of parallel computation // ACM SIGPLAN Notices. 1993. **28**, N 7. 1–12.
13. *Worsch T., Reussner R., Augustin W.* On benchmarking collective MPI operations // Lecture Notes in Computer Science. Vol. 2474. Heidelberg: Springer, 2002. 271–279.
14. *Курносов М.Г.* MPIPerf: пакет оценки эффективности коммуникационных функций стандарта MPI // Вестник Нижегородского университета им. Н.И. Лобачевского. 2012. № 5/2. 385–391.

Поступила в редакцию
8.07.2016

Analysis and Optimization of a k-Chain Reduction Algorithm for Distributed Computer Systems

M. G. Kurnosov [1]

[1] *Siberian State University of Telecommunications and Information Sciences, Center for Parallel Computational Technologies; ulitsa Kirova 86, Novosibirsk, 630102, Russia; Director of Center, e-mail: mkurnosov@gmail.com*

Received July 8, 2016

Abstract: In the LogP model of parallel computing, an analytical expression of the k-chain algorithm's execution time is derived. The optimal value of k in the LogP model is found. A new algorithm based on the optimal value of k is developed. For the reduction of root process's waiting time, an algorithm with an adaptive number of chains is proposed. The dependence of the execution time of the proposed algorithm on the number of processes has a growth rate of $O(\sqrt{P})$, which is more efficient compared to the linear running time of the original k-chain algorithm. The proposed algorithms are implemented in the MPI standard and studied on computer clusters with InfiniBand QDR networks.

Keywords: root reduction, message passing models, MPI, parallel programming, distributed computer systems.

References

1. V. G. Khoroshevsky, "Distributed Programmable Structure Computer Systems," Vestn. Sib. Gos. Univ. Telekommun. Inform. **10** (2), 3–41 (2010).

2. T. Hoefler and D. Moor, "Energy, Memory, and Runtime Tradeoffs for Implementing Collective Communication Operations," J. Supercomput. Frontiers Innovations **1** (2), 58–75 (2014).

3. P. Balaji, D. Buntinas, D. Goodell, et al., "MPI on Millions of Cores," Parallel Process. Lett. **21** (1), 45–60 (2011).

4. R. Alverson, D. Roweth, and L. Kaplan, "The Gemini System Interconnect," in *Proc. 18th IEEE Symposium on High Performance Interconnects, Mountain View, USA, August 18–20, 2010* (IEEE Press, Washington, DC, 2010), pp. 83–87.

5. D. Chen, N. A. Eisley, P. Heidelberger, et al., "The IBM Blue Gene/Q Interconnection Network and Message Unit," in *Proc. 2011 Int. Conf. for High Performance Computing, Networking, Storage and Analysis, Seattle, USA, November 12–18, 2011* (ACM Press, New York, 2011), doi 10.1145/2063384.2063419

6. V. K. Levin, B. N. Chetverushkin, G. S. Elizarov, et al., "Communication Fabric MVS-Express," Inform. Tekhnol. Vychisl. Sistemy, No. 1, 10–24 (2014).

7. R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," Int. J. High Perform. Comput. Appl. **19** (1), 49–66 (2005).

8. M. G. Kurnosov, "Algorithms of Transmission-Cyclical Information Exchanges in the Hierarchical Distributed Computing Systems," Vestn. Komp'yut. Inform. Tekhnol., No. 5, 27–34 (2011).

9. T. Ma, G. Bosilca, A. Bouteiller, and J. J. Dongarra, "Kernel-Assisted and Topology-Aware MPI Collective Communications on Multicore/Many-core Platforms," J. Parallel Distrib. Comput. **73** (7), 1000–1010 (2013).

10. G. Fagg, J. Pješivac-Grbović, G. Bosilca, et al., "Flexible Collective Communication Tuning Architecture Applied to Open MPI," in *Proc. of EuroPVM/MPI, Bonn, Germany, September 17–20, 2006* (Forschungszentrum, Julich, 2006), pp. 1–10.

11. J. Pješivac-Grbović, T. Angskun, G. Bosilca, et al., "Performance Analysis of MPI Collective Operations," Cluster Comput. **10** (2), 127–143 (2007).

12. D. Culler, R. Karp, D. Patterson, et al., "LogP: Towards a Realistic Model of Parallel Computation," ACM SIGPLAN Notices **28** (7), 1–12 (1993).

13. T. Worsch, R. Reussner, and W. Augustin, "On Benchmarking Collective MPI Operations," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2002), Vol. 2474, pp. 271–279.

14. M. G. Kurnosov, "MPIPerf: A Toolkit for Benchmarking MPI Libraries," Vestn. Lobachevskii Univ. Nizhni Novgorod, No. 5/2, 385–391 (2012).