# ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ РАЗЛИЧНЫХ МЕТОДОВ СЖАТИЯ ПРИ ПЕРЕДАЧЕ ДАННЫХ ИЗ ОСНОВНОЙ ПАМЯТИ В ПАМЯТЬ СОПРОЦЕССОРА INTEL XEON PHI

**К.Ю. Беседин, П.С. Костенецкий**

*Южно-Уральский государственный университет (национальный исследовательский университет)*

**Введение**

Как графические ускорители, так и многоядерные сопроцессоры обладают рядом технических особенностей, которые необходимо принимать во внимание при разработке высокопроизводительных алгоритмов для данных устройств. Одной из таких ключевых особенностей является необходимость передачи данных по шине PCIe из основной памяти в память устройства и обратно. Поскольку пропускная способность шины PCIe в несколько раз ниже пропускной способности основной памяти, то такая передача данных считается одним из «узких мест» при программировании для GPU и многоядерных сопроцессоров [7, 8]. В данной ситуации существует несколько путей повышения производительности кода для данных устройств. Во-первых, разработчики оборудования и инструментального ПО к нему предоставляют различные средства, позволяющие эффективно использовать возможности устройств, например, возможность параллельно выполнять передачу данных и выполнения кода на устройстве или выполнять передачу только измененных элементов данных. Во-вторых, разработчики алгоритмов учитывают данную особенность и сводят к минимуму количество данных, передаваемых между GPU или многоядерным сопроцессором и основной памятью. Часть из вносимых таким образом изменений в алгоритмы не зависит от предметной области. К таким относятся например, сохранение данных на устройстве или предпочтение вычисления значений данных их передаче из основной памяти. Некоторые же изменения специфичны для предметной области. Одним из примеров таких изменений можно считать применение сжатия данных в контексте систем баз данных.

Сжатие данных является хорошо изученным и распространенным приемом, применяемым в системах баз данных как для увеличения максимального объема хранимых данных, так и для повышения производительности за счет уменьшения объема данных, передаваемых во время операций ввода-вывода [5, 6, 9, 11]. Сжатие данных в поколоночных СУБД часто рассматривается отдельно. Особенности поколоночного хранения данных позволяют повысить как степень сжатия данных, так и скорость их обработки по сравнению с несжатыми данными [1, 2, 3].

Реализации алгоритмов сжатия данных на графических ускорителях посвящено несколько научных работ. В [13] рассмотрена реализация как алгоритмов сжатия с потерями (JPEG, vorbis), так и без потерь (LZ77). В работе [10] представлена реализация алгоритма LZSS. В [4] рассмотрено несколько алгоритмов сжатия в контексте систем баз данных. Исследований алгоритмов сжатия на Intel Xeon Phi до данного момента не проводилось.

В рамках данной работы рассматриваются три используемых в СУБД метода сжатия данных: Run Length Encoding (RLE, кодирование длин серий), Null Suppression (подавление нулей) и LZSS (Lempel-Ziv-Storer-Szymanski). Разработаны параллельные алгоритмы, реализующие рассматриваемые методы сжатия. Выполнена реализация разработанных алгоритмов на языке C++ с использованием технологии OpenMP. Проведены вычислительные эксперименты, исследующие эффективность использования рассмотренных методов сжатия при передаче данных из основной памяти в память сопроцессора Intel Xeon Phi.

**Примитивы обработки данных**

Использование небольшого набора оптимизированных простейших функций (примитивов) является эффективным способом реализации алгоритмов обработки данных [4]. Было спроектировано и реализовано несколько таких примитивов, на основе которых в дальнейшем были реализованы алгоритмы сжатия данных. Далее представлено описание этих примитивов:

- *map* параллельно выполняет итерации от *0* до *n*, вызывая пользовательскую функцию для каждой итерации
- *parallelFor* используется для параллельного выполнения итераций простого цикла со счетчиком. Примитив распределяет итерации цикла на подинтервалы между доступными процессорами/ядрами и вызывает пользовательскую функцию для каждого подинтервала
- *scatter* получает на вход два массива: массив данных и массив позиций для записи. Каждый элемент массива данных записывается в выходной массив в позицию, указываемую в соответствующем элементе массива позиций записи

- *inclusivePrefixSum* принимает на вход числовой массив $a_1, a_2, \ldots, a_n$ Результатом работы примитива является массив $b_1, b_2, \ldots, b_n$, где $b_i = \sum_{i=1}^{n} a_i$
- *exclusivePrefixSum* принимает на вход числовой массив $a_1, a_2, \ldots, a_n$ Результатом работы примитива является массив $b_1, b_2, \ldots, b_n$, где $b_i = \sum_{i=1}^{n-1} a_i$

**Реализация методов сжатия**

Для повышения быстродействия в выполненной реализации метода *Null Suppression* размер сжатого элемента может составлять 1, 2, 4 или 8 байт. При сжатии выбирается наименьший возможный размер. Алгоритм сжатия состоит из следующих шагов:

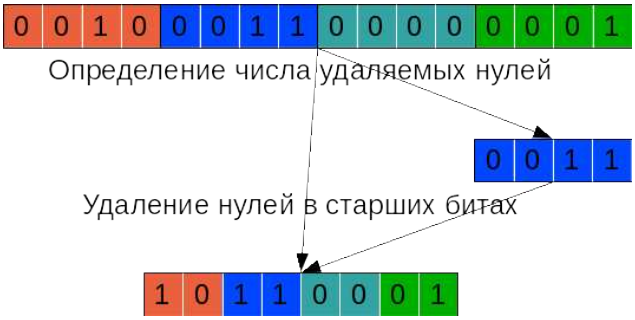


Рис. 1. Схема выполнения сжатия данных методом *Null Suppression*

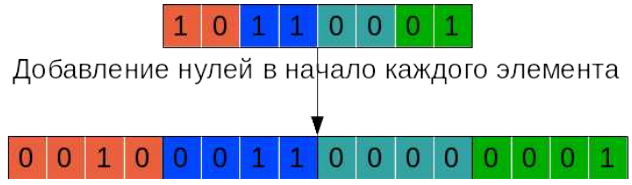


Рис. 2. Схема выполнения распаковки данных, сжатых методом *Null Suppression*

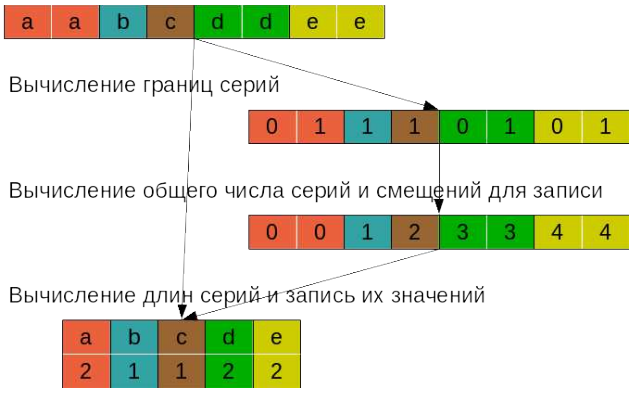


Рис. 3. Схема выполнения сжатия данных методом *RLE*

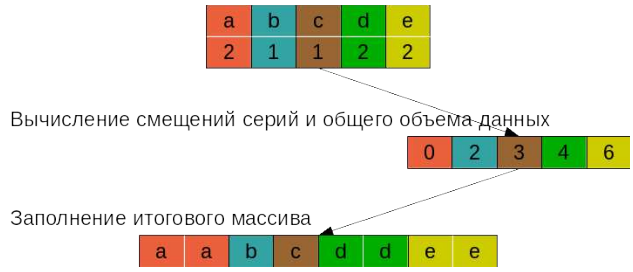


Рис. 4. Схема выполнения распаковки данных, сжатых методом *RLE*

1. определение числа удаляемых нулевых байтов (для простоты считается, что значение элемента, занимающего наибольшее число бит, известно заранее);
2. удаление из каждого элемента сжимаемых данных нулевых байтов с помощью примитива *map*.

Данный алгоритм схематично изображен на рис. 1**.**

Для распаковки данных осуществляется обратная операция: добавление определенного числа нулей в начало каждого элемента сжатых данных. Алгоритм распаковки схематично изображен на рис. 2. Представленные алгоритмы сжатия и распаковки позволяют эффективно использовать возможности, предоставляемые как современными центральными процессорами, так и сопроцессорами Intel Xeon Phi.

Для сжатия данных по методу *RLE* используется следующий алгоритм:

1. вычисление границ серий. В оперативной памяти формируется массив двоичных значений. Для элементов, которые завершают свою серию в этом массиве ставится 1, для остальных элементов — 0;
2. на основе данных о границах серии вычисляется их общее число и смещения серий в итоговом массиве значений серий;
3. вычисление длин серий и заполнение итоговых массивов значений серий и длин серий.

Схема выполнения данного алгоритма приведена на рис. 3.

Алгоритм распаковки данных, сжатых методом *RLE* схематично изображен на рис. 4. Распаковка данных, сжатых по методу *RLE*, происходит в два этапа:

1. вычисление объема распакованных данных и смещений в нем серий на основе массива длин серий;

2. на основе вычисленного на предыдущем этапе массива и массива значений серий заполняется итоговый массив с распакованными данными.

В отличие от метода Null Suppression, метод сжатия RLE не позволяет обрабатывать элементы сжимаемых данных независимо друг от друга. Для выполнения параллельного сжатия необходимо разделить последовательность сжимаемых данных на непересекающиеся подпоследовательности, которые можно обрабатывать независимо друг от друга. Распаковка данных, сжатых алгоритмом RLE, позволяет обрабатывать каждый элемент архива отдельно. Для эффективного использования возможностей, предоставляемых сопроцессором Intel Xeon Phi, представленные алгоритмы используют описанные ранее примитивы обработки данных там, где это возможно.

Оригинальный алгоритм сжатия *LZSS*, описанный в [12] не позволяет выполнить параллельное сжатие и распаковку данных. Для обеспечения параллельного сжатия и распаковки сжимаемые данные и сжатый архив разделяются на некоторое число отдельных сегментов, обработка которых может быть произведена независимо друг от друга. Схема выполнения этого алгоритма приведена на рис. 5.

Алгоритм сжатия данных методом *LZSS*:

1. распределение сжимаемых данных по отдельным сегментам;
2. параллельное сжатие каждого сегмента данных по отдельности с подсчетом размера сжатых сегментов;
3. подсчет общего размера сжатых данных и смещений сегментов в итоговом архиве;
4. копирование отдельно сжатых сегментов в итоговый архив;
5. добавление к архиву метаданных.

Схематичное изображение алгоритма распаковки приведено на рис. 6. В соответствии с этим алгоритмом распаковка данных осуществляется следующим образом:

1. вычисление общего размера распакованных данных на основе метаданных архива;
2. параллельная распаковка сжатых сегментов.

Эффективность данных алгоритмов при выполнении на сопроцессоре Intel Xeon Phi достигается как с помощью использования описанных ранее примитивов обработки данных там, где это возможно, так и с помощью учета аппаратных особенностей сопроцессора при непосредственной реализации метода сжатия.

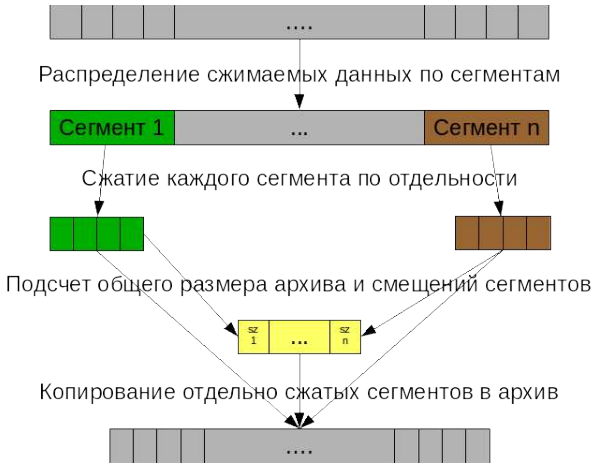


Рис. 5. Схема выполнения сжатия данных методом *LZSS*

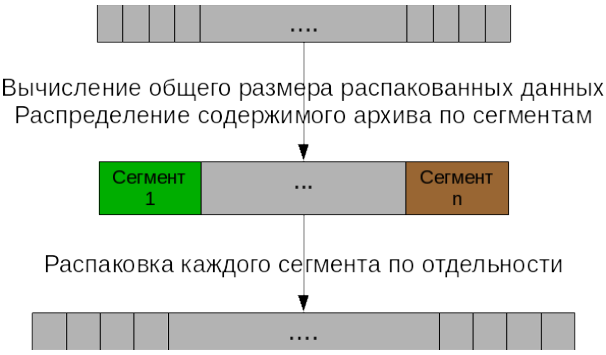


Рис. 6. Схема выполнения распаковки данных, сжатых методом *LZSS*

**Вычислительные эксперименты**

Вычислительные эксперименты производились на вычислительном кластере «Торнадо ЮУрГУ», характеристики которого представлены в табл 1.

Табл. 1. Суперкомпьютер «Торнадо ЮУрГУ»

| | | |
|---|---|---|
| Вычислительный узел | Процессор | Intel Xeon X5680 3.33 GHz |
| | Объем ОЗУ | 24 / 48 GB |
| | Число процессоров | 2 |
| | Сопроцессор | Intel Xeon Phi 7110X |
| Число вычислительных узлов | | 480 |

Степень сжатия каждого из рассматриваемых методов зависит от определенных характеристик сжимаемых данных [4]. Для каждого из методов сжатия будет представлена такая характеристика и продемонстрировано ее влияние на эффективность использования метода.

Некоторые методы сжатия позволяют осуществлять обработку сжатых данных без предварительной распаковки [2]. Среди рассмотренных в данной работе методов, к ним относятся методы RLE и Null Suppression.

Для них в экспериментах рассматривается такая обработка данных. Для примера, в качестве примера процедуры обработки данных рассматривается вычисление суммы элементов сжимаемого массива.

Все схемы были протестированы на данных объемом 250 МБ, 500 МБ, 750 МБ, 1000 МБ, 1250 МБ и 1500 МБ. В данном разделе представлены графики для 1500 МБ. Результаты для других объемов данных аналогичны.

Во всех вычислительных экспериментах, сжимаемые данные представляют собой массивы из беззнаковых целых чисел типа *uint64_t*. Вычислительные эксперименты, тестирующие методы RLE и LZSS использовали одни и те же наборы тестовых данных. Для вычислительных экспериментов, тестирующих метод Null Suppression использовался отдельный набор тестовых данных.

Для каждого метода сжатия данных приводится зависимость степени сжатия от выбранной характеристики сжимаемых данных. Также, для каждого метода сжатия сравниваются следующие показатели:

- время обработки несжатых данных;
- время обработки сжатых данных с их распаковкой;
- время обработки данных в сжатом виде (если это возможно).

Под временем обработки понимается сумма времени передачи данных, их распаковки (если требуется) и время выполнения над ними агрегатной функции - вычисления суммы При выполнении экспериментов считается, что данные находятся в оперативной памяти вычислительной системы в уже сжатом виде.

Метод сжатия *RLE* эффективен в тех случаях, когда длины серий в сжимаемых данных достаточно велики, чтобы совокупность их закодированных представлений занимала меньший объем памяти, чем сами серии [11]. Это выполняется тогда, когда число серий достаточно низко. Это число зависит от конкретной реализации метода сжатия *RLE* и от объема сжимаемых данных. Иногда, для оценки эффективности метода сжатия используется не число серий, а их средняя длина [4]. В этом случае, как и с числом серий, для определения степени сжатия требуется учитывать особенности конкретной реализации метода и объем сжимаемых данных. Для упрощения представления результатов вычислительных экспериментов для разных объемов данных в качестве варьируемой характеристики используется отношения числа серий на общее число элементов в тестовых данных. В проведенных экспериментах отношение числа серий к общему объему данных варьировалось от 0.01 до 1.

На рис. 7 показана зависимость степени сжатия 1500 МБ данных методом *RLE* от отношения числа серий к общему числу элементов данных. Из графика видно, что выполненная реализация метода эффективна когда число серий не превышает половины от числа сжимаемых элементов.

На рис. 8 показана зависимость времени обработки 1500 МБ сжатых методом *RLE* данных от отношения числа серий к общему числу сжимаемых элементов. Из графика видно, что метод *RLE* позволяет ускорить обработку баз данных на сопроцессоре Intel Xeon Phi когда соотношение числа серий в сжимаемых данных к общему числу сжимаемых элементов не превышает 0,3 в случае предварительной распаковки сжатых данных, либо 0,5, в случае обработки данных в сжатом виде.

На основании полученных результатов можно сделать следующие выводы:

- метод сжатия RLE может быть эффективно использован при передаче данных в память сопроцессора Intel Xeon Phi если число серий в данных достаточно мало;
- обработка данных в сжатом виде позволяет дополнительно увеличить эффективность использования метода *RLE*.

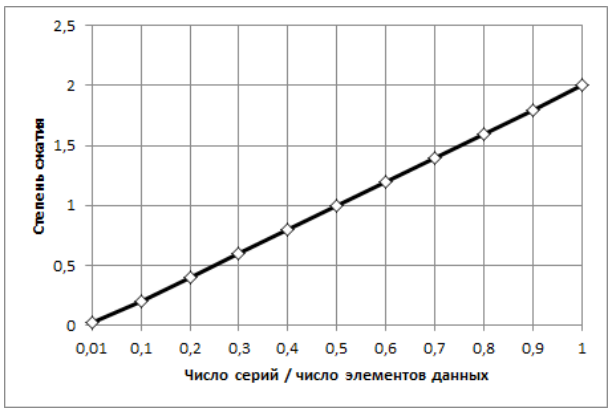


Рис. 7. Степень сжатия данных методом *RLE*

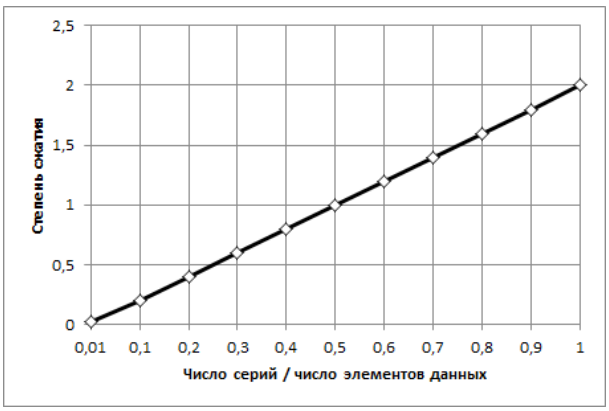


Рис. 8. Время обработки данных, сжатых методом *RLE*

Для описания зависимости эффективности сжатия данных методом *LZSS* от характеристик сжимаемых данных была использована та же характеристика, что и для метода RLE - отношение числа серий к общему числу сжимаемых данных. В отличие от методов RLE и Null Suppression, метод LZSS не позволяет производить обработку данных без их распаковки.

На рис. 9 показана зависимость степени сжатия методом *LZSS* от выбранной характеристики при обработке 1500 МБ данных. График показывает, что степень сжатия, обеспечиваемая выполненной реализацией метода *LZSS*, изменяется от 0,04 до 0,4. Видна зависимость степени сжатия от выбранной характеристики данных. Также следует отметить, что выполненная реализация *LZSS* в большинстве случаев обеспечивает более высокую, чем остальные методы степень сжатия.

На рис. 10 показана зависимость времени обработки данных, сжатых методом *LZSS* от отношения числа серий к общему числу сжимаемых элементов данных. Из графика видно, что время, требуемое на передачу сжатых данных, их распаковку и обработку, в большинстве случаев больше времени, требуемого на передачу и обработку несжатых данных. Это вызвано тем, что особенности метода сжатия LZSS не позволяют эффективно использовать векторные операции при распаковке данных.

Исходя из результатов данных экспериментов, можно заключить, что метод сжатия *LZSS* может быть эффективно использован при передаче данных из основной памяти в память сопроцессора Intel Xeon Phi только в тех случаях, когда число серий в сжимаемых данных мало.

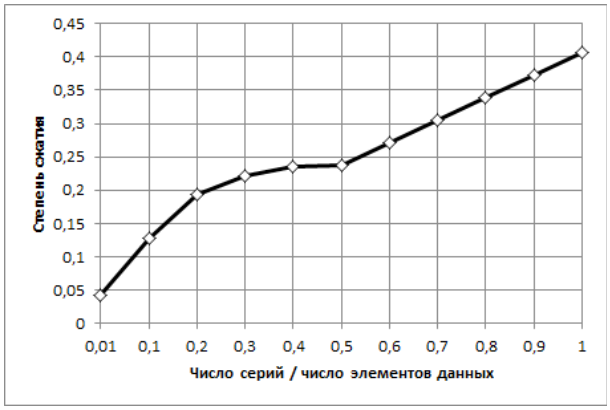


Рис. 9. Степень сжатия данных методом *LZSS*

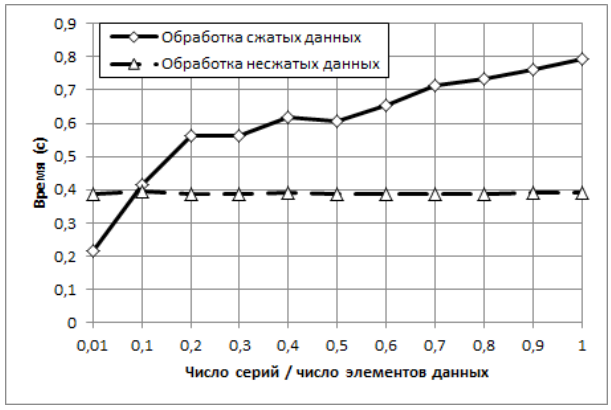


Рис. 10. Время обработки данных, сжатых методом *LZSS*

Эффективность сжатия метода *Null Suppression* зависит только от значений сжимаемых данных. Для рассматриваемых наборов данных, степень сжатия будет определяться значением максимального элемента сжимаемых данных, которое будет варьироваться так, чтобы оно требовало 1, 2, 4 или 8 байт для сжатия.

На рис. 11 показана зависимость степени сжатия данных методом *Null Suppression* от максимального значения сжатых данных. В лучшем случае, степень сжатия составляет 0.125, а в худшем - 1.

На рис. 12 показана зависимость времени обработки 1500 МБ данных, сжатых методом *Null Suppression* от значения максимального элемента этих данных. Из графика видно, что передача и обработка сжатых данных с их распаковкой эффективнее передачи и обработки несжатых данных в случае, когда максимальный элемент сжимаемых данных кодируется менее чем четырьмя байтами. Время передачи и обработки сжатых данных без их распаковки не превышает времени передачи и обработки несжатых данных.

На основании проведенных экспериментов можно сделать следующие выводы:

- метод сжатия Null Suppression может быть эффективно использован для организации обработки баз данных на сопроцессоре, если сжимаемые данные лежат в ограниченном диапазоне;
- обработка данных в сжатом виде дополнительно повышает эффективность данного метода.

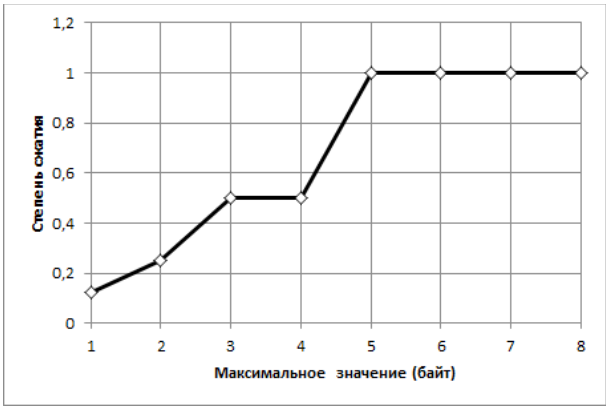


Рис. 11. Степень сжатия данных методом *Null Suppression*

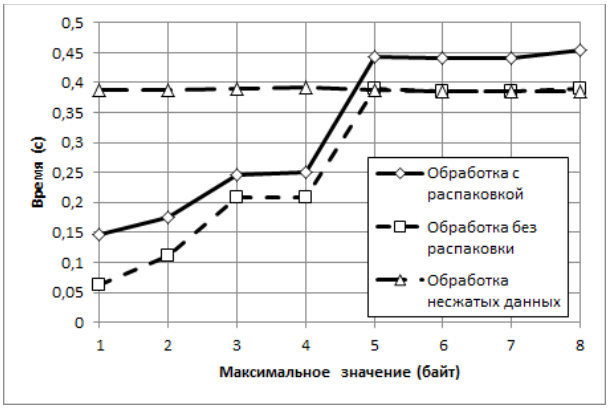


Рис. 12. Время обработки данных, сжатых методом *Null Suppression*

**Заключение**

В рамках данной работы рассмотрено три метода сжатия данных: Run Length Encoding, Null Suppression и LZSS. Были разработаны параллельные алгоритмы, реализующие рассматриваемые методы сжатия.

Разработанные алгоритмы были реализованы на языке C++ с использованием технологии OpenMP. Проведены вычислительные эксперименты, показывающие, что все рассмотренные методы сжатия могут быть эффективно использованы для обработки баз данных на сопроцессоре Intel Xeon Phi в случае, когда обрабатываемые данные удовлетворяют определенным условиям. Кроме того, показано, что обработка данных в сжатом виде при использовании методов RLE и Null Suppression позволяет дополнительно увеличить эффективность применения данных методов.

Дальнейшими направлениями исследований будут:

- исследование эффективности использования других методов сжатия;
- исследование эффективности применения комбинаций из нескольких методов сжатия в контексте данной задачи.

Работа выполнена при поддержке гранта Президента РФ МК-3711.2013.9 (2013-2014 гг.): "Моделирование параллельной обработки запросов на высокопроизводительных многопроцессорных системах с многоядерными ускорителями".

ЛИТЕРАТУРА:

1. Abadi D.J., Madden S.R., Ferreira M.C. Integrating Compression and Execution in Column-oriented Database Systems // ACM SIGMOD International Conference on Management of Data Chicago, IL, USA, June 2 - 29, 2006, Proceedings. ACM, 2006. - P. 671 - 682.
2. Abadi D.J., Madden S.R., Hachem N. Column-stores vs. Row-stores: How Different Are They Really? // ACM SIGMOD International Conference on Management of Data Vancouver, Canada, June 10 - 12, 2008, Proceedings. ACM, 2008. - P. 967 - 980.
3. Binnig C., Hildenbrand S., Farber F. Dictionary-based Order-preserving String Compression for Main Memory Column Stores // ACM SIGMOD International Conference on Management of Data Providence, Rhode Island, USA, June 29 - July 2, 2009, Proceedings. ACM, 2009. - P. 283 - 296.
4. Fang W., He B., Luo Q. Database Compression on Graphics Processors // 36th International Conference on Very Large Data Bases Singapore, September 13 - 17, 2010. Proceedings. VLDB Endowment, 2010. - P. 670 - 680.
5. Graefe G., Shapiro L. Data compression and database performance // ACM/IEEE-CS Symposium on Applied Computing, Kansas City, USA, April 3 - 5, 1991 Proceedings. IEEE, 1991. - P. 22 - 27.
6. Iyer B.R., Wilhite D. Data Compression Support in Databases // 20th International Conference on Very Large Data Bases Santiago de Chile, Chile, 12 - 15 September, 1994, Proceedings. Morgan Kaufmann Publishers Inc., 1994. - P. 695 - 704.
7. Jeffers, James Reinders Intel Xeon Phi coprocessor high-performance programming. USA: Elsevier, 2013. - 430 c.
8. Kirk D.B., Hwu W.W. Programming massively parallel processors. A hands-on approach. 2nd edition. USA: Elsevier, 2013. - 519 c.
9. Ng W.K, Ravishankar C.V. Block-Oriented Compression Techniques for Large Statistical Databases // IEEE Trans. on Knowl. and Data Eng. 1997. - Vol. 9, - No. 2 , - P. 314 - 328.
10. Ozsoy A., Swany M. CULZSS: LZSS Lossless Data Compression on CUDA // 2011 IEEE International Conference on Cluster Computing Washington, DC, USA, 26 - 30 September, 2011. Proceedings. IEEE Computer Society, 2011. - P. 403 - 416.
11. Roth M.A., Van Horn S.J. Database Compression // ACM SIGMOD Record, 1993. - Vol. 22, - No. 3, - P. 31 - 39.
12. Storer, James A. and Szymanski, Thomas G. Data Compression via Textual Substitution // J. ACM, 1982 - Vol. 29. - No. 4. - P. 928 - 951.
13. Wu L., Storus M., Cross D. Cs315a: Final project cuda wuda shuda: Cuda compression project. Technical report. - USA: Stanford University, 2009 - 11 p.