MPIPerf: пакет оценки эффективности коммуникационных функций стандарта MPI*

М.Г. Курносов[1]

Институт физики полупроводников им. А.В. Ржанова СО РАН[1]

Рассмотрены методы измерения времени выполнения коллективных операций обмена информацией (Collective communications) между ветвями параллельных MPI-программ. Отражены систематические ошибки измерений характерные для программных пакетов, реализующих эти методы. Предложен подход к измерению времени выполнения коллективных операций обменов, основанный на синхронизации моментов запуска коммуникационных функций в ветвях MPI-программы. Приведено описание разработанного пакета MPIPerf и результаты натурных экспериментов на вычислительных кластерах с сетями связи Gigabit Ethernet и InfiniBand QDR.

1. Введение

Коммуникационные библиотеки стандарта MPI (MPICH2, Open MPI, Cray MPI и др.) являются основным средством создания параллельных программ для распределенных вычислительных систем (ВС). Основу этих библиотек составляют коммуникационные функции дифференцированных (Point-to-point communications) и коллективных операций (Collective communications) обмена информацией между ветвями параллельных программ. Коллективные операции подразделяются на несколько видов [1, 2]: трансляционный (ТО, One-to-all broadcast), трансляционно-циклический (ТЦО, All-to-all broadcast), коллекторный обмены (КО, All-to-one broadcast) и коллективные операции синхронизации ветвей (Barrier, Eureka). Коллективные операции реализуется на основе дифференцированных обменов (Point-to-point communications) или аппаратурно с использованием специализированных коммуникационных сетей (например, на базе сети с древовидной топологией в системах семейства IBM Blue Gene). Для широкого класса параллельных алгоритмов время выполнения коллективных операций является критически важным и определяет их масштабируемость [3, 4].

Коллективные операции обменов информацией реализуются программно в виде коммуникационных функций. Например, операции ТО и КО в библиотеках стандарта MPI реализуются функциями: `MPI_Bcast`, `MPI_Gather`; в библиотеках Cray Shmem: `shmem_broadcast`, `shmem_collect`; в языке параллельного программирования Unified Parallel C: `upc_all_broadcast`, `upc_all_gather`.

Пользователями и разработчиками системного программного обеспечения распределенных ВС востребованы средства для оценки времени выполнения функций, реализующих коллективные операции обменов. Такая информация необходима для определения оптимального алгоритма реализации коллективной операции – алгоритма, обеспечивающего минимум времени её выполнения, для определения эффективности реализаций коммуникационных библиотек и runtime-систем языков параллельного программирования, а также для верификации аналитических (LogP, PLogP, LogGP) и имитационных моделей (LogGOPSim, PSINS, DIMEMAS, BigSim) выполнения параллельных программ [5].

В данной работе приводится обзор известных методов измерения времени выполнения коллективных операций обмена информацией. Отражены систематические ошибки измерений характерные для программных пакетов реализующих эти методы. Предложен подход к измерению времени выполнения коллективных операций, в котором учтены известные систематические ошибки измерений, характерные для существующих пакетов.

* Работа выполнена при поддержке РФФИ (грант № 11-07-00105), Совета по грантам Президента РФ для поддержки ведущих научных школ (грант НШ-2175.2012.9) и в рамках госконтракта № 07.514.11.4015 с Минобрнауки РФ.

2. Методы измерения времени выполнения коллективных операций обмена информацией

Сформулируем суть задачи измерения времени выполнения функции, реализующей коллективную операцию обмена информацией между ветвями параллельной программы. Задана функция и значения ее входных параметров (буферы приема-передачи, размеры сообщений и номера ветвей, участвующих в операции). Требуется измерить время $t_0, t_1, \ldots, t_{n-1}$ выполнения функции в n ветвях программы на заданной подсистеме ЭМ.

В основе большинства известных методов (пакеты Intel MPI Benchmarks [6], mpptest [7], LLCBench [8], Phloem [9], MPIBlib [10], OSU Micro-Benchmarks [11] и др.) время выполнения коллективной операции оценивается путем измерения времени k выполнений функции (рис. 1, здесь и далее примеры приводятся для функций стандарта MPI). За время t выполнения функции в ветви принимается среднее время одного её запуска (среднее время одной итерации цикла измерений).

```
MPI_Bcast(buf, count, MPI_BYTE, root, comm)              /* Инициализация */
MPI_Barrier(comm)                                        /* Синхронизация */
t = MPI_Wtime()
for i = 1 to k do                                       /* Цикл измерений */
    MPI_Bcast(buf, count, MPI_BYTE, root, comm)
end for
t = (MPI_Wtime() – t) / k                 /* Среднее время одного запуска */
```

Рис. 1. Процедура измерения времени выполнения функции `MPI_Bcast`

Время выполнения коллективной операции зависит от значений её входных параметров. Так, на время выполнения функции трансляционного обмена `MPI_Bcast` влияют способ выделения памяти для буфера приема-передачи `buf` (выбор границы выравнивания адреса буфера, размещение передаваемой информации в кэш-памяти процессора), тип передаваемых данных (встроенный тип данных MPI с непрерывным размещением элементов в памяти или производный тип данных с размещением элементов в памяти по несмежным адресам), выбор корневого процесса `root` и коммуникатора `comm`, который задает распределение ветвей программы по ЭМ системы. Кроме того, на время выполнения функции оказывают влияние и ветви других программ, которые загружают совместно используемые ресурсы системы (коммуникационную сеть, контроллер(ы) оперативной памяти вычислительных узлов и др.). По этой причине результаты измерений времени выполнения коллективных операций известными пакетами различны. Кроме этого, существующим методам присущи систематические ошибки измерений.

3. Систематические ошибки измерения времени выполнения коллективных операций обмена информацией

Рассмотрим распространенные систематические ошибки измерений времени выполнения коллективных операций обмена информацией, характерные для существующих методов и программных пакетов, реализующих их.

3.1 Игнорирование отложенной инициализации MPI-функций

Время выполнения коллективной операции измеряется без её предварительной инициализации (пакеты Intel MPI Benchmarks, Phloem, MPIBlib, SKaMPI [12]). Во многих библиотеках стандарта MPI и системах параллельного программирования ветви программы, участвующие в коллективной операции, устанавливают между собой сетевое соединение только при её первом вызове (Late initialization). Поэтому время выполнения первого вызова может значительно превосходить время выполнения последующих обращений к функции. Результаты первого вызова не должны учитываться в итоговой оценке времени выполнения операции. В табл. 1 приведены отношения времени выполнения ветвью 0 первого вызова MPI-функции ко второму вызову.

Результаты получены на вычислительном кластере с коммуникационной сетью InfiniBand QDR (64 процесса – 8 узлов по 8 ядер, при повторном вызове функции использовался буфер такого же размера – 8192 байт, но размещенный по другому адресу в оперативной памяти).

Таблица 1. Отложенная инициализация MPI-функций

MPI-функция	Отношение времени выполнения ветвью 0 первого вызова MPI-функции к её второму вызову	
	Библиотека OpenMPI 1.4.4	Библиотека Intel MPI 4.0.0.028
`MPI_Bcast`	6178	2,59
`MPI_Allgather`	5,61	131,8
`MPI_Barrier`	5,50	9,01
`MPI_Reduce`	6944	12901
`MPI_Allreduce`	11,71	353,37

3.2 Учёт времени выполнения коллективной операции только в одной ветви

В качестве конечного значения времени выполнения коллективной операции обмена информацией принимается только время определенной ветви (например, ветви 0). Однако, время выполнения операции в ветвях различно. Это обусловлено тем, что алгоритмы коллективных операций реализуются на основе дифференцированных обменов. В зависимости от используемого алгоритма коллективной операции ветви выполняют различное количество таких обменов [2, 13, 14]. Например, на рис. 2 приведена диаграмма выполнения коллекторного приема информации (`MPI_Reduce`) алгоритмом биномиального дерева (Binomial tree) [14].

За время выполнения коллективной операции обмена информацией следует принимать максимальное из времен выполнения операции в ветвях программы.

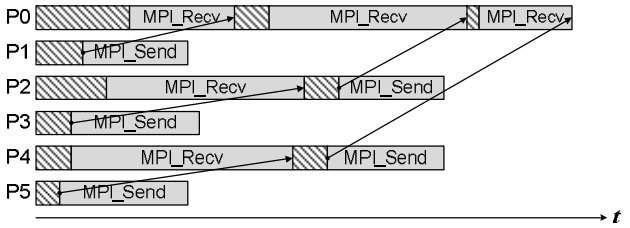

Рис. 2. Диаграмма выполнения коллекторного приема информации (`MPI_Reduce`) ветвью 0 (6 ветвей, результаты получены пакетом Intel Trace Analyzer and Collector)

3.3 Измерение времени выполнения MPI-функции при разных значениях входных параметров

На каждой итерации цикла измерений (рис. 1) используются различные значения входных параметров. Например, в пакетах Intel MPI Benchmarks, mpptest при измерении времени выполнения функции `MPI_Bcast` на каждой итерации цикла номер `root` корневой ветви изменяется. Это приводит к тому, что при выполнении коллективной операции используются различные каналы связи (меняется последовательность обменов сообщениями между ветвями).

3.4 Игнорирование иерархической организации подсистемы памяти вычислительных узлов

В некоторых пакетах не учитывается то, что при повторном использовании одного и того же буфера приема/передачи его данные с большой вероятностью будут размещены в кэш-памяти процессора. Это приводит к тому, что время выполнения первого и последующих вызовов коллективной операции могут значительно отличаться.

3.5 Некорректная синхронизация ветвей программы перед выполнением коллективной операции

Синхронизация ветвей параллельной программы перед и/или в цикле измерений выполняется при помощи барьерной синхронизации (`MPI_Barrier`). Это приводит к неравномерному смещению моментов запуска коллективной операции в ветвях. Причина в том, что барьерная синхронизация реализуется путем дифференцированных обменов сообщениями нулевой длины [13, 14]. Количество таких обменов, выполняемых каждой ветвью, различно и зависит от используемого алгоритма барьерной синхронизации. В конечном счете, ветви осуществляют выход из функции синхронизации в разные моменты времени. На рис. 3 приведен пример выполнения барьерной синхронизации "рассеивающим" алгоритмом (Dissemination barrier) [15]. Каждая из 8 ветвей выполняет два вызова операции приёма-передачи (`MPI_Sendrecv`). Видно, что ветви заканчивают выполнение операции в различные моменты времени.

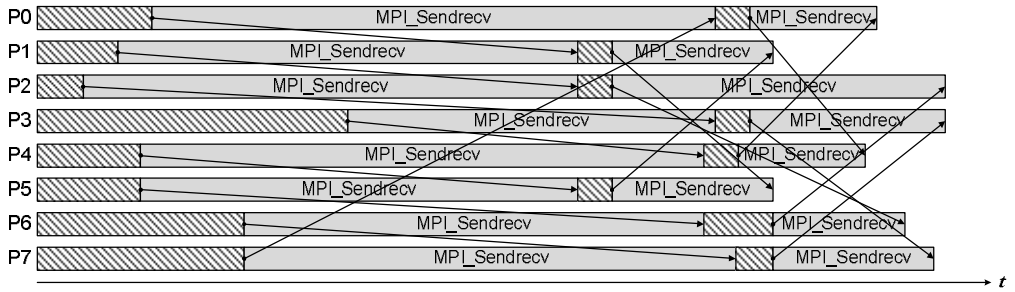

Рис. 3. Диаграмма выполнения "рассеивающего" алгоритма барьерной синхронизации между 8 параллельными ветвями (результаты получены пакетом Intel Trace Analyzer and Collector)

3.6 Измерение времени выполнения коллективной операции без синхронизации моментов её запуска в ветвях

Измерение времени выполнения MPI-функции выполняется в цикле без синхронизации моментов её запуска в ветвях программы (рис. 1). Это приводит к тому, что результирующая оценка времени выполнения операции включает и время ожидания начала следующей итерации цикла измерений. Рассмотрим измерение времени выполнения функции `MPI_Bcast`, в соответствие с процедурой на рис. 1. Для определенности будем полагать, что `MPI_Bcast` реализуется линейным алгоритмом, при котором сообщение размером m байт из ветви 0 передается в ветвь 1, затем в ветвь 2 и т.д. На рис. 4 показана диаграмма выполнения этого алгоритма. Время t_i выполнения обменов ветвью $i \in \{0,1,\ldots,n-1\}$ выражено в модели Дж. Хокни, в которой α – латентность канала связи, β – время передачи одного байта по каналу связи.

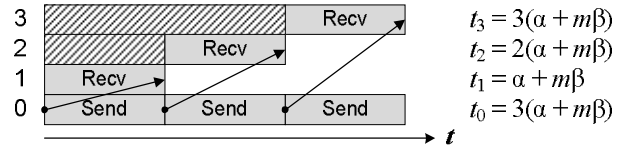

Рис. 4. Диаграмма выполнения линейного алгоритма трансляционного обмена (`MPI_Bcast`) между 4 ветвями параллельной программы:
– обмен информацией; – ожидание

На рис. 5 приведена диаграмма выполнения четырех итераций цикла измерений (рис. 1). Из диаграммы видно, что измеренное время T_1 выполнения функции в ветви 1 в 2,5 раза превосходит "истинное" время t_1 (рис. 4). Это объясняется тем, что после выполнения первой итерации, ветвь 1 сразу (без синхронизации) переходит к итерации 2 и запускает операцию, в которой она ожидает сообщение от ветви 0. В состоянии ожидания ветвь 1 прибывает до тех пор, пока ветвь 0 не передаст сообщения ветвям 2 и 3. Таким образом, на каждой итерации к истинному време-

ни выполнения операции добавляется время ожидания. Измеренное таким способом время выполнения функции включает систематическую ошибку, связанную с несинхронизированным запуском операций в ветвях программы. Сказанное справедливо не только для измерения времени выполнения алгоритмов ТО, но и для других коллективных операций.

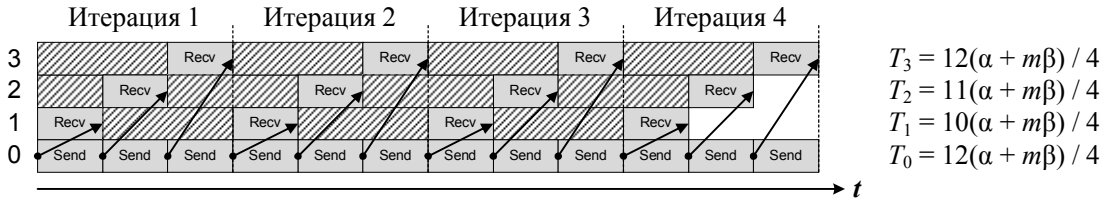

Рис. 5. Диаграмма выполнения четырех итераций цикла измерений времени выполнения линейного алгоритма трансляционного обмена

Обозначим через $t_i(n)$ время выполнения линейного алгоритма ТО в ветви $i \in \{0,1,\ldots,n-1\}$. Аналогично, $T_i(n,k)$ – время выполнения линейного алгоритма ТО в ветви i, измеренное k итерациями цикла (рис. 1). Нетрудно показать, что

$$t_i(n) = \begin{cases} (n-1)(\alpha + m\beta), & \text{при } i = 0, \\ i(\alpha + m\beta), & \text{при } i > 0, \end{cases} \qquad T_i(n,k) = \begin{cases} (n-1)k(\alpha + m\beta) / k, & \text{при } i = 0, \\ ((n-1)(k-1) + i)(\alpha + m\beta) / k, & \text{при } i > 0. \end{cases}$$

Тогда отношение $r_i(n,k)$ времени $T_i(n,k)$ к $t_i(n)$ позволяет судить об ошибке метода, приведенного на рис. 1:

$$r_i(n,k) = \frac{T_i(n,k)}{t_i(n)} = \begin{cases} 1, & \text{при } i = 0, \\ \dfrac{(n-1)(k-1)+i}{ik} & \text{при } i > 0. \end{cases}$$

Аналогичные оценки систематической ошибки метода можно построить, используя более точные модели дифференцированных обменов: LogGP [16], PLogP [17].

Как было отмечено выше, причина возникновения рассмотренной ошибки заключается в несинхронизированном запуске коллективной операций в цикле измерений (рис. 1). Одним из известных подходов к устранению этой ошибки является организация одновременного запуска операций в ветвях. Это достигается за счет синхронизации показаний локальных часов ветвей и формирования расписаний запуска операции в них. Такой подход лежит в основе пакетов SKaMPI, MPIBench [18] и Netgauge [19]. Однако эти пакеты не лишены перечисленных выше систематических ошибок измерений.

Автором предложен метод измерения времени выполнения коллективных операций обменов информацией, основанный на синхронизации моментов запуска функций в ветвях. В методе учтены известные систематические ошибки измерений времени выполнения таких операций.

4. Описание метода

Разработанный метод включает нижеследующие шаги.

1. *Синхронизация показаний локальных часов ветвей.* Каждая ветвь i вычисляет смещение o_i показаний своих локальных часов относительно часов ветви 0, показания которых принимаются за глобальное время. Зная свое локальное время T_i, ветвь i может вычислить показания глобальных часов $T_0 = T_i + o_i$ и наоборот: $T_i = T_0 - o_i$.

2. *Оценка времени выполнения ТО.* Формируется оценка сверху времени t_{bcast} выполнения трансляционной передачи (`MPI_Bcast`) из ветви 0 сообщения с показанием её часов. Как правило, показания часов представляется вещественным числом двойной точности в формате IEEE 754.

3. *Измерение времени выполнения коллективной операции.* Процесс измерения времени выполнения коллективной операции разбивается на этапы (рис. 6).

3.1. На каждом этапе $j = 0,1,\ldots$ ветвь 0 запрашивает показание T_0 своих локальных часов и используя трансляционный обмен (`MPI_Bcast`) передает ветвям время τ_j первого запуска операции: $\tau_j = T_0 + t_{bcast}$.

3.2. По значению τ_j ветви формируют расписание k запусков операции на этапе j. Запуск $l = 0,1,\ldots,k-1$ осуществляется по глобальным часам в момент времени $\tau_{jl} = \tau_j + l\delta_j$. На каждый запуск отводится δ_j секунд, таким образом, запуск l должен завершиться до момента $\tau_{j,l+1} = \tau_j + (l+1)\delta_j$. Если ветвь осуществляет запуск l позднее момента τ_{jl} или завершает выполнение операции после $\tau_j + (l+1)\delta_j$, то результат выполнения операции считается некорректным. Такие ситуации могут возникать по причине динамического характера загрузки ресурсов распределенных ВС.

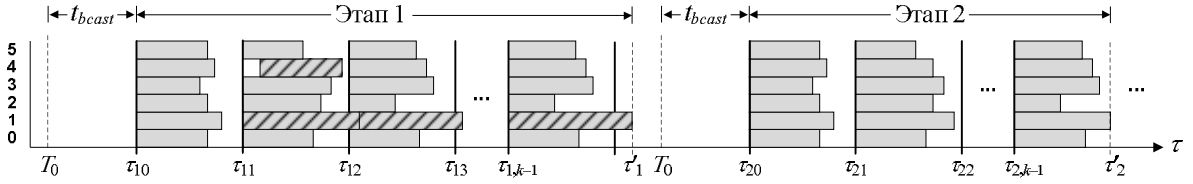

Рис. 6. Этапы измерения времени выполнения коллективной операции обмена информацией:
▨ – некорректное выполнение; ▭ – корректное выполнение

3.3. После окончания этапа осуществляется анализ результатов измерений. За время выполнения операции на запуске l принимается максимальное из времен выполнения ветвей. Если на этапе j количество некорректных запусков превышает g процентов, то выполняется корректировка длины интервала $\delta_{j+1} = \gamma(\tau'_j - \tau_j)/k$, где τ'_j – время завершения этапа j (максимальное из времени завершения выполнения ветвей на k-ом запуске этапа j), γ – масштабный коэффициент ($1,1 < \gamma < 2$, подбирается эмпирически).

3.4. Проверяются условия окончания измерений. Если стандартная ошибка среднего времени выполнения коллективной операции достигла заданного уровня или количество запусков превысило максимально допустимое, то измерения прекращаются.

Инициализация операции осуществляется на этапе 0. Выполняется k' её запусков при длине интервала $\delta_0 = 0$. Результаты этого этапа не учитываются при формировании итоговых результатов измерений. Время выполнения этапа 0 используется для формирования начального значения длины интервала $\delta_1 = \gamma(\tau'_0 - \tau_0)/k'$.

4. *Статистическая обработка результатов измерений.* После завершения измерений осуществляется обработка их результатов. Пусть Q количество корректных запусков операции за время выполнения всех этапов, а t^q – время выполнения операции на запуске $q \in \{1,2,\ldots,Q\}$. Из последовательности значений $t^1, t^2, \ldots, t^Q$ удаляется g' процентов минимальных и максимальных значений. За итоговое время t выполнения коллективной операции принимается статистическая оценка математического ожидания времени выполнения корректных запусков операции. Для измеренного времени t формируется доверительный интервал с заданной доверительной вероятностью.

5. Пакет MPIPerf

Описанный метод реализован автором в пакете MPIPerf. В текущей версии реализованы тесты для всех коллективных операций стандарта MPI 2.2.

Синхронизация локальных часов ветвей в пакете MPIPerf осуществляется по выбору пользователя линейным или кольцевым алгоритмом. Оба алгоритма основаны на процедуре син-

хронизации часов двух ветвей – корневой ветви 0 и ветви i. Смещение o_i показаний своих локальных часов ветвь i рассчитывается следующим образом (рис. 7)

$$o_i = T_0 - \frac{T_{\text{RTT}}}{2} - T_i', \quad T_{\text{RTT}} = T_i'' - T_i',$$

где T_0 – показание часов ветви 0, а T_i' и T_i'' показания часов ветви i. Величина T_{RTT} – это суммарное время передачи и последующего приема сообщения с показаниями часов (RTT – Round Trip Time). За значение T_{RTT} принимается минимальное из измеренных. Причем измерения величины T_{RTT} заканчиваются, если её текущее минимальное значение не изменялось на протяжении последних d итераций (по умолчанию $d = 100$).

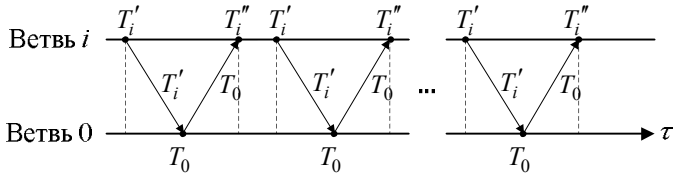

Рис. 7. Синхронизация локальных часов ветвей i и 0

Линейный алгоритм последовательно выполняет синхронизацию часов ветвей $1, 2, \ldots, n-1$ с часами ветви 0. Коммуникационная сложность (асимптотическая оценка количества выполняемых дифференцированных обменов) линейного алгоритма составляет $T_{linear} = O(n \cdot T_{sync})$, где T_{sync} – коммуникационная сложность процедуры синхронизации двух ветвей (рис. 7).

В кольцевом алгоритме каждая ветвь i аналогично рассчитывает отклонение $o_{i,(i-1+n) \bmod n}$ показаний своих часов от ветви $(i-1+n) \bmod n$. Далее ветвь 0, используя функцию трансляционного обмена (`MPI_Gather`), принимает смещения времени o_{ij} всех ветвей. Итоговое смещение локальных часов ветви i рассчитывается ветвью 0 как $o_i = o_{10} + o_{21} + \ldots + o_{i,(i-1+n) \bmod n}$. Затем значения o_i рассылаются ветвям функцией трансляционного обмена (`MPI_Scatter`). Коммуникационная сложность кольцевого алгоритма складывается из сложности T_{sync} и сложности алгоритмов коллекторного и трансляционного обменов. Как правило, в современных реализациях библиотек стандарта MPI функции `MPI_Scatter` и `MPI_Gather` реализуются алгоритмами с логарифмической коммуникационной сложностью, в этом случае сложность кольцевого алгоритма $T_{ring} = O(T_{sync} + \log_2 n)$.

Ниже приведены значения параметров предложенного метода, используемые по умолчанию в пакете MPIPerf (пользователю доступна регулировка этих значений):

- количество k' запусков операции в процессе её инициализации: 4;
- количество k запусков операции на каждом этапе измерений: 8;
- количество некорректных запусков, по достижению которого корректируется длина интервала δ_j: 25% от общего количества запусков на этапе j;
- значение масштабного коэффициента γ для корректировки длины интервала δ_j: 1,1;
- условия окончания измерений: вариант 1 – количество измерений больше 100 или корректных измерений больше 30; вариант 2 – относительная ошибка среднего значения времени выполнения операции не превосходит 0,05 и количество успешных измерений не менее 10;
- количество g' отбрасываемых минимальных и максимальных значений измеренного времени выполнения операции: 25% от общего числа корректных измерений.

По окончанию работы пакет MPIPerf формирует отчет включающий: значение варьируемого параметра (размер сообщения или количество ветвей в программе); общее количество n_t запусков коллективной операции; количество n_c корректных запусков; количество n_s корректных запусков после статистической обработки; статистическая оценка T математического

ожидания времени выполнения операции (формируется по n_s значениям); стандартная ошибка SE измерения времени T ($SE = \sigma[T] / \sqrt{n_c}$, где $\sigma[T]$ – несмещённая оценка среднеквадратичного отклонения T относительно его математического ожидания); минимальное значение T, максимальное значение T, абсолютная ошибка E измерений времени T ($E = \alpha \cdot SE$, где α – коэффициент Стьюдента), доверительный интервал $P(T - \alpha \cdot SE \le T \le T + \alpha \cdot SE) = p$, где p – заданная пользователем надежность ($p \in \{0,9; 9,95; 0,99\}$).

Реализована возможность формирования отчетов о времени выполнения операции в каждой ветви программы.

6. Экспериментальное исследование

Эксперименты с пакетом MPIPerf проводились на двух вычислительных кластерах:

– кластер А (Центр параллельных вычислительных технологий ФГОБУ ВПО "Сибирский государственный университет телекоммуникаций и информатики"): 10 вычислительных узлов, на каждом узле установлено 2 четырехъядерных процессора Intel Xeon E5420, коммуникационная сеть – Gigabit Ethernet, операционная система – CentOS 5.2 x86_64 (ядро linux 2.6.18-92.el5);

– кластер Б (Информационно-вычислительный центр ФГОБУ ВПО "Новосибирский национальный исследовательский государственный университет"): использована подсистема из 96 узлов HP BL2x220 G7, на каждом узле 2 шестиядерных процессора Intel Xeon X5670, коммуникационная сеть – InfiniBand 4x QDR, операционная система – SUSE Linux Enterprise Server 11 x86_64 (ядро linux 2.6.27.19-5).

6.1. Исследование точности синхронизации моментов запуска коллективной операции в ветвях программы

Точность синхронизации моментов запуска операции в ветвях можно оценить при помощи специальных коллективных функций ожидания, время выполнения которых известно априори. В первой функции `WaitPatternUp` [12] каждая ветвь i ожидает $(i+1) \cdot 10^{-6}$ секунд (выполняет пустой цикл). Если все ветви запустили эту функцию в один момент времени, то время её выполнения составит $n \cdot 10^{-6}$ секунд, где n – количество ветвей в программе. Во второй функции `WaitPatternNull` каждая ветвь засекает время запуска операции и сразу завершает её выполнение. Время работы этой функции 0 секунд.

На рис. 8 показано время выполнения теста `WaitPatternUp` на вычислительном кластере А с использованием библиотеки MPICH2 1.2.1. Видно, что время операции, измеренное пакетом MPIPerf, совпадает с её действительным временем выполнения. На рис. 9 показано время выполнения того же теста на кластере Б (использована библиотека OpenMPI 1.4.4) пакетами MPIPerf (кривая 1) и SKaMPI (кривая 2). Результаты не совпадают с реальным временем выполнения операции. Причина в том, что на узлах кластера Б операционная система в качестве источника времени (clock source) использует высокоточный таймер событий HPET – High Precision Event Timer, с которым она некорректно работает [20] и формирует неточные значения функции `gettimeofday`. В свою очередь, эта функция используется во многих библиотеках MPI для реализации подпрограммы `MPI_Wtime`, средствами которой осуществляются измерения времени в большинстве известных пакетов (Intel MPI Benchmarks, mpptest, Phloem, MPIBlib, OSU Micro-Benchmarks). Для решения этой проблемы в MPIPerf реализована возможность выбора таймера, который будет использован для измерения времени выполнения операций и синхронизации ветвей. Поддерживаются таймеры на основе функций `gettimeofday`, `MPI_Wtime` и значений регистра TSC – Time Stamp Counter [21] современных процессоров. На рис. 9 (случай 3) видно, что пакет MPIPerf с таймером TSC корректно измеряет время выполнения операции.

Перед использованием пакета MPIPerf рекомендуется проверить корректность показаний системных таймеров, путем измерения ими времени выполнения операции `WaitPatternNull`. На рис. 10 приведены результаты выполнения теста на кластере Б.

По виду кривой 1 можно судить о некорректной работе функции `MPI_Wtime`, в тоже время кривая 2 свидетельствует о корректной работе таймера на основе значения регистра TSC.

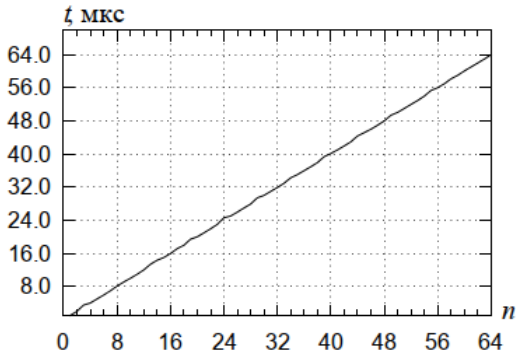

Рис. 8. Зависимость времени t выполнения на кластере А теста `WaitPatternUp` от количества n ветвей в программе (таймер `MPI_Wtime`)

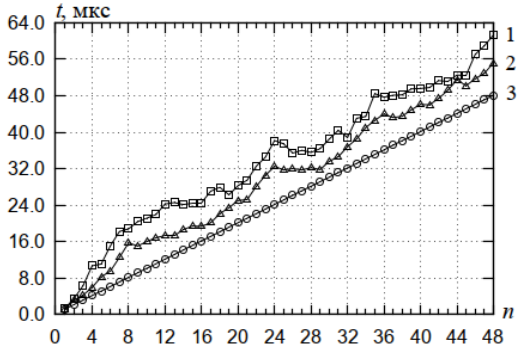

Рис. 9. Зависимость времени t выполнения на кластере Б теста `WaitPatternUp` от количества n ветвей в программе: *1* – MPIPerf, таймер `MPI_Wtime`; *2* – пакет SKaMPI, таймер `MPI_Wtime`; *3* – пакет MPIPerf, таймер TSC

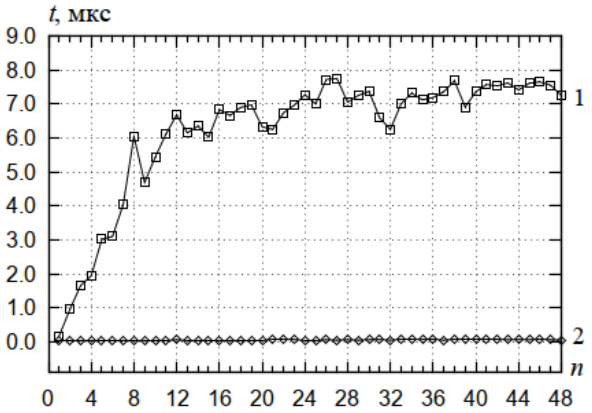

Рис. 10. Зависимость времени t выполнения на кластере Б теста `WaitPatternNull` от количества n ветвей в программе: *1* – MPIPerf, таймер MPI_Wtime; *2* – MPIPerf, таймер TSC

6.2. Сравнение с пакетом SKaMPI

На рис. 11 приведены результаты измерения времени выполнения на кластере А функции `MPI_Bcast` (библиотека MPICH2 1.2.1) пакетами MPIPerf и SKaMPI. Результаты измерений пакетами различны. Это объясняется разными подходами к выделению памяти под буферы приёма и передачи сообщений, а также динамическим характером загрузки ресурсов кластера.

Важным вопросом при разработке средств измерения времени выполнения коллективных операций является воспроизводимость результатов. На рис. 12 показана относительная ошибка измерения среднего времени выполнения функции `MPI_Barrier` пакетами SKaMPI и MPIPerf (использован кластер Б и библиотека Intel MPI 4.0.0.028 с таймером `MPI_Wtime` на основе значения регистра TSC). Оба пакета запускались по 10 раз. На каждом запуске измерялось время выполнения функции `MPI_Barrier` при различном количестве n ветвей в программе. По результатам 10 запусков для каждого значения n рассчитывалось среднее значение $\mathrm{M}[t]$ времени t выполнения функции, среднее квадратическое отклонение $\sigma[t]$ времени выполнения и относительная ошибка RSE измерения среднего времени выполнения функции

$$\mathrm{RSE} = \frac{\sigma[t]}{\mathrm{M}[t] \cdot \sqrt{10}}.$$

Кривые 1 и 2 на рис. 12 свидетельствуют об удовлетворительной для практики воспроизводимости результатов измерений пакетом MPIPerf.

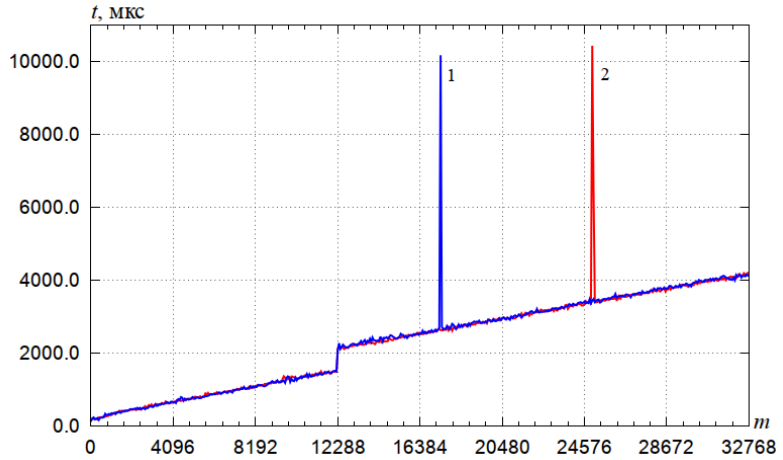

Рис. 11. Зависимость времени t выполнения функции `MPI_Bcast` от размера m передаваемого сообщения: *1* – пакет SKaMPI; *2* – пакет MPIPerf

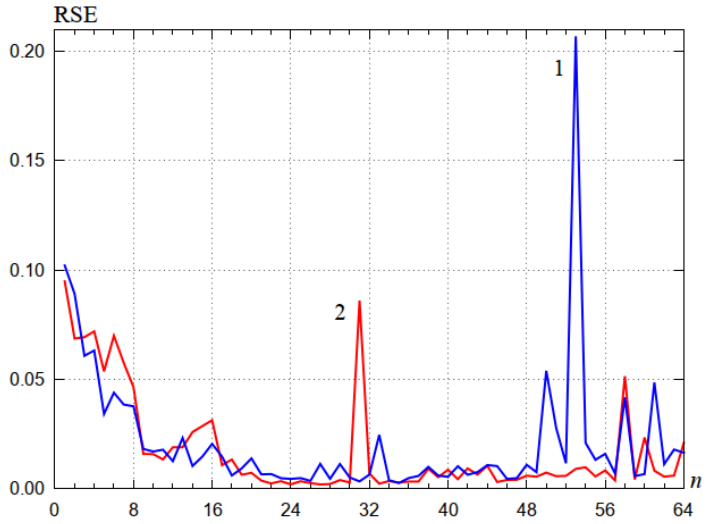

Рис. 12. Зависимость относительной ошибки RSE измерения среднего времени выполнения функции `MPI_Barrier` от количества n ветвей в программе: 1 – пакет SKaMPI; 2 – MPIPerf

Отличительными особенностями созданного пакета MPIPerf являются нижеследующие.

1. Реализованная методика измерения времени выполнения коллективных операций на основе синхронизации моментов их запуска по показаниям глобальных часов; учет отложенной инициализации MPI-функций и иерархической организации подсистемы памяти вычислительных узлов ВС.

2. Расширяемая архитектура пакета, обеспечивающая возможность создания тестов для производных типов данных (MPI Derived data types) и виртуальных топологий (MPI Virtual topologies).

3. Набор тестов для всех коллективных операций стандарта MPI 2.2.

4. Возможность выбора и оценки корректности показаний таймера для измерения времени выполнения коллективных операций.

Пакет распространяется в исходных кодах на условиях лицензии BSD (`http://mpiperf.cpct.sibsutis.ru`).

7. Заключение

Предложенный метод измерения времени выполнения коммуникационных функций основан на синхронизации моментов запуска операции в ветвях MPI-программы и учитывает известные систематические ошибки измерений, характерные для существующих пакетов (OSU MPI Benchmarks, SKaMPI, Netgauge, Intel MPI Benchmarks, Phloem MPI Benchmarks, MPIBLib). Разработанный подход применим и для измерения времени выполнения коммуникационных функций в runtime-системах языков параллельного программирования (IBM X10, Cray Chapel, Unified Parallel C), а также для оценки производительности коммуникационных сетей при реализации коллективных операций обмена информацией.

Программная реализация метода в пакете MPIPerf обеспечивает возможность измерения времени выполнения всех коллективных операций стандарта MPI 2.2. Результаты натурных экспериментов на вычислительных кластерах с сетями связи InfiniBand QDR и Gigabit Ethernet подтвердили эффективность предложенного метода.

В будущих версиях MPIPerf планируется реализовать измерение времени выполнения неблокирующих коллективных операций (Nonblocking collective communications), появление которых ожидается в стандарте MPI 3.0, а также реализовать модуль оценки потребления оперативной памяти библиотеками MPI для исследования их масштабируемости.

Литература

1. Хорошевский В.Г. Распределенные вычислительные системы с программируемой структурой // Вестник СибГУТИ. - 2010. - № 2 (10). - С. 3-41.
2. Курносов М.Г. Алгоритмы трансляционно-циклических информационных обменов в иерархических распределенных вычислительных системах // Вестник компьютерных и информационных технологий. - 2011. - № 5. - С. 27-34.
3. Rabenseifner R.. Automatic MPI Counter Profiling // Proceedings of the 42nd Cray User Group. - Noorwijk, The Netherlands, 2000. - 19 pp.
4. Han D., Jones T.. MPI Profiling // Technical Report UCRL-MI-209658 - Lawrence Livermore National Laboratory, USA, 2004. - 15 pp.
5. Иванников В.П., Аветисян А.И., Гайсарян С.С., Падарян В.А. Прогнозирование производительности MPI-программ на основе моделей // Автоматика и телемеханика. - 2007. - №5. - С. 8-17.
6. Intel MPI Benchmarks 3.2.2 // Intel Software Network. URL: http://software.intel.com/en-us/articles/intel-mpi-benchmarks.
7. Gropp W., Lusk E.L. Reproducible Measurements of MPI Performance Characteristics // In Proc. of the 6th European PVM/MPI Users Group Meeting, 1999. - P. 11-18.
8. LLCbench - Low Level Architectural Characterization Benchmark Suite // LLCBench Home Page. URL: http://icl.cs.utk.edu/projects/llcbench.
9. Phloem MPI Benchmarks // The ASCI Purple Benchmark Codes. URL: https://asc.llnl.gov/sequoia/benchmarks/#phloem.
10. Lastovetsky A., Rychkov V., O'Flynn M. MPIBlib: Benchmarking MPI Communications for Parallel Computing on Homogeneous and Heterogeneous Clusters // In Proc. Of 15th European PVM/MPI User's Group Meeting, Berlin, 2008. - P. 227-238.
11. OSU Micro-Benchmarks // OSU Network-Based Computing Laboratory. URL:http://mvapich.cse.ohio-state.edu/benchmarks/.
12. Worsch T., Reussner R., Werner A. On Benchmarking Collective MPI Operations // Proceedings of the 9th EuroPVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. - 2002. - P. 271-279.

13. Pjesivac-Grbovic J., Angskun T., Bosilca G., Fagg G. Gabriel E. and Dongarra J. Performance Analysis of MPI Collective Operations // Cluster Computing. - 2007. -Vol. 10, No. 2. - P. 127-143.

14. Thakur R., Rabenseifner R., and Gropp W. Optimization of collective communication operations in MPICH // Int. Journal of High Performance Computing Applications. - 2005. - Vol. 19, No. 1. - P. 49 66.

15. Hensgen D., Finkel R. and Manbet U. Two Algorithms for Barrier Synchronization // International Journal of Parallel Programming. - 1988. - Vol. 17(1). - P. 1-17.

16. Alexandrov A., Mihai I., Schauser K. and Scheiman C. LogGP: Incorporating Long Messages into the LogP Model // Technical Report, University of California at Santa Barbara. - 1995. - 21 p.

17. Kielmann T., Bal H. E., Kees V. Fast Measurement of LogP Parameters for Message Passing Platforms // Proceedings of the 15 Workshops on Parallel and Distributed Processing. - London, UK, 2000. - P. 1176-1183.

18. MPIBench // MPIBench Home Page. URL: http://www.dhpc.adelaide.edu.au/projects/mpibench.

19. Hoefler T., Mehlan T., Lumsdaine A. and Rehm W. Netgauge: A Network Performance Measurement Framework // Proceedings of High Performance Computing and Communications, 2007. - P. 659-671.

20. Bug 444496 "hpet increasing min_delta_ns" // Novell Bugzilla. - URL: https://bugzilla.novell.com/show_bug.cgi?id=444496.

21. Intel's Applications Notes. Using the RDTSC Instruction for Performance Monitoring // URL: http://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf.