Исследование влияния параметров функций параллельного ввода-вывода MPI на скорость файловых обменов в суперкомпьютерах*

А.В. Путилин, А.Н. Сальников

факультет Вычислительной Математики и Кибернетики Московского Государственного Университета

Многие научные приложения широко используют параллельный ввод-вывод, например через использование библиотеки Parallel-NetCDF. Часто для научных приложений (например сборка генома с использованием, mpiBLAST) ввод-вывод становится узким местом, существенно влияющим на время исполнения программы. В настоящее время практически все суперкомпьютеры используют специально выделенную подсистему для хранения данных. В данной работе авторы ищут тонкости, связанные с использованием файлового хранилища через комунникационную среду суперкомпьютера. Исследование ведется путем создания специальной системы тестов на основе MPI-IO. С помощью разработанной системы тестов был исследован вычислительный кластер СКИФ-МГУ «Чебышев».

1. Введение

Для любой параллельной программы важны следующие три параметра: скорость выполнения последовательного кода, объем коммуникаций и объем ввода-вывода. Многие научные библиотеки, например Parallel-NetCDF [1] широко используют параллельный ввод-вывод, для некоторых научных приложений (например, mpiblast) ввод-вывод становится узким местом [2].

При наличии данных о производительности отдельных узлов можно предсказать время работы линейных участков кода последовательной программы. Имея данные о скорости и латентности сетевого обмена между процессами можно оценить время коммуникаций. Но если программа каким-то образом использует ввод-вывод, то для точной оценки времени выполнения необходимо иметь данные о производительности дисковых операций.

В настоящее время существует множество многопроцессорных вычислительных систем, таких как СКИФ МГУ «Чебышев» (НИВЦ МГУ), IBM Blue Gene/P (ВМК МГУ), суперкомпьютер «Ломоносов» (НИВЦ МГУ). Эти вычислительные системы используют специально выделенную программно/аппаратную подсистему для хранения данных различного характера. Это могут быть результаты экспериментов и вычислений, резервные копии и так далее. В тексте эту подсистему мы будем называть «файловым хранилищем». В случае, когда вычислительная система имеет кластерную архитектуру, на самих вычислительных узлах аппаратура для хранения может отсутствовать вовсе, или ее объем может быть сильно ограничен. Как правило доступ к общему для различных вычислительных узлов файловому хранилищу осуществляется по сети. Сами файловые хранилища могут иметь кластерную архитектуру и на них могут использоваться специальные распределенные файловые системы для достижения высокой скорости чтения и записи.

Одной из наиболее популярных технологий для разработки параллельных программ на данный момент является MPI — Message Passing Interface (интерфейс передачи сообщений). MPI предоставляет программисту набор функций MPI-IO — эффективный и переносимый интерфейс для осуществления параллельного ввода-вывода. Однако, несмотря на внешнюю простоту этого интерфейса, он скрывает под собой много тонких мест в плане своей реали-

*данная работа частично поддержана грантами РФФИ 11-07-00756-а, 11-07-00614-а и госконтрактами ФЦП «Научные и научно-педагогические кадры инновационной России» П1317, П1367.

зации — это приводит к тому, что в общем случае очень сложно делать априорные предсказания производительности параллельной программы с интенсивным вводом-выводом. В частности, для эффективного ввода-вывода важно понимать, как влияют на время работы такие параметры как: используемая функция ввода-вывода в MPI-IO, размер файла, размер блока записи, представление файла на диске. Определение степени влияния указаных выше параметров позволит выдавать рекомендации по использованию MPI-IO пользователям и программистам современных суперкомпьютеров.

2. Краткое описание MPI-IO

Поскольку работа существенно использует возможности MPI-IO авторы считают необходимым провести краткое описание архитектуры библиотеки MPI-IO. Это важно также в связи с тем, что MPI-IO является одной из наиболее популярных библиотек для организации параллельного ввода-вывода в параллельной программе.

Типичный сеанс работы с файлом в MPI включает в себя следующие пункты.

1. Вызов MPI_File_open для получения внутреннего файлового дескриптора MPI с помощью MPI_File_open.

2. Вызов MPI_File_set_view для установки «представления» (view в терминологии MPI) файла — описание способа отображения Участков файла в логическое представление файла в MPI. Данные и в оперативной памяти, и в файле могут быть представлны с пропусками – «дырками».

3. Непосредственно осуществление ввода/вывода. MPI для действий подобного рода предоставляет богатое множество функций.

4. Вызов MPI_File_close для закрытия файла и освобождения связанных с ним ресурсов.

Наибольший интерес с точки зрения времени исполнения программы представляют пункты 2 и 3: установка представления и функции непосредственного чтения-записи.

Все функции, осуществляющие ввод-вывод, имеют похожий набор параметров, в который входят:

- Файловый дескриптор MPI
- Адрес блока для чтения или записи
- Размер блока

Но между ними есть и существенные отличия. Во-первых, их можно разделить на три группы по типу позиционирования в файле:

1. Функции, работающие с точными смещениями (explicit offset), которые необходимо передавать явно в качестве параметра. Примером такой функции является MPI_File_write_at.

2. Функции, работающие с индивидуальными файловыми указателями. Для таких функций MPI самостоятельно, без вмешательства программиста поддерживает текущее смещение, которое увеличивается на размер блока после чтения/записи. Все процессы имеют независимые смещения. Пример: MPI_File_write.

3. Функции, работающие с общим файловым указателем. MPI поддерживает общее для всех процессов смещение, которое меняется после операции ввода-вывода, происходящей в любом процессе. Пример: MPI_File_write_shared.

Также функции можно подразделить на индивидуальные и коллективные. Разница между ними состоит в том, что коллективные операции должны быть вызваны всеми процессами, котрые вызывали MPI_File_open — это позволяет MPI выполнять разного рода оптимизации доступа.

Наконец, все функции можно разделить на блокирующие и неблокирующие. Блокирующие функции завершаются только после окончания соответствующей операции ввода-вывода, а неблокирующие функции позволяют программисту выполнять вычисления во время выполнения ввода-вывода, и только затем проверить успешность операции.

Функция MPI_File_set_view интересна тем, что она позволяет программисту установить представление файла — то, как он будет выглядеть для процесса. Эта функция добавляет гибкости, позволяя устанавливать начало представления, тип элементов, при этом этот тип может содержать «дыры», что бывает полезно, например, при записи и чтении матриц. Общий принцип проиллюстрирован на изображении 1.

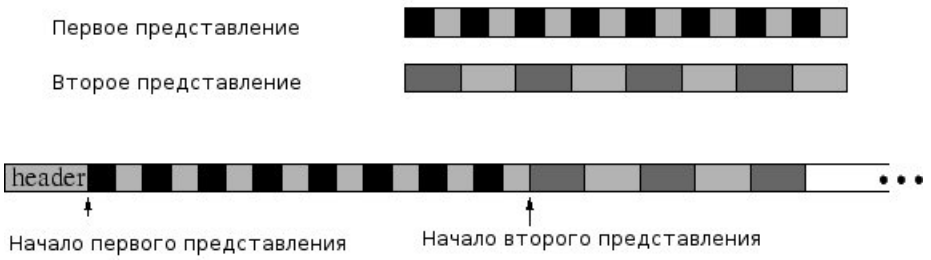

Рис. 1. Задание отображения в MPI-IO

За дополнительной и более подробной информацией можно обратиться к стандарту MPI [3].

3. Некоторые системы тестирования параллельного ввода-вывода

В этом разделе будут рассмотрены существующие системы тестирования ввода-вывода, а также обсуждены их достоинства и недостатки с точки зрения предсказания времени выполнения параллельной прогаммы.

3.1. MADBench2

Существует приложение для решения прикладной задачи космологии MADspec. На основе этого приложения был построен тест ввода-выода MADBench2 [5]. MadBench2 сохраняет всю вычислительную сложность оригинального приложения, но при этом использует автоматически сгенерированные данные. Таким образом, MADBench2 настроен на тестирование общей производительности коммуникационной, вычислительной и дисковой подсистем в контексте данной задачи. То есть MADBench2 не тестирует производительность ввода-вывода отдельно от коммуникаций и вычислений, что с одной стороны является плюсом, поскольку этот тест позволяет оценить общую производительность суперкомпьютера, а с другой – минусом, поскольку невозможно получить оценку производительности только для операций ввода-вывода.

3.2. b_eff_io

b_eff_io [4] преследует две цели: получить некоторую общую среднюю оценку производительности дисковой подсистемы суперкомпьютера, и получить более детализированную оценку производительности при разных образцах поведения программы при доступе к файлам, таким, как первая запись, перезапись, чтение, запись больших областей. Ввод-вывод

здесь тестируется при разных размерах блока, который может равняться 1 КБ, 32 КБ, 1 МБ, максимому из 2 МБ и memory_available_to_one_node/128. Тестирование также производится на размерах блока, не кратных 2. Эти размеры получаются путем добавления 8 байт к предыдущим рассмотреным размерам файла.

Результатом работы теста является число – общая оценка производительности дисковой подсистемы, а также файл с более детальными результатами для каждого образца поведения при доступе к файлу.

3.3. noncontig

Основной задачей noncontig [6] является получение данных о производительности дисковых операций, в четырех случаях: когда данные не расположены непрерывно ни в памяти, ни в файле; когда данные расположены непрерывно в памяти, но «разрывно» в файле; когда данные расположены «разрывно» в памяти, но непрерывно в файле; когда данные расположены непрерывно, как в файле, так и в памяти.

Для каждого из этих случаев печатаются средние данные о производительности.

3.4. mpi-tile-io

mpi-tile-io [7] — это еще один тест от авторов noncontig. В этом тесте файл с данными представляет собой матрицу, над которой выполняются операции чтения и записи. Тест позволяет менять размеры матрицы по вертикали и горизонтали, размер ее элементов, позволяет указать что лучше использовать на данном действии коллективные операции, или индивидуальные. После завершения теста выводятся данные о лучшем, худшем и среднем временах доступа. Также выводится некоторая общая оценка производительности, основанная на худшем времени доступа.

3.5. LANL MPI-IO Test [8]

Тест написан исследователями из Лос-Аламосской национальной лаборатории(Loss-Alamos National Laboratory, LANL), тест измеряет скорость ввода-вывода при разных патернах доступа, таких как:

- N процессов пишут в N файлов
- N процессов пишут в один файл
- N процессов отсылают данные M процессам, которые пишут в M файлов
- N процессов отсылают данные M процессам, которые пишут в один файл

При этом тест позволяет указать, являются ли данные «перемешанными» между собой. На рисунке 2 приведена иллюстрация этого принципа (серым цилиндром изображен диск, на нем разными цветами отмечены области файла, а стрелками показано соответствие между представлением файла в каждом из процессов и представлением файла на диске).

Все перечисленные системы обладают следующими недостатками: отсутствует возможность проводить серию экспериментов с разными параметрами, отсутствует возможность тестировать производительность конкретных функций MPI-IO. Для некоторых из описанных систем тестов написание новых тестов затруднительно всвязи со сложившейся в них организацией кода. С целью минимизации этих недостатков было принято решение разработать собственную систему тестов.

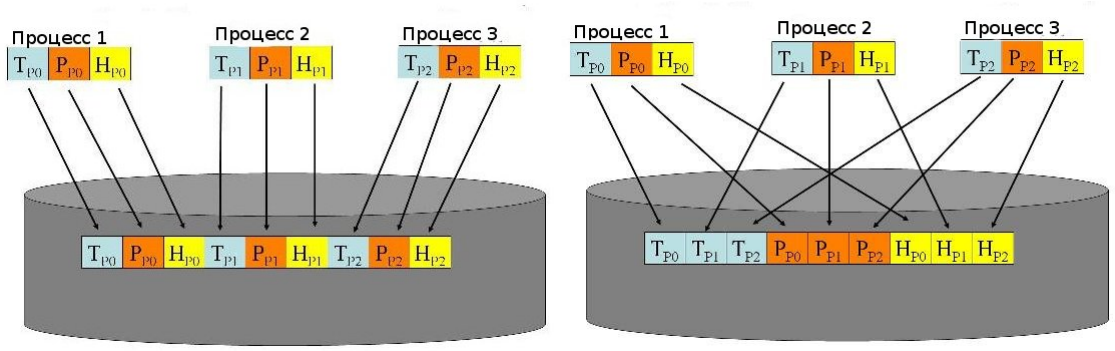

Рис. 2. Установка представления

4. Описание разрабатываемой системы тестов

Было решено разработать систему тестов для MPI-IO, позволяющую определять и оценивать производительность дисковой подсистемы вычислительных кластеров в зависимости от функции, применяемой для ввода-вывода, а также параметров этих функций: размер блока для единовременного чтения/записи, размера файла, а также представления. Запустить полученную систему тестов на суперкомпьютере СКИФ-МГУ «Чебышев» и попытаться определить характеристики, влияющие на эффективностьопераций ввода-вывода.

Разрабатываемая система тестов нацелена на исследование эффективности ввода-вывода при использовании различных функций MPI-IO, а также при различных параметрах, таких как размер файла, и размера блока данных, атомарно передаваемого в функцию MPI. Процесс тестирования производительности отдельно взятой функции ввода-вывода разбит на следующии стадии:

1. Сбрасывание буферов файловой системы. Во многих случаях после чтения файла, этот файл оказывается сохранен в свободной оперативной памяти вычислительного узла, поэтому при повторном чтении этого файла обращение к диску выполняться не будет. Такой эффект полезен в реальных приложениях, но нежелателен при тестировании производительности файлового хранилища. Сбрасывание выполняется следующим образом: файл полностью переписывается другими данными.

2. Открытие файла в режиме для чтения и записи (MPI_MODE_RDWR).

3. Установка метода отображения файла из процесса, что требуется для корректной работы MPI-IO. Для тестирования были предложены три режима. В первом, каждому процессу достается отдельный непрерывный участок файла. Во втором, каждому процессу достается файл целиком (данный режим подходит только для тестов чтения). В третьем режиме участки файлов оказываются перемежены: первому процессу достается 1-й, (NP+1)-й, (2*NP+1)-й и т.д. байты файла, второму 2-й, (NP+2)-й, (2*NP + 2)-й байты файла и так далее до NP-го процесса, которому достается (NP-1)-й (2*NP-1)-й ... (NP – обозначает количество процессов, занимающихся вводом-выводом). Для обеспечения последнего способа отображения создается новый тип данных MPI, содержащий «дыры» (своего рода маска).

4. Непосредственное осуществление ввода-вывода. На данный момент тестируются все функции из MPI-IO. Для этого в каждом процессе вычисляется количество байт, которое необходимо записать или считать, а затем исследуемая функция последовательно вызывается в каждом процессе до тех пор, пока требуемое число байт не будет прочитано или записано. Замеры эффективности выполняются с использованием функции MPI_Wtime.

5. Закрытие файла с использованием MPI_File_close.

Поддерживается проведение серии экспериментов, когда тест каждой функции выполняется с разными размером файла и размером блока для ввода или вывода. Для каждой пары параметров выдается средняя, максимальная и минимальная достигнутая скорость выполнения (в секундах, а также в КБ/с). Во время тестирования для каждого набора параметров выводятся следующие результаты тестирования (средняя, максимальная и минимальная скорости записи). После завершения выводятся матрицы результатов (матрицы собирают скорость ввода-вывода и общее время выполнения ввода-вывода). Этот режим упрощает выяснение зависимости эффективности ввода-вывода от различных параметров.

5. Исследование производительности на кластере СКИФ-МГУ «Чебышев»

Разработанная система тестов запускалась на кластере СКИФ-МГУ «Чебышев». На этом кластере используется параллельная файловая система Panasas, а объем дисковой подсистемы составляет 60 ТБ. Тестирование проводилось со следующими параметрами: размер файла составлял 100 ГБ, количество процессов – 128, размер блока, передаваемого в функции ввода-вывода менялся от 1 МБ до 99 МБ с шагом в 2 МБ.

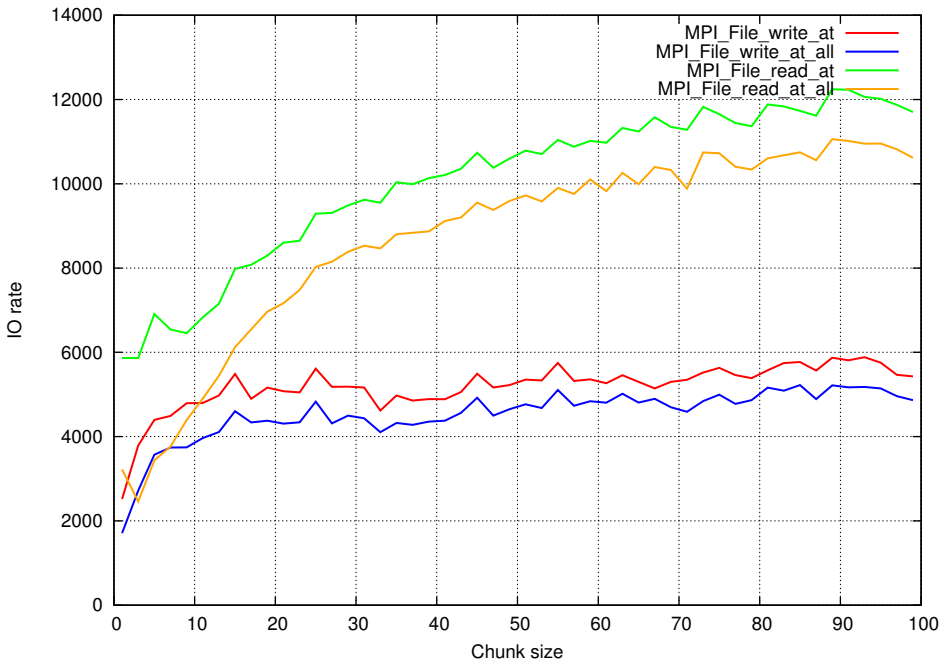

Рис. 3. Результат тестирования. Ось X – размер блока(МБ). Ось Y – эффективность функции(МБ/с)

На графике 3 показаны результаты тестирования. На этом графике видны следующие особенности:

1. Скорость записи при размере блока, ниже 5 МБ оказалась существенно ниже, чем скорость записи при размере блока $\geq$ 5 МБ, что связано с меньшим количеством накладных расходов при меньшем количестве вызовов функций.

2. Аналогичный вывод можно сделать и для чтения, только здесь хороший размер блока составляет около 25 МБ.

3. Коллективные операции проигрывают индивидуальным, из-за того, что для индивидуальных операций не требуется дополнительная синхронизация.

4. На графиках MPI_File_write_at, MPI_File_write_at_all прослеживаются как пики, так и падения производительности. Например это заметно при размере блока около 25 МБ, 45 МБ, 55 МБ, 97 МБ, а также видно несколько более мелких пиков. Аналогичное явление можно заметить и для графиков MPI_File_read_at, MPI_File_read_at_all. Причины такой регулярности пиков пока неясны, и требуют дополнительного исследованияю

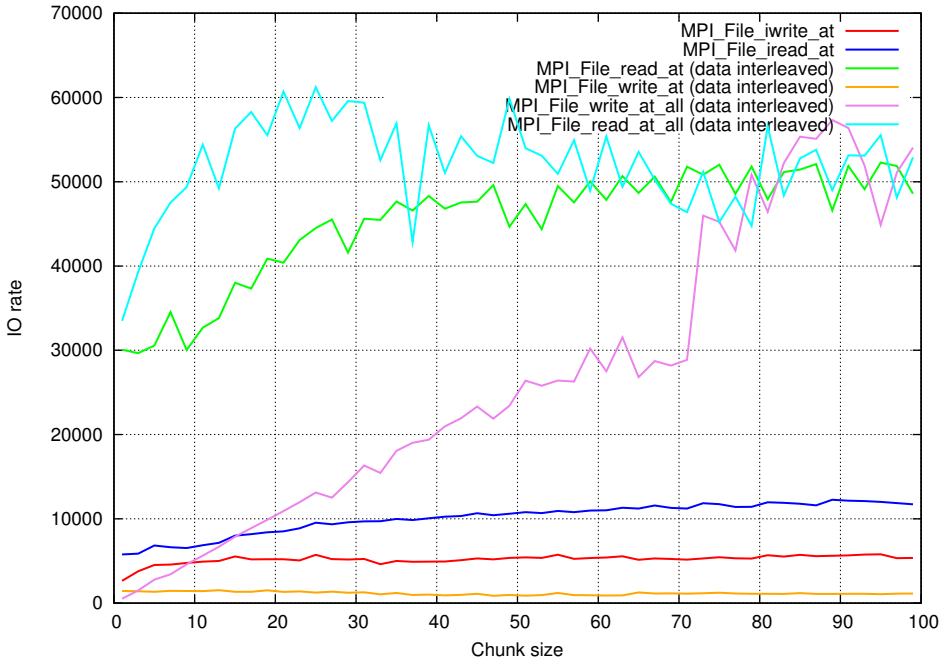

Рис. 4. Результат тестирования. Ось X – размер блока(МБ). Ось Y – эффективность функции(МБ/с)

На графике 4 приведены данные для тех же функций при третьем расположении данных (в легенде дополнительно добавлено «data interleaved», а также для наглядности приведены графики с диаграммы 3). Здесь можно выделить следующие тенденции:

1. Скорость ввода-вывода существенно выше при коллективных операциях, поскольку MPI удается оптимизировать все вызовы функций в один запрос к жесткому диску. Особенно хорошо это видно на графиках записи(оранжевый и фиолетовый). Это менее заметно на графиках чтения, поскольку при чтении из файлового хранилища происходит буферизация.

2. Скорость ввода-вывода при «перемешанном» расположении данных и достаточном размере блока значительно выше скорости ввода-вывода в случае, когда данные расположены последовательно.

3. При размере блока порядка 75 МБ происходит насыщение сети, поэтому скорость записи становится равна скорости чтения, то есть сеть, по которой данные доставляются к файловому хранилищу становится узким местом.

На графике 5 приведены результаты экспериментов, по которым была построена поверхность в трехмерном пространстве, изображающая зависимость скорости записи

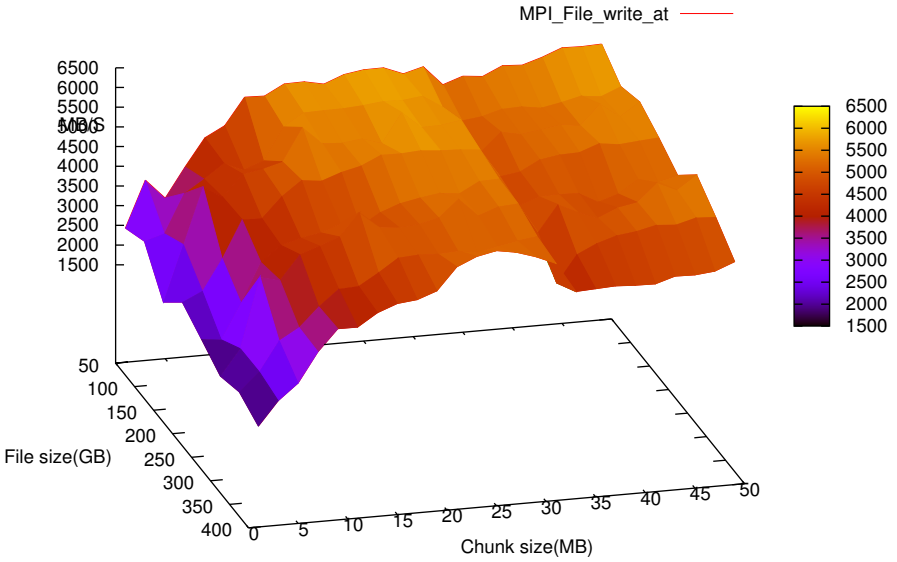

Рис. 5. Зависимость эффективности ввода-вывода от размера файла и размера блока.

(MPI_File_write_at) от размера файла и размера блока. Как видно из этого графика зависимость скорости записи от размера файла в целом несущественна. То есть характер зависимости скорости ввода-вывода и от размера блока схож при всех протестированных размерах файла. Однако, из этого графика видно, что пики в производительности могут все же иметь некоторую зависимость от размера файла.

Что касается функций, работающих с разделяемым указателем (MPI_File_read_shared, MPI_File_write_shared их их аналоги), то они показали себя плохо. Однако, это может быть связано с проблемами в настройке mvapich на кластере, поскольку во время запуска тестов наблюдалась ошибка «**io File or directory doesn't exist». Это сообщение не связано с недостатками или ошибками в разрабатываемой системе тестов, потому что аналогичное сообщение наблюдалось и при запуске систем тестов, реализованных другими разработчиками.

Разница в эффективности между неблокирующими и блокирующими вариантами функций не была обнаружена, равно как и разница между функциями, работающими с точными смещениями(explicit offsets) и их аналогами, но работающими с индивидуальными файловыми указателями (individual file pointers), такими как MPI_File_write и MPI_File_write_at, MPI_File_write_all и MPI_File_write_at_all, MPI_File_read и MPI_File_read_at и т.д. Данный факт связан с тем, что в реализации MPI-IO от mvapich происходит обращение к одним и тем же функциям более низкого уровня.

В заключение хотелось бы сказать несколько слов о масштабируемости ввода-вывода. Система тестов была запущена с аналогичными параметрами на 16 процессах, и на графике 6 приведены его результаты, из которых видно, что ввод-вывод хорошо масштабируется с 16 на 128 процессов.

6. Выводы

В ходе выполнения исследования производительности кластера СКИФ-МГУ «Чебышев» были выяснены следующие вещи:

- Эффективность ввода-вывода серьезно зависит от размера блока, записываемого или

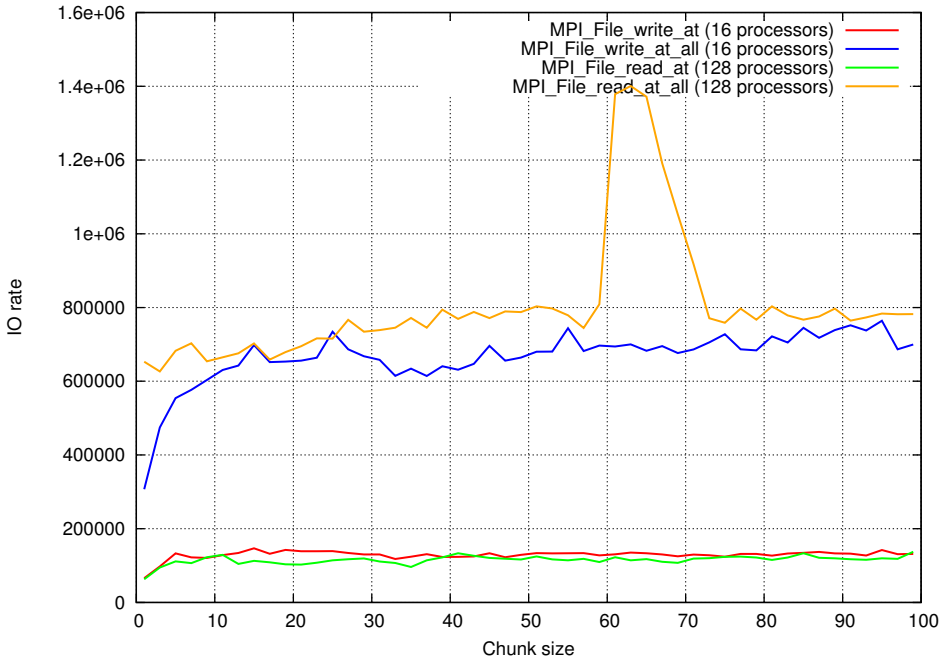

Рис. 6. Результаты тестирования. Ось X – размер блока. Ось Y – эффективность функции(МБ/С)

считываемого за один вызов функции. Запись/чтение крупными блоками более эффективна, поэтому по возможности стоит накапливать данные в памяти перед записью на диск.

- Расположение данных существенно влияет на производительность ввода-вывода, что стоит учитывать при написании приложений
- В случае записи необходимо более аккуратно подходить к выбору функции записи, поскольку разница между коллективными и индивидуальными операциями являются достаточно существенной. Что касается разницы между индивидуальными и коллективными операциями чтения, то она оказалась гораздо менее выраженной, что связано с буферизацией на файловом хранилище.
- Зависимость скорости записи от размера файла отсутствует(по крайней мере, для файлов порядка сотен ГБ).
- Ввод-вывод является хорошо масштабируемым(по крайней мере, при смене числа процессов с 16 на 128)

В дальнейшем авторы планируют расширить систему тестов новыми тестами, которые будут направлены на исследование взаимного влияния операций чтения и записи. планируется также доработать инструмент визуализации данных при помощи gnuplot или libgnuplot, чтобы наглядно представлять результаты тестирования (в текущей реализации данные необходимо предварительно преобразовывать к определенному формату). Планируется сравнить производительность различных реализаций MPI, а также производительность других вычислительных систем, к примеру IBM BlueGene/P и «Ломоносов» на операциях ввода-вывода. Планируется автоматически определять «устойчивые» пики и провалы в производительности, а также попробовать динамически менять величину шага в размере блока около таких точек.

Литература

1. Jianwei Li, Wei-keng Liao, Alok Choudhary «Parallel netCDF: A High-Performance Scientific I/O Interface» // SC '03 Proceedings of the 2003 ACM/IEEE conference on Supercomputing, 2003 P. 39, http://www.mcs.anl.gov/ robl/pnetcdf/docs/pnetcdf-sc2003.pdf

2. P. Balaji, W. Feng, J. Archuleta, H. Lin, R. Kettimuthu, R. Thakur, X. Ma «Semantics-based Distributed I/O for mpiBLAST» // PPoPP '08 Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming 2008, P. 293–294,

3. MPI: A Message-Passing Interface Standard , Version 2.2 2009 http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf

4. W. Gropp, E. Lusk «Reproducible Measurement of MPI Performance Characteristics. In J. Dongarra et al. (eds.)» // Recent Advances in Parallel Virtual Machine and Message Passing Interface, proceedings of the 6th European PVM/MPI Users' Group Meeting, EuroPVM/MPI'99, Barcelona, Spain Sept. 26-29, 1999, LNCS 1697, P. 11–18.

5. J. Borrill, L. Oliker, J. Shalf, H. Shan, A. Uselton «HPC Global File System Performance Analysis Using A Scientific-Application Derived Benchmark» http://crd-legacy.lbl.gov/ oliker/papers/PC09_MADbench.pdf

6. R. Latham, R. Ross «PVFS, ROMIO, and the noncontig Benchmark» www.mcs.anl.gov/romio/noncontig-perf.pdf

7. Parallel I/O Benchmarking Consortium http://www.mcs.anl.gov/research/projects/pio-benchmark/

8. The Los Alamos National Lab MPI-IO Test http://public.lanl.gov/jnunez/benchmarks/mpiiotest.htm