Адаптивный алгоритм барьерной синхронизации процессов в MPI-программах на основе аналитических оценок времени выполнения информационных обменов в модели LogP

В. В. Жариков[1], А. А. Пазников[2]
Санкт-Петербургский государственный электротехнический университет
«ЛЭТИ» им. В. И. Ульянова (Ленина)
[1]zharikov.vitalik@yandex.ru, [2]apaznikov@gmail.com

***Аннотация*. Разработан алгоритм барьерной синхронизации, обеспечивающий субоптимальный выбор схемы реализации барьерной синхронизации ветвей параллельных MPI-программ на основе оценки времени выполнения операции в модели LogP параллельных вычислений. Предложенный адаптивный алгоритм реализован в стандарте MPI. Приводятся результаты натурных экспериментов на кластерных вычислительных системах. Результаты экспериментов позволяют проследить зависимость выбора алгоритма реализации барьерной синхронизации от параметров модели LogP. Созданный алгоритм позволяет сократить среднее время выполнения барьерной синхронизации на 4%, по сравнению с распространенными детерминированными алгоритмами. Построенные аналитические оценки времени согласуются с результатами натурных экспериментов.**

Ключевые слова: коллективные информационные обмены; барьерная синхронизация; распределённые вычислительные системы; LogP; MPI; параллельное программирование

I. Введение

В настоящее время для решения сложных задач применяются распределенные вычислительные системы (ВС). Распределенная ВС представляется множеством элементарных машин (ЭМ), взаимодействующих между собой через коммуникационную среду [1]. ЭМ может быть представлена как процессорным ядром, так и многоядерным SMP/NUMA-узлом, укомплектованным специализированными ускорителями (например, графическими процессорами). Количество ЭМ в системе может достигать нескольких миллионов. Так, например, суперкомпьютер Sunway TaihuLight, возглавляющий рейтинг TOP500, включает в себя более 10 миллионов процессорных ядер. Для выполнения параллельных программ на таких системах в большинстве случаев используется модель передачи сообщений, представленная стандартом MPI (Message Passing Interface).

В значительной части существующих параллельных MPI-программ используются коллективные информационные обмены (групповые, collective), в которых участвуют все ветви (процессы) параллельной программы. На коллективные обмены приходится значительная доля суммарного времени выполнения программ [2]. Поэтому эффективность реализации коллективных операций существенным образом влияет на масштабируемость параллельных программ, в связи с чем задача оптимизации коллективных обменов является актуальной. Основное направление работ по оптимизации коллективных обменов – разработка масштабируемых алгоритмов реализации коллективных операций на основе дифференцированных (двусторонних, точка-точка, point-to-point) обменов.

Одной из наиболее распространенных коллективных операций является барьерная синхронизация (барьер, barrier synchronization) – коллективная операция, реализующая ожидание процессами выполнения условия, когда каждый из них достигнет определенной точки в программе. Выполнение барьера (рис. 1) включает в себя шаг захвата (capture), по достижении которого процесс переходит в состояние ожидания. После того, как все процессы захвачены, выполняется процедура освобождения (release), которая выводит процессы из ожидания.

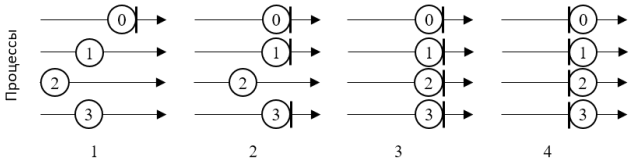

Рис. 1. Стадии выполнения барьерной синхронизации

Работа выполнена при поддержке Совета по грантам Президента РФ для государственной поддержки молодых российских ученых (проект СП-4971.2018.5). Публикация выполнена в рамках государственной работы «Инициативные научные проекты» базовой части государственного задания Министерства образования и науки Российской Федерации (ЗАДАНИЕ № 2.6553.2017/БЧ).

II. Обзор существующих работ

К наиболее распространенным базовым алгоритмам барьерной синхронизации относятся Central Counter, Dissemination, Combining Tree, Tournament и алгоритм Брука. Большая часть современных работ направлена на развитие данных алгоритмов.

Авторы статьи [3] предлагают адаптивные версии алгоритма Combining Tree с целью уменьшения объема накладных расходов на каждом узле дерева, включая захват и освобождение. В работе [4] предлагается оптимизированная версия алгоритма Брука для числа процессов кратного двум. Алгоритмы, представленные в работе [5], ориентированы на применение в сетях InfiniBand. В работе [6] предлагается метод на основе модели LogP параллельных вычислений с целью минимизации потребления энергии и метод масштабирования частоты процессоров для алгоритмов Tournament и Central Counter. В статье [7] предлагаются методы верификации алгоритмов Central Counter, Combining Tree, Dissemination и Tournament. В работе [8] проводится анализ эффективности алгоритмов барьерной синхронизации для языка Java. В библиотеке MPICH применяется Dissemination, в Open MPI реализованы адаптивные схемы, которые осуществляют выбор между Central Counter и Combining Tree, в зависимости от числа процессов. При большом количестве процессов применяется Combining Tree.

Недостатком детерминированного выбора алгоритма является то, что конкретные алгоритмы могут не обеспечивать минимальное время выполнения барьерной синхронизации. Для этого может применяться адаптивный подход, реализующий динамический выбор алгоритма коллективной операции в процессе выполнения программы. При выборе алгоритма необходимо учитывать размер сообщений, интенсивность взаимодействия между параллельными ветвями и архитектурные свойства ВС [9].

В большинстве существующих адаптивных схем выполнения коммуникационных обменов для оценки времени выполнения применяется модель Хокни, которая недостаточно точно отражает реализацию дифференцированных обменов. В данной работе используется модель LogP [10], которая позволяет с высокой точностью оценить реальное время выполнения операций дифференцированных обменов в распределенных ВС. Оптимизация выполняется с целью минимизации времени выполнения барьерной синхронизации.

III. Модель LogP параллельных вычислений

Основными параметрами модели LogP являются L – латентность (latency) среды связи, o – накладные расходы времени (overhead), на передачу или приемом сообщения, g – временной интервал (gap) между двумя передачами или приемами сообщений, P – количество процессов в системе.

Введем обозначения: o_r, o_s – промежуток времени, в течение которого процесс занят приемом и передачей сообщения [11]. Время отправки и получения одного сообщения можно оценить как $o_s + L + o_r$, время отправки n сообщений: $o_s + (n-1)\max\{o_s, g\}$. Время приема n сообщений (относительно первого отправленного пакета на стороне отправителя): $o_s + L + o_r + (n-1)\max\{o_s, g\}$.

Используем также обозначения [12]:

$$f_r = \max\{o_r, g\},\ f_s = \max\{o_s, g\};$$

$$t_r = \max\{f_r, o_s + L + o_r\} = \max\{g, o_s + L + o_r\};$$

$$t_s = \max\{g, o_s + L + o_r\};$$

$$f_r = f_s = o;$$

$$t_r = t_s = 2 \times o + L.$$

IV. Адаптивный алгоритм барьерной синхронизации

Создан адаптивный алгоритм AdaptiveBarrierLogP барьерной синхронизации. Входными данными алгоритма являются коммуникатор MPI, параметры модели LogP (L, o_r, o_s, g, P) и набор базовых алгоритмов BarrierAlgs. Параметры коммуникационной среды можно получить от системы мониторинга, например, Netgauge [13].

Предложенный алгоритм включает в себя три основных шага (рис. 2):

1. Вычисление оценки времени реализации барьерной синхронизации каждого из рассматриваемых базовых алгоритмов.

2. Поиск минимального из значений времени и выбор алгоритма на основе результатов поиска.

3. Выполнение барьерной синхронизации на основе выбранного алгоритма.

```
Входные данные:
L, o, g, P – параметры модели LogP
BarrierAlgs – массив алгоритмов длины n.
1  for i = 1 to n do
2     // Сохранить в массиве время выполнения алгоритмов
3     t[i] = CompTime(L, o_r, o_s, P, BarrierAlgs[i])
4  end for
5  // Получить индекс элемента массива
6  // с минимальным значением времени
7  i* = argmin(t[i])
8  // Вызвать функцию, реализующую выбранный алгоритм
9  MPI_Barrier(BarrierAlgs[i*])
```

Рис. 2. Алгоритм AdaptiveBarrierLogP

V. Аналитические модели алгоритмов

В данной работе будем использовать аналитические оценки для распространенных алгоритмов барьерной синхронизации, полученные в статье [12].

A. Central Counter

Алгоритм Central Counter (рис. 3) включает две фазы: в ходе 1 фазы каждый некорневой процесс отправляет сообщение корневому процессу о том, что он достиг барьера. Если процессы отправляют сообщения одновременно, а их латентность равна между собой, то

пакеты сериализуются в получателе, поскольку принятое сообщение «блокирует» сетевой интерфейс на время f_r. На этом шаге корневой процесс является узким местом.

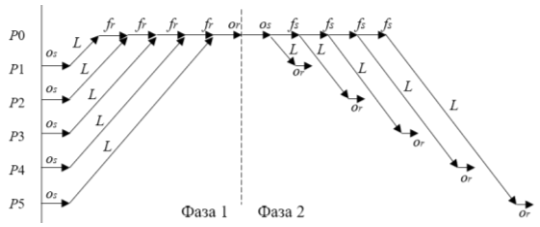

Рис. 3. Алгоритм Central Counter

Вторая фаза начинается после получения корневым процессом всех пакетов и включает отправку сообщений об освобождении (release) всем процессам. В ходе её выполнения также имеет место блокировка корневого процесса, поскольку каждый отправленный пакет «блокирует» отправителя на время f_s.

Время выполнения первой фазы:

$$t' = o_s + L + (P - 2)\, f_r + o_r$$

Вторая фаза начинается с момента t' и продолжается до времени выполнения последнего процесса:

$$t'' = o_s + (P - 2)\, f_s + L + o_r$$

Процесс 0 завершил барьер после:

$$t_0 = t' + o_s + (P - 2)\, f_s = 2o_s + o_r + L + (P - 2)\, f_r + (P - 2)\, f_s$$

Процесс $i \in \{1, \ldots, P\}$ заканчивается после:

$$t_i = t' + o_s + L + (i - 1)\, f_s + o_r =$$

$$= 2(o_s + L + o_r) + (P - 2)\, f_r + (i - 1)\, f_s$$

Последний процесс (P) заканчивается и для всего барьера получаем следующее время реализации [12]:

$$t = 2(o_s + L + o_r) + (P - 2)\, f_r + (P - 2)\, f_s$$

B. Combining Tree

Основная его идея состоит разбиении операции барьерной синхронизации на дерево барьеров меньшего размера и объединении ветви запросов и ветви уведомлений (рис. 4). Алгоритм имеет сложность $O(n\log_n P)$, где n – число потомков каждого узла.

Каждому процессу присваивается уникальный узел дерева, который связан в дерево захвата по родительской ссылке и в дерево освобождения набором дочерних ссылок. Родитель уведомляет каждого из своих потомков, устанавливая флаг в узлах, соответствующих им. Потомок, в свою очередь, устанавливает флаг в родительском узле, сигнализируя о его попадании в барьер.

Время выполнения первой фазы (сбор данных на корневом процессе):

$$t' = (o_s + L + f_r \times (n - 2) + o_r) \times \log_n P$$

Время выполнения второй фазы алгоритма, использующего в своей основе биномиальное дерево:

$$t'' = o_s + (\log_2 P - 1) \max\{g, o_s + L + o_r\} + L + o_r =$$
$$= o_s + (\log_2 P - 1)\, t_s + L + o_r$$

Суммарное время реализации [12]:

$$t = (o_s + L + f_r \times (n - 2) + o_r) \log_n P + o_s + (\log_2 P - 1)\, t_s + L + o_r$$

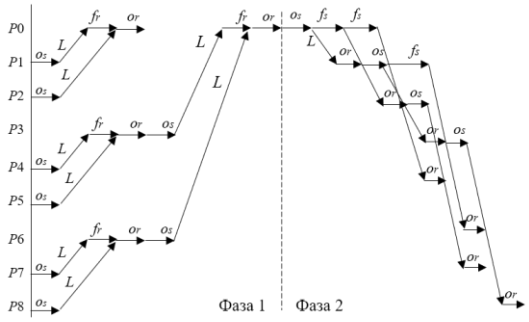

Рис. 4. Алгоритм Combining Tree

C. Dissemination

Алгоритм Dissemination имеет сложность $O(\log_2 P)$ и относится к классу барьеров, которые теоретически быстрее рассматриваемых в статье алгоритмов. В процессе работы каждый процесс выполняет два вызова операции приема-передачи сообщения (рис. 5). Каждый процесс проходит через $\log_2 N$ раундов. В конце раунда процесс имеет информацию о том, что другие процессы достигли барьера и готовы перейти к следующему раунду.

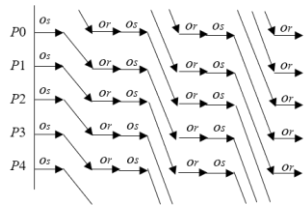

Рис. 5. Алгоритм Dissemination

Время выполнения алгоритма в модели LogP можно оценить следующим образом [12]:

$$t = \max\{f_r, f_s, o_s + L + o_r\} \log_2 P = \max\{t_r, t_s\} \log_2 P$$

VI. Эксперименты

Эксперименты производились на кластерных ВС информационно-вычислительного центра Новосибирского национального исследовательского государственного университета. Использовался вычислительный кластер, укомплектованный 30 узлами с двумя 12-ядерными процессорами Intel Xeon E5-2680v3. Использовались компилятор GCC 4.8.5, библиотека Open MPI 1.10.

Тестовая программа представляет собой циклический запуск функции барьерной синхронизации. Между вызовами барьерной синхронизации установлена задержка 1000 ± 100 мс. Количество итераций цикла – 400. Количество процессов варьировалось от 2 до 100. В качестве показателя эффективности использовалось среднее время t выполнения операции.

По результатам экспериментов (рис. 6а) видно, что Central Counter существенно уступает алгоритмам Dissemination, Combinig Tree и AdaptiveBarrierLogP при увеличении числа процессов. Dissemination и Combinig-Tree демонстрируют близкие результаты, но последний характеризуется большим временем выполнения. Разработанный алгоритм AdaptiveBarrierLogP в большинстве экспериментов показывает результаты, лучшие в среднем на 4%.

Частота использования схем реализации барьера алгоритмом AdaptiveBarrierLogP: Dissemination (96%), Combinig Tree (3%), Central Counter (1%). Для числа процессов меньше 20 (рис. 6б) превалирует выбор Combinig Tree, в остальных случаях более эффективным оказывается выбор Dissemination. На рис. 7 представлено время выполнения операции, полученное на основе используемых в данной работе аналитических формул, для следующих значений параметров модели LogP (L = 125,6, o_r = 123,8, o_s = 0,43, g = 0,22, $P \in \{2, ..., 100\}$).

На основе критерия хи-квадрат построены вероятности соответствия экспериментальных данных: p_1 = 0,9950419 (Combinig Tree), p_2 = 0,999992 (Dissemination) и p_3 = 0,296601 (Central Counter). Данные вероятности близки к 1, что значительно превышает уровень значимости α = 0,05. Поэтому можно утверждать, что экспериментальные данные не противоречат аналитическим.

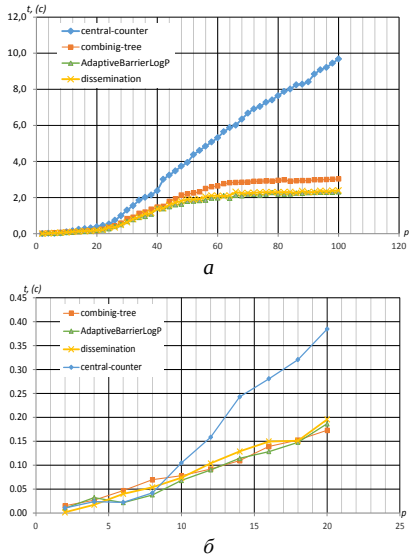

Рис. 6. Время реализации операций барьерной синхронизации

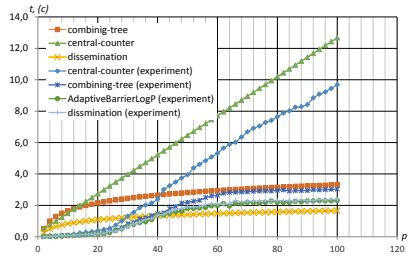

Рис. 7. Сравнение теоритических и экспериментальных результатов

VII. Заключение

Разработан адаптивный алгоритм AdaptiveBarrierLogP субоптимального выбора алгоритмов барьерной синхронизации в стандарте MPI. В процессе выбора учитываются оценки время выполнения информационных обменов в модели параллельных вычислений LogP. Приводятся результаты натурных экспериментов на базе кластерных ВС.

Результаты исследования созданного алгоритма показывают, что среднее время реализации выполнения синхронизации, с учетом параметров модели LogP, уменьшилось в среднем на 4%, по сравнению с использованием детерминированного выбора алгоритма. При числе процессов меньших 20 превалирует выбор алгоритма Combinig Tree, в остальных случаях более эффективным оказывается выбор Dissemination.

Созданный алгоритм реализован программно в стандарте MPI и может быть использован для сокращения времени выполнения существующих MPI-программ.

Список литературы

[1] Хорошевский В. Архитектура вычислительных систем. М.: Изд-во МГТУ им. Н.Э. Баумана, 2008. 520 с.

[2] Paul S., Sandra W. Berkeley‘s Dwarfs on CUDA // RWTH Aachen University, 2011. 27 с.

[3] Scott M., Mellor-Crummey J. Fast contention-free combining tree barriers for shared-memory multiprocessors // International Journal of Parallel Programming. 1994. Vol. 22(4). P. 449-481.

[4] Hensgen D., Finkel R., Manber U. Two algorithms for barrier synchronization // International Journal of Parallel Programming. 1988. Vol. 17(1). P. 1-17.

[5] Hoefler T., Mehlan T., Mietke F., Rehm W. Fast Barrier Synchronization for InfiniBand // IPDPS. 2006. N 2. P. 272-272.

[6] Juan C., Yong D. Energy optimization of representative barrier algorithms // Journal of Central South University. 2012. Vol. 19(2). P. 2823-2831.

[7] Malkis A., Banerjee A. Automation in the Verification of Software Barriers // International Journal of Parallel Programming. 2014. Vol. 52(3). P. 275-329.

[8] Carwyn B., Mark B. Barrier Synchronisation in Java // Technical report, High-End Computing program (UKHEC). 2005. N 1. P. 1-25.

[9] Hoefler T., Mehlan S., Mietke A., Rehm T., Wolfgang S. A Survey of Barrier Algorithms for Coarse Grained Supercomputers // Technical report 3. University of Chemnitz (Germany). 2004. N 1, P. 1-20.

[10] Pješivac-Grbović J., Angskun T., Bosilca G., Graham E., Edgar G., Jack J. Performance analysis of MPI collective operations // Cluster Computing. 2007. Vol. 10(2). P. 127-143.

[11] Kielmann T., Bal H., Verstoep K. Fast measurement of LogP parameters for message passing platforms // IPDPS. 2000. P. 1176-1183.

[12] Hoefler T., R. Wolfgang A practical Approach to the Rating of Barrier Algorithms using the LogP Model and Open MPI // Parallel Processing. 2005. P. 562-569.

[13] Hoefler T. Netgauge: A network performance measurement framework // HPCC. 2007. P. 659-671.