УДК 004.657

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

П. С. Костенецкий[1], **К. Ю. Беседин**[2]

Одной из важнейших особенностей работы с многоядерными сопроцессорами и графическими ускорителями является необходимость передачи данных по шине PCI Express (Peripheral Component Interconnect), которая для ряда задач становится узким местом. В настоящей статье исследуется применение сжатия для ускорения обмена данными с сопроцессором Intel Xeon Phi при параллельной обработке баз данных. Рассматриваются три алгоритма сжатия: LZSS (Lempel–Ziv–Storer–Szymanski), Null Suppression и RLE (Run-Length Encoding). Приводится описание реализаций выбранных методов для Intel Xeon Phi. Проведен ряд вычислительных экспериментов, в которых показано, что выбранные методы сжатия могут быть использованы для повышения эффективности обработки баз данных на многоядерном сопроцессоре при выполнении определенных условий относительно обрабатываемых данных. Показано, что в случаях, когда метод сжатия позволяет осуществлять обработку сжатых данных без предварительной распаковки, такая обработка позволяет дополнительно увеличить эффективность применения сжатия.

Ключевые слова: СУБД, сжатие данных, Intel Xeon Phi, LZSS compression, RLE compression, Null Suppression.

Введение. Обработка сверхбольших объемов данных является одной из актуальных на сегодняшний день научных проблем. Одним из важных подходов к решению этой проблемы является использование параллельных СУБД [3, 5, 14]. В рамках данного подхода особую роль играет организация эффективной обработки запросов к базам данных. Использование для этой цели многоядерных сопроцессоров и графических ускорителей является одним из перспективных направлений исследований [1, 4]. Как графические ускорители, так и многоядерные сопроцессоры обладают рядом технических особенностей, которые необходимо принимать во внимание при разработке высокопроизводительных алгоритмов для указанных устройств. Одной из таких ключевых особенностей является необходимость передачи данных по шине PCIe (Peripheral Component Interconnect express) 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 (Graphics Processing Unit) и многоядерных сопроцессоров [1, 2, 12, 13].

В рассматриваемой ситуации существуют несколько путей повышения производительности кода. Во-первых, разработчики оборудования и инструментального ПО к нему предоставляют различные средства, позволяющие эффективно использовать возможности устройств, например возможность параллельной передачи данных и исполнения кода на устройстве или выполнять передачу только измененных элементов данных. Во-вторых, разработчики алгоритмов учитывают эту 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 ним 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 [10,

[1] Южно-Уральский государственный университет, факультет вычислительной математики и информатики, просп. Ленина, 76, 454000, г. Челябинск; доцент, e-mail: kostenetskiy@susu.ru

[2] Южно-Уральский государственный университет, факультет вычислительной математики и информатики, просп. Ленина, 76, 454000, г. Челябинск; студент-магистрант, e-mail: besedin.k@gmail.com

11, 15, 17]. 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 [6–8].

Реализации алгоритмов сжатия данных на графических ускорителях посвящено несколько научных работ. В [19] рассмотрена реализация как алгоритмов сжатия с потерями (JPEG — Joint Photographic Experts Group и Vorbis format), так и без потерь (LZ77 — Lempel–Ziv–Storer–Szymanski compression algorithm). В работе [16] представлена реализация алгоритма LZSS. В [9] рассмотрено несколько алгоритмов сжатия в контексте систем баз данных. Исследований алгоритмов сжатия на Intel Xeon Phi до данного момента не проводилось.

В настоящей статье рассматриваются три используемых в СУБД алгоритма сжатия данных: Run-Length Encoding (RLE, кодирование длин серий), Null Suppression (подавление нулей) и LZSS. 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.

1. Примитивы обработки данных. Использование небольшого набора высокооптимизированных простейших функций (примитивов) является эффективным способом реализации алгоритмов обработки данных [9]. Спроектировано и реализовано несколько таких примитивов, на основе которых в дальнейшем были реализованы алгоритмы сжатия данных. Кратко рассмотрим некоторые из этих примитивов.

Примитив `parallelFor` используется для параллельного выполнения итераций простого цикла со счетчиком. Примитив распределяет `lastIndex-firstIndex` итераций на подынтервалы между доступными процессорами/ядрами и вызывает `function` для каждого подынтервала.

Примитив `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.

Примитив `inclusivePrefixSum` принимает на вход массив с числами $A_1, A_2, \ldots, A_n$. Результатом работы примитива является массив $B_1, B_2, \ldots, B_n$, где $B_i = \sum_{j=1}^{i} A_j$.

Примитив `exclusivePrefixSum` принимает на вход массив с числами $A_1, A_2, \ldots, A_n$. Результатом работы примитива является массив $B_1, B_2, \ldots, B_n$, где $B_0 = 0$, $B_i = \sum_{j=1}^{i-1} A_j$.

2. Реализация методов сжатия.

2.1. Алгоритм Null Suppression. Для повышения быстродействия в выполненной реализации алгоритма 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:

— определение числа удаляемых нулевых байтов (для простоты считается, что значение элемента, занимающего наибольшее число бит, известно заранее);

— удаление из каждого элемента сжимаемых данных нулевых байтов с помощью примитива `map`.

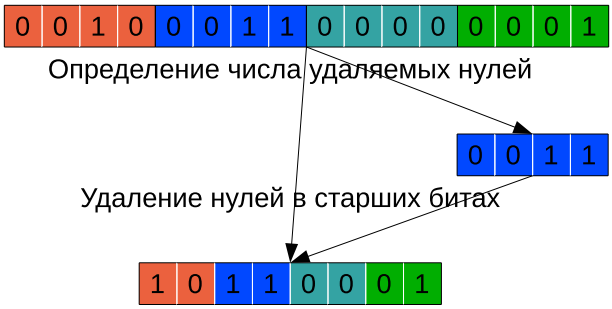

Рис. 1. Схема выполнения сжатия алгоритмом Null Suppression

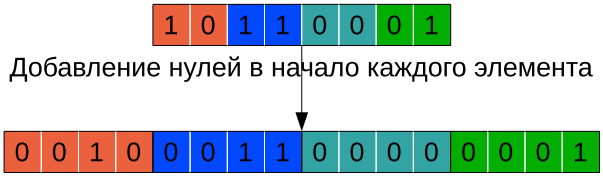

Рис. 2. Схема выполнения распаковки данных, сжатых алгоритмом Null Suppression

Рассматриваемый 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.

2.2. Алгоритм RLE. Для сжатия данных по методу RLE используется следующий алгоритм:

— 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;

— на основе данных о границах серии вычисляется их общее число и смещения серий в итоговом массиве значений серий;

— вычисление длин серий и заполнение итоговых массивов значений серий и длин серий.

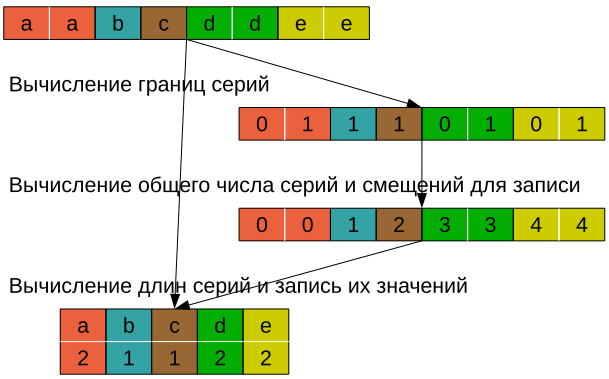

Рис. 3. Схема выполнения сжатия алгоритмом RLE

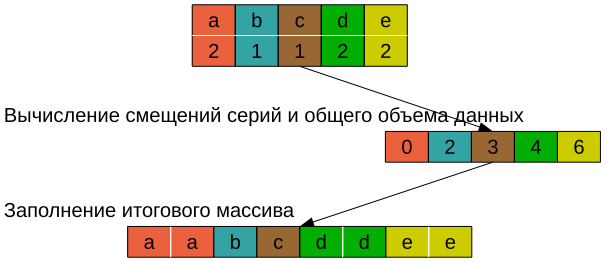

Рис. 4. Схема выполнения распаковки данных, сжатых алгоритмом RLE

Схема выполнения рассматриваемого 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.

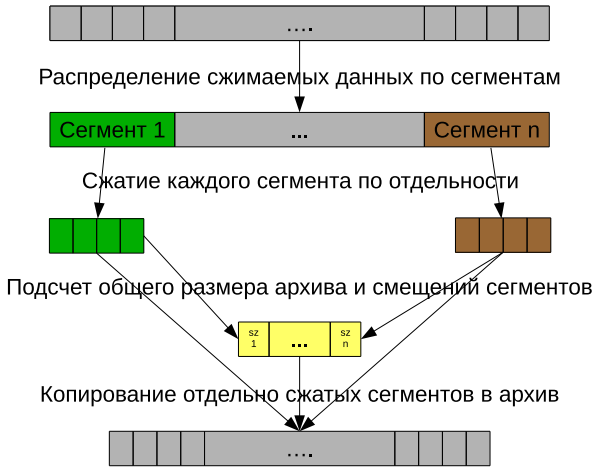

Рис. 5. Схема выполнения сжатия алгоритмом LZSS

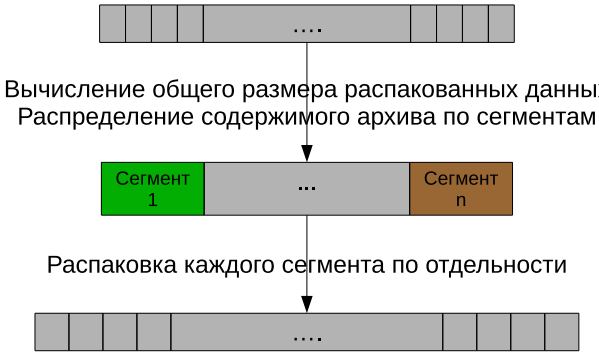

Рис. 6. Схема выполнения распаковки данных, сжатых алгоритмом LZSS

Алгоритм LZSS. 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 состоит из следующих шагов:

— распределение сжимаемых данных по отдельным сегментам;

— параллельное сжатие каждого сегмента данных по отдельности с подсчетом размера сжатых сегментов;

— подсчет общего размера сжатых данных и смещений сегментов в итоговом архиве;

— копирование отдельно сжатых сегментов в итоговый архив;

— 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 рассмотренных 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.

3. Вычислительные эксперименты. Вычислительные эксперименты производились на вычислительном кластере “Торнадо ЮУрГУ”, характеристики которого представлены в таблице.

Суперкомпьютер “Торнадо ЮУрГУ”

Процессор узла	Intel Xeon 5680 3.33 ГГц
Объем ОЗУ узла	24 / 48 Гб
Число процессоров в узле	2
Число узлов	480
Число узлов с сопроцессорами	384

Степень сжатия каждого из рассматриваемых методов зависит от определенных характеристик сжимаемых данных [9]. 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 [7]. Среди рассмотренных в данной работе методов к ним относятся методы RLE и Null Suppression. Для них в экспериментах рассматривается такая обработка данных. 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 этом разделе статьи представлены графики для объема 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. Кроме того, 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.

3.1. Метод RLE. 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 [17]. Это выполняется тогда, когда число серий достаточно мало. 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 [9]. 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 отношение числа серий к общему числу 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 объема 1500 Мб данных методом RLE от отношения числа серий к общему числу элементов данных. Из графика видно, что выполненная реализация метода

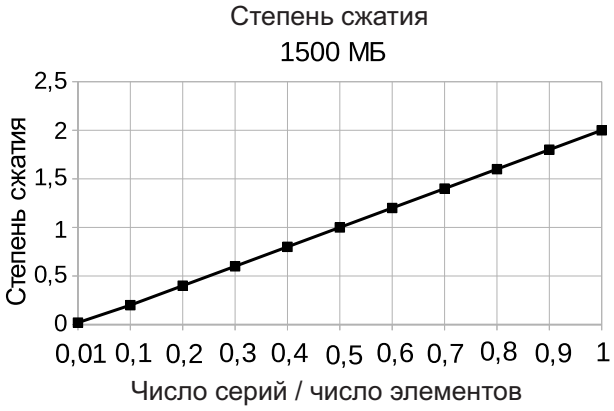

Рис. 7. Степень сжатия методом RLE

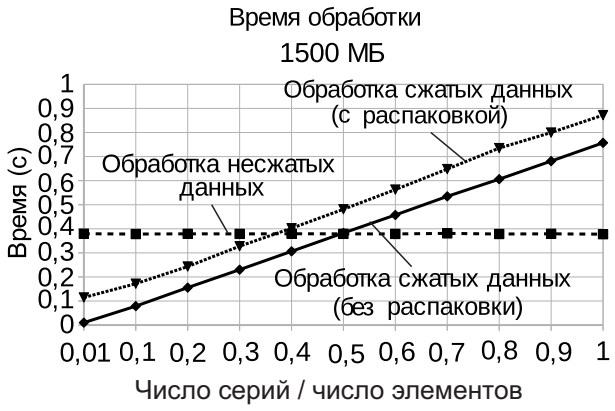

Рис. 8. Время обработки данных, сжатых методом RLE

эффективна тогда, 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 тогда, 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.

3.2. Метод LZSS. 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.

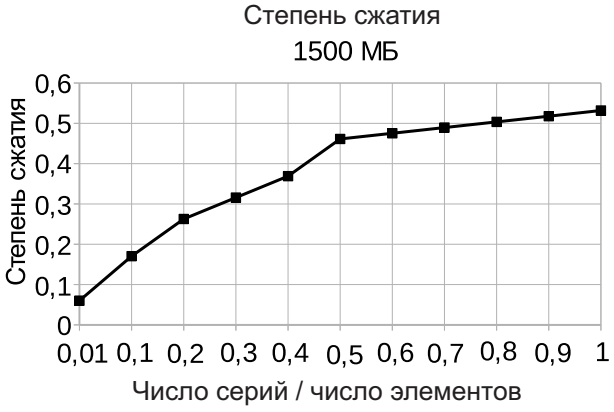

Рис. 9. Степень сжатия методом LZSS

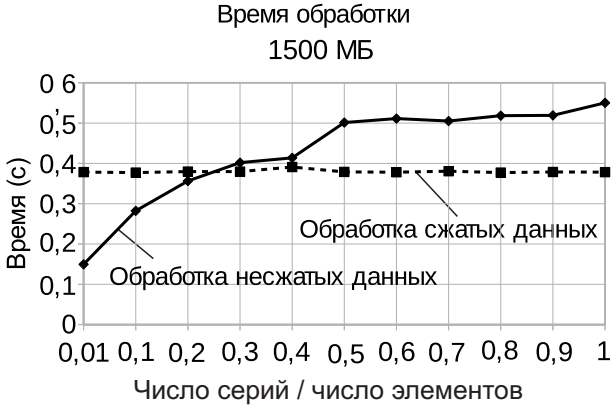

Рис. 10. Время обработки данных, сжатых методом LZSS

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. Следует также 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 рассмотренных 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.

3.3. Null Suppression. 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.

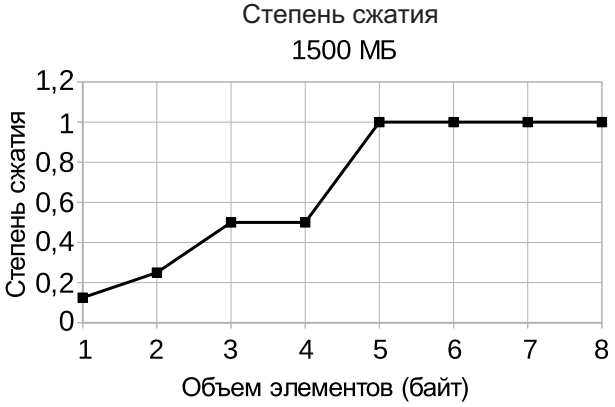

Рис. 11. Степень сжатия методом Null Suppression

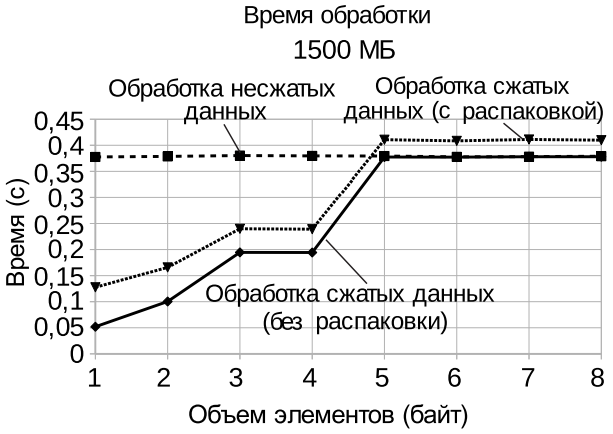

Рис. 12. Время обработки данных, сжатых методом Null Suppression

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 этого метода.

Заключение. В рамках настоящей статьи рассмотрены три метода сжатия данных: Run Length Encoding, Null Suppression и LZSS. 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.

Дальнейшие исследования предполагается проводить в следующих направлениях:

— исследование эффективности использования других методов сжатия;

— исследование эффективности применения комбинаций из нескольких методов сжатия в контексте рассмотренной выше 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”. Статья рекомендована к публикации Программным комитетом Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: многообразие суперкомпьютерных миров” (http://agora.guru.ru/abrau2014).

СПИСОК ЛИТЕРАТУРЫ

1. *Беседин К.Ю., Костенецкий П.С.* Моделирование обработки запросов на гибридных вычислительных системах

с многоядерными сопроцессорами и графическими ускорителями // Программные системы: теория и приложения. 2014. **5**, № 1. 91–110.
2. *Беседин К.Ю., Костенецкий П.С.* Применение многоядерных сопроцессоров в параллельных системах баз данных // Тр. Международной научной конференции "Параллельные вычислительные технологии" (ПаВТ'2013). 1–5 апреля 2013 г., Челябинск. Челябинск: Издательский центр ЮУрГУ, 2013. 583.
3. *Костенецкий П.С., Соколинский Л.Б.* Моделирование иерархических многопроцессорных систем баз данных // Программирование. 2013. **39**, № 1. 13–22.
4. *Костенецкий П.С.* Обработка запросов на кластерных вычислительных системах с многоядерными ускорителями // Вестн. Южно-Уральского гос. ун-та. Серия: Вычислительная математика и информатика. 2012. № 2. 59–67.
5. *Костенецкий П.С., Лепихов А.В., Соколинский Л.Б.* Технологии параллельных систем баз данных для иерархических многопроцессорных сред // Автоматика и телемеханика. 2007. № 5. 112–125.
6. *Abadi D.J., Madden S.R., Ferreira M.C.* Integrating compression and execution in column-oriented database systems // Proc. ACM SIGMOD Int. Conf. on Management of Data. Chicago, USA. June 26–29, 2006. New York: ACM Press, 2006. 671–682.
7. *Abadi D.J., Madden S.R., Hachem N.* Column-stores vs. row-stores: how different are they really? // Proc. ACM SIGMOD Int. Conf. on Management of Data. Vancouver, Canada. June 10–12, 2008. New York: ACM Press, 2008. 967–980.
8. *Binnig C., Hildenbrand S., Färber F.* Dictionary-based order-preserving string compression for main memory column stores // Proc. ACM SIGMOD Int. Conf. on Management of Data Providence. Rhode Island, USA. June 29–July 2, 2009. New York: ACM Press, 2009. 283–296.
9. *Fang W., He B., Luo Q.* Database compression on graphics processors // Proc. 36th Int. Conf. on Very Large Data Bases. Singapore. September 13–17, 2010. Singapore: VLDB Endowment, 2010. 670–680.
10. *Graefe G., Shapiro L.D.* Data compression and database performance // Proc. ACM/IEEE-CS Symp. on Applied Computing. Kansas City, USA. April 3–5, 1991. New York: IEEE Press, 1991. 22–27.
11. *Iyer B.R., Wilhite D.* Data compression support in databases // Proc. 20th Int. Conf. on Very Large Data Bases. Santiago de Chile, Chile. September 12–15, 1994. San Francisco: Morgan Kaufmann Publ., 1994. 695–704.
12. *Jeffers J., Reinders J.* Intel Xeon Phi coprocessor high-performance programming. Waltham: Morgan Kaufmann Publ., 2013.
13. *Kirk D.B., Hwu W.W.* Programming massively parallel processors: a hands-on approach. Waltham: Morgan Kaufmann Publ., 2013.
14. *Kostenetskiy P.S., Sokolinsky L.B.* Analysis of hierarchical multiprocessor database systems // Proc. 2007 Int. Conf. on High Performance Computing, Networking and Communication Systems (HPCNCS-07). Orlando, USA. July 9–12, 2007. Tallahassee: ISRST, 2007. 245–251.
15. *Ng W.K., Ravishankar C.V.* Block-oriented compression techniques for large statistical databases // IEEE Trans. on Knowledge and Data Engineering. 1997. **9**, N 2. 314–328.
16. *Ozsoy A., Swany M.* CULZSS: LZSS lossless data compression on CUDA // Proc. 2011 IEEE Int. Conf. on Cluster Computing. Washington, D.C., USA. September 26–30, 2011. New York: IEEE Press, 2011. 403–416.
17. *Roth M.A., van Horn S.J.* Database compression // ACM SIGMOD Record. 1993. **22**, N 3. 31–39.
18. *Storer J.A., Szymanski T.G.* Data compression via textual substitution // Journal of the ACM. 1982. **29**, N 4. 928–951.
19. *Wu L., Storus M., Cross D.* Cs315a: final project CUDA WUDA SHUDA: CUDA compression project. Stanford: Stanford Univ., 2009.

Поступила в редакцию
18.09.2014

Efficiency Evaluation of Some Compression Methods for Data Transfer between Main Memory and Intel Xeon Phi Coprocessors

P. S. Kostenetskiy[1] **and K. Yu. Besedin**[2]

[1] *South Ural State University, Faculty of Computational Mathematics and Informatics; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Ph. D., Associate Professor, e-mail: kostenetskiy@susu.ru*

[2] *South Ural State University, Faculty of Computational Mathematics and Informatics; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Student, e-mail: besedin.k@gmail.com*

Received September 18, 2014

Abstract: The need to transfer data through a PCI-E (Peripheral Component Interconnect Express) bus is one of the key characteristics of GPU and multicore coprocessors programming, which is considered as a bottleneck for a number of applications. This paper focuses on evaluating the efficiency of data compression for optimizing the data transfer between main memory and Intel Xeon Phi for database applications. Three compression methods are evaluated: LZSS (Lempel–Ziv–Storer–Szymanski), Null Suppression, and RLE (Run-Length Encoding). An implementation of these methods for Intel Xeon Phi coprocessors is described. It is shown experimentally that these compression methods can be used to increase the efficiency of database processing under certain conditions imposed on the data under treatment. It is also shown that, when a compression method allows one to process data without decompression, such a processing procedure can additionally increase the efficiency of this method.

Keywords: database management systems, data compression, Intel Xeon Phi, LZSS compression, RLE compression, Null Suppression.

References

1. K. Yu. Besedin and P. S. Kostenetskiy, "Simulating of Query Processing on Multiprocessor Database Systems with Modern Coprocessors," Program. Sistemy: Teor. Pril. **5** (1), 91–110 (2014).

2. K. Yu. Besedin and P. S. Kostenetskiy, "Application of Multicore Coprocessors in Parallel Database Systems," in *Proc. Int. Conf. on Parallel Computational Technologies, Chelyabinsk, Russia, April 1–5, 2013* (South Ural State Univ., Chelyabinsk, 2013), p. 583.

3. P. S. Kostenetskii and L. B. Sokolinsky, "Simulation of Hierarchical Multiprocessor Database Systems," Programmirovanie **39** (1), 13–22 (2013) [Program. Comput. Soft. **39** (1), 10–24 (2013)].

4. P. S. Kostenetskiy, "Query Processing on Cluster Based Systems with Multicore Accelerators," Vestn. South Ural State Univ. Ser. Vychisl. Mat. Inf., No. 2, 59–67 (2012).

5. P. S. Kostenetskii, A. V. Lepikhov, and L. B. Sokolinskii, "Technologies of Parallel Database Systems for Hierarchical Multiprocessor Environments," Avtomatika i Telemekhanika, No. 5, 112–125 (2007) [Autom. Remote Control **68** (5), 847–859 (2007)].

6. D. J. Abadi, S. R. Madden, and M. C. Ferreira, "Integrating Compression and Execution in Column-Oriented Database Systems," in *Proc. ACM SIGMOD Int. Conf. on Management of Data. Chicago, USA, June 26–29, 2006* (ACM Press, New York, 2006), pp. 671–682.

7. D. J. Abadi, S. R. Madden, and N. Hachem, "Column-Stores vs. Row-Stores: How Different are They Really?," in *Proc. ACM SIGMOD Int. Conf. on Management of Data, Vancouver, Canada, June 10–12, 2008* (ACM Press, New York, 2008), pp. 967–980.

8. C. Binnig, S. Hildenbrand, and F. Färber, "Dictionary-Based Order-Preserving String Compression for Main Memory Column Stores," in *Proc. ACM SIGMOD Int. Conf. on Management of Data Providence, Rhode Island, USA, June 29–July 2, 2009* (ACM Press, New York, 2009), pp, 283–296.

9. W. Fang, B. He, and Q. Luo, "Database Compression on Graphics Processors," in *Proc. 36th Int. Conf. on Very Large Data Bases, Singapore, September 13–17, 2010* (VLDB Endowment, Singapore, 2010), pp. 670–680.

10. G. Graefe and L. D. Shapiro, "Data Compression and Database Performance," in *Proc. ACM/IEEE-CS Symp. on Applied Computing, Kansas City, USA, April 3–5, 1991* (IEEE Press, New York, 1991), pp. 22–27.

11. B. R. Iyer and D. Wilhite, "Data Compression Support in Databases," in *Proc. 20th Int. Conf. on Very Large Data Bases, Santiago de Chile, Chile, September 12–15, 1994* (Morgan Kaufmann Publ., San Francisco, 1994), pp. 695–704.

12. J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High-Performance Programming* (Morgan Kaufmann Publ., Waltham, 2013).

13. D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach* (Morgan Kaufmann Publ., Waltham, 2013).

14. P. S. Kostenetskiy and L. B. Sokolinsky, "Analysis of Hierarchical Multiprocessor Database Systems," in *Proc. 2007 Int. Conf. on High Performance Computing, Networking and Communication Systems (HPCNCS-07), Orlando, USA, July 9–12, 2007* (ISRST, Tallahassee, 2007), pp. 245–251.

15. W. K. Ng and C. V. Ravishankar, "Block-Oriented Compression Techniques for Large Statistical Databases," IEEE Trans. Knowl. Data Eng. **9** (2), 314–328 (1997).

16. A. Ozsoy and M. Swany, "CULZSS: LZSS Lossless Data Compression on CUDA," in *Proc. 2011 IEEE Int. Conf. on Cluster Computing, Washington, D.C., USA, September 26–30, 2011* (IEEE Press, New York, 2011), pp. 403–416.

17. M. A. Roth and S. J. van Horn, "Database Compression," ACM SIGMOD Rec. **22** (3), 31–39 (1993).

18. J. A. Storer and T. G. Szymanski, “Data Compression via Textual Substitution,” J. ACM **29** (4), 928–951 (1982).

19. L. Wu, M. Storus, and D. Cross, *Cs315a: Final Project CUDA WUDA SHUDA: CUDA Compression Project* (Stanford Univ., Stanford, 2009).