БИБЛИОТЕКА PARIMPR ДЛЯ ВИЗУАЛИЗАЦИИ И ОБРАБОТКИ ИЗОБРАЖЕНИЙ НА СУПЕРКОМПЬЮТЕРАХ

Д.Ю. Андреев, О.В. Джосан, С.В. Коробков, А.И. Серебрийский

В настоящее время актуальным являет вопрос перехода к суперкомпьютерным вычислениям петафлопсного уровня. Объем данных в таких вычислительных экспериментах достигает сотен терабайт, а в перспективе и петабайт. Например, при моделировании турбулентного горения на вычислительной системе BlueGene /P для проекта FLASH с использованием 8000 четырехядерных вычислительных узлов генерируется 16Gb данных каждые 10-15 минут, общий объем данных эксперимента составляет 300Tb[1]. Объем данных, получаемых в задаче климатического моделирования, составляет 345Tb[2]. Объем данных, получаемых в результате проведения экспериментов на системе BlueGene /P[4], установленной в МГУ, измеряется сотнями гигабайт[3]. Возможность эффективной и удобной с точки зрения пользователя работы с такими объемами данных является неотъемлемой задачей, которую требуется решить при написании прикладных программных пакетов для суперкомпьютерного моделирования на вычислителях такого уровня.

Существует множество задач по обработке данных, представимых в виде многомерного массива. При реализации параллельного алгоритма, решающего подобную задачу, с использованием библиотеки MPI, программистам часто приходится решать одни и те же вопросы, связанные с эффективной загрузкой данных из внешнего хранилища в оперативную память вычислительной системы, распределением блоков данных на определённые вычислительные узлы, сжатие данных при передачах внутри коммуникационной среды вычислительной системы. Вне зависимости от эффективности алгоритма решающего задачу, это проблемы могут повлиять на производительность. В нашей работе мы постарались создать программную структуру, которая позволить разделить функциональность, направленную на решение вышеуказанных проблем, и реализацию алгоритма, решающего прикладную задачу.

В данной работе предложен интерфейс Parimpr, разработанный на факультете ВМК МГУ имени М.В. Ломоносова, который позволить разнести функциональность, направленную на решение вышеуказанных проблем ввода-вывода, и алгоритм, решающий прикладную задачу. Интерфейс Parimpr реализован в формате библиотеки и может быть использован для оптимизации работы MPI-программ, решающих задачи, где ввод-вывод занимает существенную часть времени вычислений. В частности, применение библиотеки Parimpr эффективно при решении на суперкомпьютерах задач, связанных с визуализацией результатов научных вычислений и обработкой изображений большого размера.

Архитектура и функциональность библиотеки Parimpr

Архитектурные особенности вычислительной системы могут повлиять на скорость загрузки данных из внешней памяти. Рассмотрим этот вопрос на примере суперкомпьютера IBM BlueGene /P расположенного на факультете ВМК МГУ имени М.В. Ломоносова.

Операции ввода/вывода в файловую систему контролируются специальными процессорами, которые не участвуют в вычислениях, и их общее количество сильно меньше количества вычислительных узлов. Каждый вычислительный узел может обращаться к одному определённому процессору ввода-вывода. Процессоры ввода-вывода обращаются к внешним устройствам хранения через выделенную коммуникационную сеть. С помощью библиотеки mpix[5] можно определить, какие узлы используют один процессор ввода-вывода, но данная библиотека завязана на архитектуре IBM BlueGene /P, для другой вычислительной системы средства получения информации о архитектуре ввода-вывода могут выглядеть иначе или вообще отсутствовать. В нашей работе все архитектурно зависимые возможности вынесено в отдельную функцию, которая должна быть вызвана до начала работы с другими возможностями разрабатываемой библиотеки. В результате работы этой функции, выясняется сколько доступно узлов ввода-вывода и какие процессоры с ними связаны. Одной из целей разрабатываемой библиотеки является распределение нагрузки между процессорами ввода-вывода. Для параллельного доступа к одним и тем же данным требуется так же поддержка со стороны файловой системы и формата хранения данных. В IBM Blue Gene/P МГУ используется файловая система GPFS[6], позволяющая высокоскоростной одновременный доступ к файлам из приложений. Для примера формата хранения данных используется форматы предоставляемый библиотеками HDF5[7] и NetCDF. Остальная функциональность библиотеки не завязана на типе источника данных.

Пользователю предоставляется возможность выбрать количество процессоров, которые он хочет использовать в работе с нашей библиотекой, и количество узлов, которые он предполагает использовать для загрузки данных. Если пользователь хочешь использовать количество процессоров, меньшее чем изначально выделено вычислительной системой, то создаётся MPI-коммуникатор, в рамках которого будут работать функции разрабатываемой библиотеки, иначе используется глобальный MPI-коммуникатор. Для выделенного коммуникатора автоматически выбираются узлы, которым соответствует максимальное количество процессоров ввода-вывода. Дальнейшая работа с внешними устройствами хранения происходит на выбранных узлах, вычисления могут производиться на всех остальных, выделенных для работы библиотеки. В рамках одной

программы может быть создано несколько не пересекающихся MPI-коммуникаторов для различной работы с библиотекой.

В некоторых задачах требуется обрабатывать одни и те же данные разными способами или организовывать конвейер обработчиков. В рамках одного коммуникатора, выделенного для библиотеки, программист может указать, сколько ему требуется процессоров для одного обработчика данных. Одновременно может быть запущенно несколько обработчиков, использующих одни входные данные, но исполняющихся на не пересекающихся подмножествах процессоров. Одна последовательность обработчиков исполняется в рамках одного MPI-коммуникатора, что позволяет отделить разработку алгоритма решающего отдельную задачу от вопросов извлечения данных из внешних устройств хранения и их распределения по вычислительным узлам. Для распределения данных достаточно указать, какой блок данных требуется для каждого процессора. После распределения запускается последовательность обработчиков, в качестве обработчика может выступать функция, реализующая параллельный алгоритм обработки данных, распределённых в рамках одного MPI-коммуникатора. Результаты работы функций представляют блоки данных, которые необходимо собрать на узлах работы с внешними устройствами или перераспределить для дальнейшей обработки.

Каждая часть библиотеки может модифицироваться независимо от других. Вся архитектурная зависимость вынесена в один вызов сбора информации о вычислительной системе. Планируется поддержка других источников данных, различные алгоритмы обработки данных, несколько стратегий выбора узлов работы с внешними устройствами (на основе информации о вычислительной системе или на основе результатов измерения производительности системы ввода-вывода).

Пример использования библиотеки Parimpr: параллельная реализация алгоритма RHSEG

В качестве примера фильтра был реализован фильтр сегментации изображения по алгоритму RHSEG. Данный алгоритм по заданному изображению строит иерархию сегментаций, каждая их которых представляет собой “улучшение” предыдущей. Алгоритм RHSEG представляет собой модификацию алгоритма HSEG[8].

Алгоритм HSEG очень требователен к ресурсам, в частности потребление памяти составляет O(s2), где s - площадь картинки, количество сравнений также составляет O(s2) Поэтому запуск алгоритма на больших изображениях затруднителен. Для преодоления этих сложностей был разработан алгоритм RHSEG.

Суть алгоритма RHSEG заключается в следующем:

1. Изображение делится на 4 части.
2. Для каждой части рекурсивно вызывается либо RHSEG, либо HSEG без последнего шага. Что именно вызывается, определяется настройками пользователя и текущим уровнем рекурсии.
3. Полученные сегментации частей собираются в одну сегментацию.
4. Над полученной сегментацией вызывается HSEG без последнего шага или, в случае если это нулевой уровень рекурсии, полный HSEG.

Рассмотрим параллельную реализацию данного алгоритма. Его параллельная реализация предусматривала, что шаг 2 выполняется на узлах, отличных от того, на котором выполняется шаги 1,3,4. Т.е. узел разбивал изображение на части, отдавал каждому из подчиненных ему узлов часть, ждал пока они сходным образом обработают её, объединял полученные сегментации в одну, выполнял над ней HSEG без 7-го шага и возвращал тому узлу, которому он подчинялся. Исключение составлял самый первый процесс – он выполнял полный HSEG и возвращал пользователю иерархию сегментаций.

Для использования архитектурных особенностей BlueGene /P в алгоритм были внесены следующие изменения:

1. Изменен порядок рекурсии – в исходном алгоритме порядок был следующий:
   - на вход узлу подавалось изображение
   - он разбивал его на части, выдавал подчиненным
   - ждал, пока они обработают выданные части изображения
   - забирал обработанные результаты
   - обрабатывал полученные результаты
   - отдавал наверх или выделял иерархию сегментаций

   Заметим, что на шаге 3 узел простаивает. В текущей реализации изображение изначально разложено по узлам с помощью методов библиотеки Parimpr и потом рекурсивно собирается также с помощью методов этой библиотеки.
2. В отличии от оригинальной реализации, где изображение хранилось на подмножестве задействованных процессоров, в текущей реализации изображение изначально раскладывается по всем доступным процессам, тем самым снижая интенсивность обмена сообщениями между процессорами.
3. За счет возможности регулировать количество частей, из которых будет потом сформирован следующий уровень рекурсии, предоставляемой библиотекой Parimpr, можно менять необходимую глубину рекурсии и уменьшать количество простаивающих узлов.

Сжатие данных в библиотеке Parimpr

Поскольку библиотека Parimpr нацелена на обработку больших массивов данных, то встает актуальная задача по уменьшению затрачиваемой на хранение данных памяти и также уменьшению времени передачи этих

данных. Данную задачу можно эффективно решать путем сжатия данных. В тоже время, необходимо понимать, что сжатие и декомпрессия данных не должны занимать время существенно большее, чем сам процесс обработки данных. Следует выделить несколько периодов параллельной обработки данных, для которых целесообразно применять сжатие:

1. Этап загрузки массива данных с жесткого диска в оперативную память вычислителя. Как известно, в вычислителях скорость обмена с жестким диском очень низкая из-за характеристик самих жестких дисков и из-за того, что большинство узлов построены без доступа к внешним носителям для уменьшения количества прерываний и, соответственно, затрачиваемого времени на их обработку. Библиотека Parimpr, используемая для чтения и записи данных, позволяет применять конвейер фильтров к данным и также позволяет сжимать данные алгоритмом SZIP.
2. Этап хранения массивов данных в памяти. Объемы данных, которые необходимо хранить при обработке изображений, велики. Для эффективной обработки и уменьшения числа процессоров вычислителя затрачиваемых на хранение данных и их обработку можно применять сжатие массивов данных хранящихся в памяти. При разработке алгоритмов сжатия данных в памяти необходимо учитывать особенности операций, проводимых над данными. В качестве примера был разработан алгоритм сжатия данных в памяти для алгоритма RHSEG, который требователен к памяти и требует переименования значений больших областей данных.
3. Этап распределения данных с узлов ввода-вывода на вычислительные узлы. На данном этапе целью сжатия является уменьшение времени простоя вычислительных узлов, затрачиваемое на получение информации предназначенной к обработке. Здесь важно понимать, что время сжатия, декомпрессии данных и пересылки сжатых данных не должно превышать время пересылки несжатых данных.

В работе предложена концепция интерфейса Parimpr, предназначенная для эффективной работы с данными большого объема в системах обработки изображений и визуализации на суперкомпьютерах терафлопного и петафлопсного диапазона. Рассмотрен пример использования данной библиотеки для реализации алгоритма сегментации RHSEG. Предложена концепция сжатия потоков данных в системе Parimpr. Дальнейшее направление работ предполагает развитие набора алгоритмов обработки изображений и визуализации, для которых использование библиотеки Parimpr существенно улучшит производительность. Также дальнейшим направлением работ является перенесение разработанных алгоритмов на другие платформы.

Работа выполнена в рамках НОЦ «Суперкомпьютерные технологии для решения задач обработки, хранения, передачи и защиты информации» при поддержке Федеральной целевой программы "Научные и научно-педагогические кадры инновационной России" на 2009 - 2013 годы и грантов РФФИ 08-07-00445-а, 09-07-12068-офи_м.

ЛИТЕРАТУРА:

1. T. Peterka, R. B. Ross, H.-W. Shen, K.-L. Ma, W. Kendall, and H. Yu, "Parallel Visualization on Leadership Computing Resources," Preprint ANL/MCS-P1656-0709, July 2009.
2. T. Peterka, R. Ross, H. Yu, K.-L. Ma, W. Kendall, and J. Huang, "Assessing and Improving Large Scale Parallel Volume Rendering on the IBM Blue Gene/P," Preprint ANL/MCS-P1554-1008, October 2008
3. Джосан О.В., Попова Н.Н., Шумкин Г.Н. Методы визуальной поддержки для задач молекулярного моделирования на суперкомпьютере Blue Gene /P // тезисы конференции «Научный сервис в сети Интернет», Россия, Новороссийск, 2009, сс. 426-428, М.: Изд-во МГУ, 2009.
4. Сайт BlueGene /P, установленного в МГУ http://hpc.cs.msu.su
5. Библиотека mpix www.mpix.com/
6. Система GPFS http://ru.wikipedia.org/wiki/General_Parallel_File_System
7. Формат хранения данных HDF5 http://www.hdfgroup.org/HDF5/
8. J. C. Tilton, Method for Recursive Hierarchical Segmentation combining Greedy and Hierarchical Stepwise Optimal Approaches and Region Splitting // NASA Case No. GSC 14,474-1, NASA’s Goddard Space Flight Center, April 19, 2001.
9. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing, 22(6):789–828, September 1996.
10. Carlos Sosa and Brant Knudson. IBM System Blue Gene Solution: Blue Gene/P Application Development. International Technical Support Organization, August 2009.
11. Evan Speight Jian Ke, Martin Burtscher. Runtime compression of mpi messages to improve the performance and scalability of parallel applications. In Proceedings of the 2004 ACM/IEEE conference on Supercomputing, page 59, November 2004.
12. Martin Burtscher Paruj Ratanaworabhan, Jian Ke. Fast lossless compression of scientific floating-point data. In Proceedings of the Data Compression Conference, pages 133–142, 2006.
13. Paruj Ratanaworabhan Martin Burtscher. Fpc: A high-speed compressor for double-precision floating-point data. IEEE Transactions on Computers, pages 18–31, January 2009.

14. Alejandro Calderón Rosa Filgueira, David E. Singh and Jesús Carretero. Compi: Enhancing mpi based applications performance and scalability using run-time compressio. In Recent Advances in Parallel Virtual Machine and Message Passing Interface, pages 207–218. Springer Berlin / Heidelberg, 2009.