Использование сопроцессоров Intel Xeon Phi для выполнения естественного соединения над сжатыми данными*

Е.В. Иванова, Л.Б. Соколинский

ФГБОУ ВПО «ЮУрГУ» (НИУ)

В работе описывается сопроцессор баз данных для высокопроизводительных кластерных вычислительных систем с многоядерными ускорителями, использующий распределенные колоночные индексы с интервальной фрагментацией. Работа сопроцессора рассматривается на примере выполнения операции естественного соединения. Параллельная декомпозиция естественного соединения выполняется на основе использования распределенных колоночных индексов. Предложенный подход позволяет выполнять реляционные операции на кластерных вычислительных системах без массовых обменов данными. Приводятся результаты вычислительных экспериментов с использованием сопроцессоров Intel Xeon Phi, подтверждающие эффективность разработанных методов и алгоритмов.

1. Введение

В настоящее время фактически единственным эффективным решением проблемы хранения и обработки сверхбольших баз данных является использование параллельных систем баз данных, обеспечивающих распределенную обработку запросов на многопроцессорных вычислительных системах с распределенной памятью [1-4]. В последние годы основным способом наращивания производительности процессоров является увеличение количества ядер, а не тактовой частоты, и эта тенденция, вероятно, сохранится [5]. Многоядерные сопроцессоры Intel Xeon Phi значительно опережают традиционные процессоры в производительности по арифметическим операциям и пропускной способности памяти. Последние исследования показывают, что Intel Xeon Phi могут эффективно использоваться для обработки запросов к базам данных [6-8]. При обработке сверхбольших баз данных строковое представление информации, характерное для реляционной модели, оказывается неэффективным при обработке OLAP запросов. В работе [9] была предложена колоночная модель хранения данных, которая на некоторых классах запросов, характерных для хранилищ данных, способна обеспечить повышение производительности на два порядка по сравнению со строковой [10]. Указанная модель была реализована в целом ряде колоночных СУБД. В качестве примера можно привести такие известные СУБД как MonetDB [11] и C-Store [10]. Недостатком этого подхода является то, что в колоночных СУБД не может применяться техника эффективной оптимизации SQL-запросов, хорошо зарекомендовавшая себя в реляционных СУБД. Авторами статьи в работах [12–14] были предложены индексные структуры специального вида, названные *колоночными индексами*, которые позволяют сочетать преимущества строкового и столбового хранения данных. В данной работе описывается новый подход к проектированию СУБД для высокопроизводительных кластерных вычислительных систем с многоядерными ускорителями, который предполагает использование сопроцессора баз данных, позволяющего эффективно выполнять ресурсоемкие части реляционных операций без обращения к исходным таблицам со строковым хранением и без обменов данными между процессами. Это достигается путем создания, поддержания и использования распределенных колоночных индексов, хранящихся в сжатом виде в оперативной памяти сопроцессоров Intel Xeon Phi. Для обозначения реляционных операций в статье используется нотация из монографии [15].

* Работа выполнена при финансовой поддержке Минобрнауки РФ в рамках ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2014—2020 годы» (Госконтракт № 14.574.21.0035).

2. Общая архитектура системы

Система баз данных, использующая колоночные индексы и интервальную фрагментацию, включает в себя сервер баз данных и сопроцессор баз данных (рис. 1). Сервер баз данных реализуется в виде СУБД, в которой введен дополнительный уровень абстракции при выполнении реляционной операции: на первой фазе вычисляются адреса кортежей, из которых строится результат (*таблица предвычислений, ТПВ*); на второй фазе конструируется результирующее отношение путем считывания кортежей по адресам из ТПВ. Для часто повторяющихся ресурсоемких операций администратор создает необходимые колоночные индексы, которые фрагментируются, а затем хранятся и обрабатываются сопроцессором баз данных в оперативной памяти узлов кластерной вычислительная системы с многоядерными ускорителями. Каждый фрагмент колоночного индекса хранится в сжатом виде. На одном вычислительном узле может храниться несколько фрагментов. При необходимости выполнить соответствующую ресурсоемкую операцию в ходе выполнения запроса, вычисление ТПВ выполняется сопроцессором баз данных. Каждый фрагмент колоночного индекса разжимается, обрабатывается и сжимается отдельным ядром многоядерного ускорителя. Балансировка загрузки между ядрами многоядерного ускорителя производится автоматически.

Сопроцессор баз данных включает в себя подсистему «Исполнитель», реализующую выполнение ресурсоемкой части реляционной операции над фрагментами, и подсистему «Координатор», реализующую рассылку запроса от СУБД между подсистемами «Исполнитель», получение от подсистем «Исполнитель» частичных результатов, сбор ТПВ и отправку ТПВ на сервер.

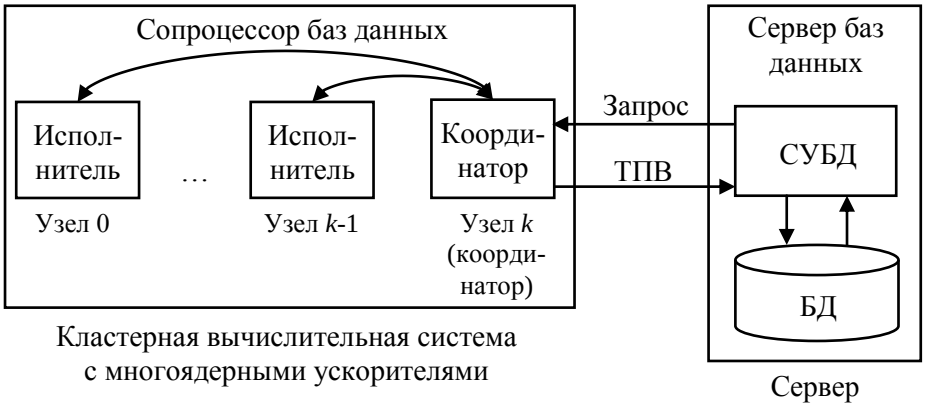

Рис. 1. Архитектура СУБД с сопроцессором баз данных.

3. Колоночный индекс и интервальная фрагментация

Колоночный индекс $I_{R.B}$ для атрибута *B* таблицы *R* представляет собой таблицу из двух колонок с именами *A* и *B* (см. рис. 2). Количество строк в колоночном индексе совпадает с количеством строк в индексируемой таблице. Колонка *B* индекса $I_{R.B}$ включает в себя все значения колонки *B* таблицы *R* (с учетом повторяющихся значений), отсортированных в порядке возрастания. Каждая строка *x* индекса $I_{R.B}$ содержит в колонке *A* *суррогатный ключ* (адрес кортежа, представляющий собой идентификатор целочисленного типа, однозначно определяющий кортеж) строки *r* в таблице *R*, имеющей такое же значение в колонке *B*, что и строка *x*.

Пусть на домене $\mathfrak{D}_B$ атрибута *B* задано отношение линейного порядка. Пусть также задано разбиение множества $\mathfrak{D}_B$ на $k > 0$ непересекающихся интервалов, которые называются *доменными интервалами* и вместе составляют все множество $\mathfrak{D}_B$. Колоночный индекс распределяется по узлам вычислительной системы в соответствие с *функцией фрагментации* $\varphi_{I_{R.B}}: I_{R.B} \to \{0, \dots, k-1\}$, которая ставит в соответствие каждому кортежу индекса некоторый узел вычислительной системы. В *i*-тый фрагмент $I_{R.B}^{i}$ попадают кортежи, у которых значение атрибута *B* принадлежит *i*-тому доменному интервалу. Будем называть фрагментацию, построенную таким

образом, *интервальной*. Количество фрагментов k будем называть *степенью фрагментации*. Если атрибуты различных колоночных индексов имеют один и тот же домен, то их

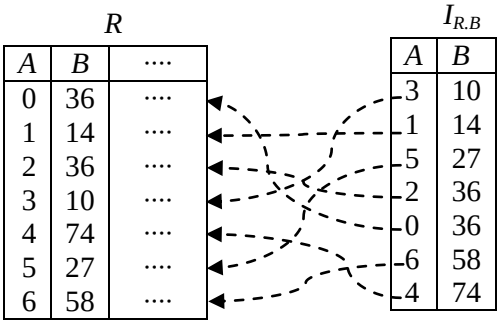

Рис. 2. Колоночный индекс.

одинаковые значения в результате фрагментации гарантированно попадут на один и тот же узел вычислительной системы. Пример использования интервальной фрагментации степени 2 изображен на рис. 3. В качестве функции фрагментации для колоночного индекса $I_{R.B}$ используется функция

$$\varphi_{I_{R.B}}(x)=\begin{cases}0, & при \quad x.B<30\\ 1, & при \quad x.B\geq 30\end{cases}.$$

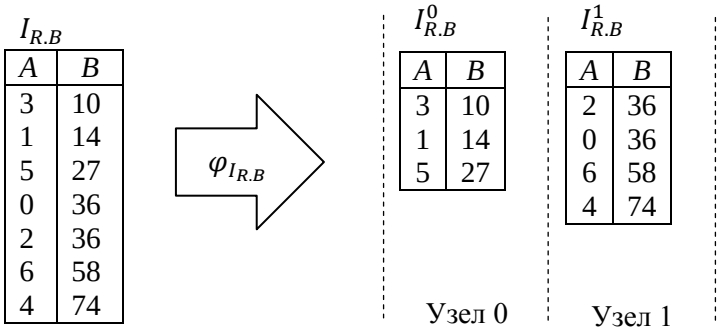

Рис. 3. Интервальная фрагментация.

4. Декомпозиция операции естественного соединения

Пусть заданы два отношения

$$R(A, B_1, \dots, B_u, C_1, \dots, C_v)$$

и

$$S(A, B_1, \dots, B_u, D_1, \dots, D_w).$$

Пусть имеется два набора колоночных индексов по атрибутам $B_1, \dots, B_u$:

$$I_{R.B_1}, \dots, I_{R.B_u};$$
$$I_{S.B_1}, \dots, I_{S.B_u}.$$

Пусть для всех этих индексов задана интервальная фрагментация степени k, распределяющая индексы по k-узлам вычислительной системы. На каждом i-ом узле ($i = 0, \dots, k-1$) выполняется попарное соединение фрагментов $I^i_{R.B_j}$ и $I^i_{S.B_j}$ по B_j для всех $j = 1, \dots, u$, в результат выносятся только суррогатные ключи фрагментов. Результатом операции является таблица P^i_j из двух колонок A_R (суррогатные ключи фрагмента $I^i_{R.B_j}$) и A_S (суррогатные ключи фрагмента $I^i_{S.B_j}$). Данные вычисления могут выполняться независимо на k различных узлах без обменов данными. Затем каждый из k узлов пересылает полученные таблицы P^i_j на узел-координатор, где таблицы для одного и того же атрибута B_j объединяются в одну. Таким образом, получаем u таблиц P_j.

Следующим шагом в таблицах P_j необходимо найти те кортежи, которые присутствуют во всех таблицах одновременно. Полученное множество является ТПВ.

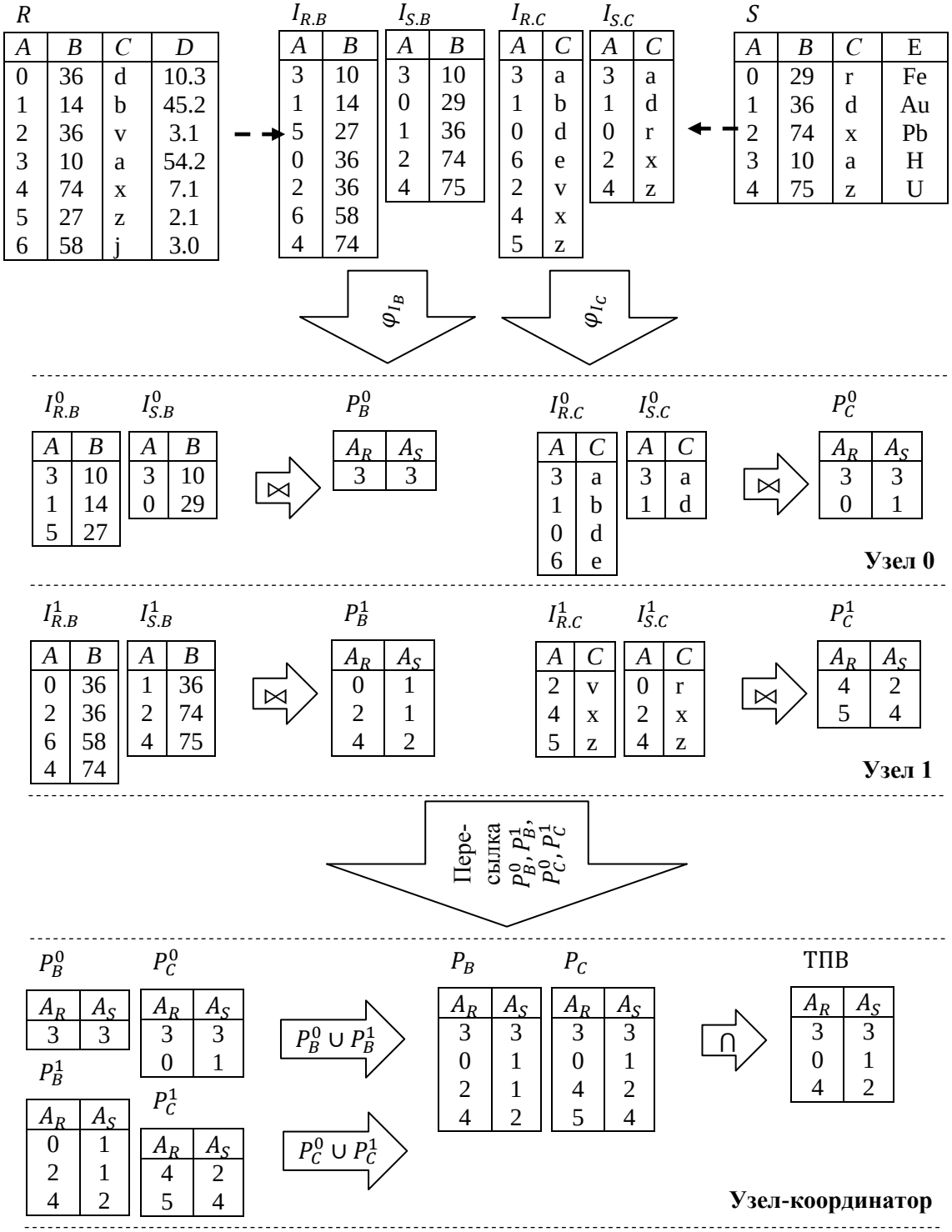

Рис. 4. Получение ТПВ для операции естественного соединения $R \bowtie S$ с использованием колоночных индексов.

На заключительном шаге происходит конструирование кортежей результата с помощью ТПВ. Для каждой пары адресов из ТПВ производится поиск по суррогатному ключу A_R кортежа r из R и по суррогатному ключу A_S – кортежа s из S. Множество кортежей вида $q = (r.B_1, \dots, r.B_u, r.C_1, \dots, r.C_v, s.D_1, \dots, s.D_w)$ является результатом операции естественного соединения отношений R и S. Пример получения ТПВ для операции естественного соединения

двух отношений R и S по двум атрибутам B и C изображен на рис. 4. Используется интервальная фрагментация степени 2. Для колоночных индексов $I_{R.B}$ и $I_{S.B}$, $I_{R.C}$ и $I_{S.C}$ используются следующие функции фрагментации:

$$\varphi_{I_B}(x) = \varphi_{I_{R.B}}(x) = \varphi_{I_{S.B}}(x) = \begin{cases} 0, & при \quad x.B < 30 \\ 1, & при \quad x.B \geq 30 \end{cases},$$

$$\varphi_{I_C}(x) = \varphi_{I_{R.C}}(x) = \varphi_{I_{S.C}}(x) = \begin{cases} 0, & при \quad x.C \in [\text{a..m}) \\ 1, & при \quad x.C \in [\text{m..z}] \end{cases}.$$

На рис. 5 представлено конструирование кортежей результата с использованием полученной ТПВ.

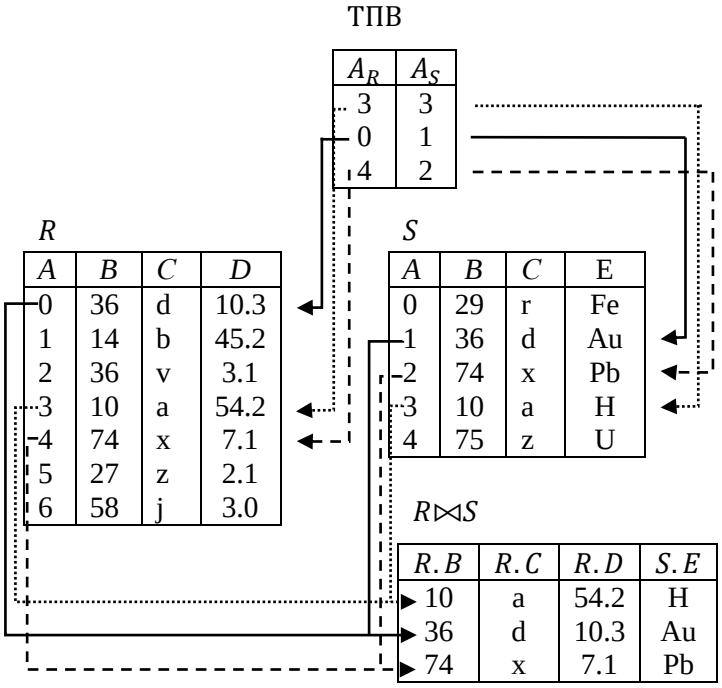

Рис. 5. Конструирование кортежей результата с использованием ТПВ на сервере баз данных.

5. Вычислительные эксперименты

Для подтверждения эффективности выполнения реляционных операции с использованием распределенных колоночных индексов создан прототип колоночного сопроцессора баз данных, с помощью которого были проведены вычислительные эксперименты. Исходный код прототипа доступен в репозитории GitHub [16].

В ходе экспериментов генерировалась тестовая база данных, состоящая из двух отношений R и S с общим целочисленным атрибутом B. Атрибут B отношения R является первичным ключом, атрибут B отношения S – внешним ключом. Размеры отношений были следующие: 600 000 кортежей в R и 60 000 000 кортежей в S. Генерация атрибута B в отношении S производилась двумя способами. Первый способ предполагал использование равномерного (uniform) распределения значений в столбце $S.B$, что означает одинаковое количество кортежей в каждом фрагменте. При втором способе столбец $S.B$ генерировался с использованием неравномерного распределения (правила «80-20», «65-20», «45-20») [17].

Для генерации неравномерного распределения была использована вероятностная модель. В соответствии с этой моделью коэффициент перекоса Θ, $(0 \leq \Theta \leq 1)$ задает распределение, при котором каждому различному значению в $S.B$ назначается некоторый весовой коэффициент $p_i, i = 1 \ldots N$, определяемый формулой

$$p_i = \frac{1}{i^{\Theta} \cdot H_N^{(\Theta)}}, \quad \sum_{i=1}^{N} p_i = 1,$$

где N – количество различных значений атрибута *S.B* и $H_N^{(s)} = 1^{-s} + 2^{-s} + \cdots + N^{-s}$ *N*-е гармоническое число порядка *s*. В случае $\Theta = 0$ распределение весовых коэффициентов соответствует равномерному распределению. При $\Theta = 0.86$ распределение соответствует правилу «80-20», в соответствии с которым, 80% кортежей отношения будет храниться в 20% фрагментов. При $\Theta = 0.73$ распределение соответствует правилу «65-20». При $\Theta = 0.5$ распределение соответствует правилу «45-20».

Для столбцов *R.B* и *S.B* были созданы колоночные индексы $I_{R.B}$ и $I_{S.B}$ соответственно. Индексы $I_{R.B}$ и $I_{S.B}$ фрагментировались на основе интервального принципа. Количество фрагментных интервалов было кратно 600 000 и являлось параметром экспериментов. Каждый фрагмент колоночных индексов сжимался с помощью библиотеки Zlib [18].

В ходе экспериментов выполнялось соединение колоночных индексов $I_{R.B}$ и $I_{S.B}$ по атрибуту *B* с созданием ТПВ. Операция соединения производилась с использованием алгоритма соединения слиянием (Merge Join, MJ). Особенностью алгоритма MJ является то, что соединение осуществляется за одно сканирование каждой из входных таблиц, что существенно уменьшает количество операций по сравнению с алгоритмом вложенных циклов.

Эксперименты были проведены на узле с многоядерным ускорителем Intel Xeon Phi SE10X кластерной вычислительной системы «Торнадо ЮУрГУ», установленной в Южно-Уральском государственном университете. Многоядерный ускоритель Intel Xeon Phi имеет 61 вычислительное ядро и 8 Гб оперативной памяти.

Операция соединения выполнялась на 1, 2 и 4 нитях, запущенных на одном ядре Intel Xeon Phi (рис. 6). Результаты этого эксперимента показывают, что наибольшее ускорение достигается при запуске на каждом ядре только одной нити.

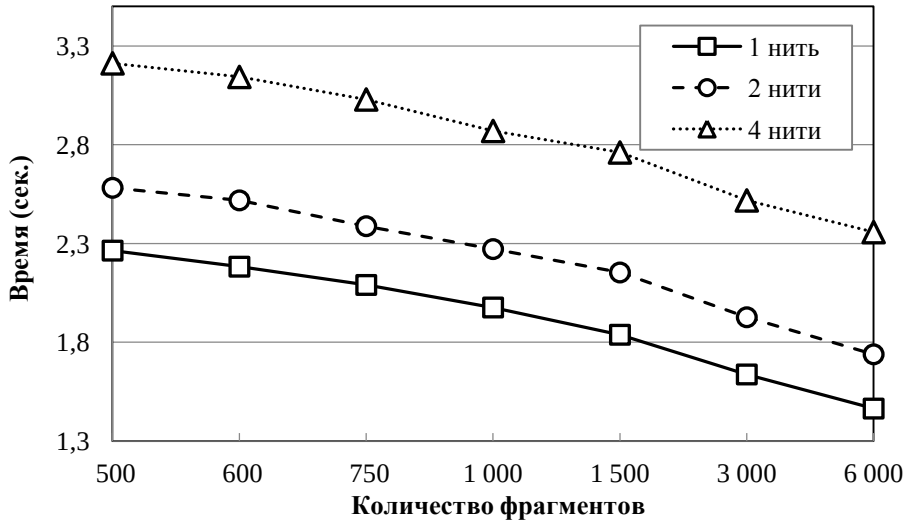

Рис. 6. Зависимость времени выполнения соединения от количества фрагментов.

Также нами были исследованы ускорение и расширяемость при выполнении соединения колоночных индексов на 15, 30, 45 и 60 ядрах Intel Xeon Phi. Результаты представлены на рис. 7. Ускорение считалось как отношение времени выполнения соединения на заданном количестве ядер к времени выполнения этого же соединения на 15 ядрах (база данных не менялась). При исследовании расширяемости вычислялись те же отношения времен, но при этом размер базы данных увеличивался пропорционально увеличению количества ядер. Эксперименты показали, что многоядерный ускоритель Intel Xeon Phi демонстрирует линейные ускорение и масштабируемость. Это означает, что при выполнении описанной операции соединения с использованием колоночных индексов в структуре Intel Xeon Phi не возникает узких мест. Это согласуется с высоким процентом утилизации процессорных ядер Intel Xeon Phi при выполнении соединения

сжатых фрагментированных колоночных индексов (измерения с помощью команды Linux TOP показали утилизацию 98%).

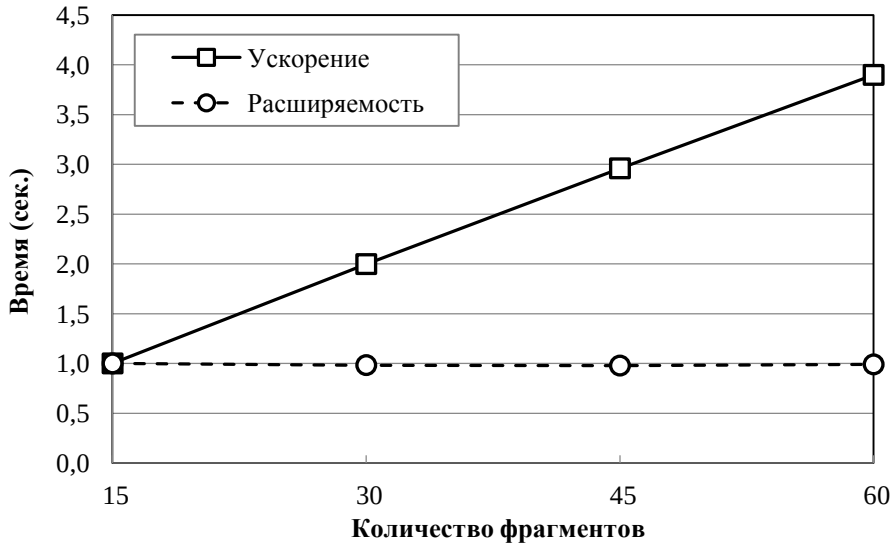

Рис. 7. Ускорение и расширяемость при увеличении количества используемых ядер Xeon Phi.

В следующем эксперименте исследовалась проблема дисбаланса загрузки ядер Intel Xeon Phi при наличии перекосов в распределении значений по фрагментам. Результаты представлены на рис. 8. Эксперимент показал, что деление колоночных индексов на достаточно большое количество фрагментов позволяет эффективно сбалансировать загрузку процессорных ядер даже при наличии большого перекоса в распределении значений в колоночных индексах.

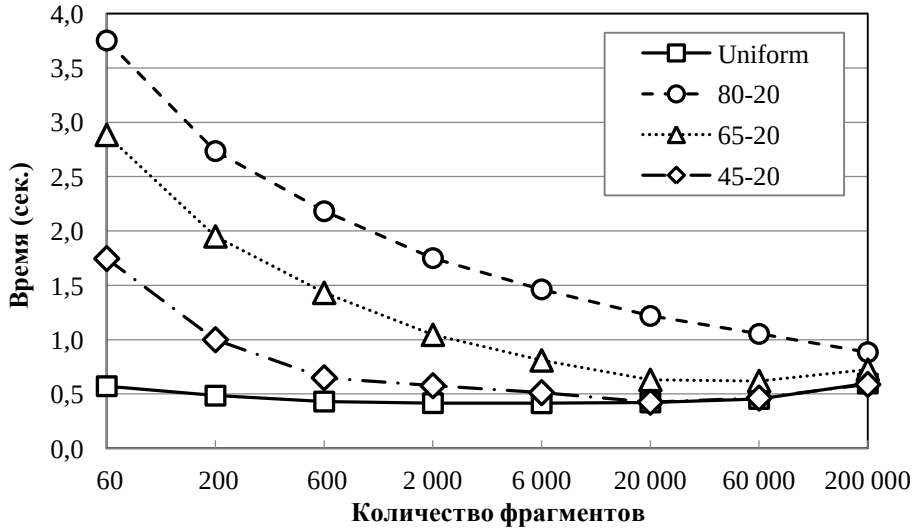

Рис. 8. Влияние количества фрагментов на баланс загрузки процессорных ядер Xeon Phi.

Проведенные исследования показывают, что предложенный подход на основе распределенных колоночных индексов позволяет выполнять ресурсоемкую операцию естественного соединения двух таблиц, содержащих 600 000 и 60 000 000 кортежей, менее чем за одну секунду на одном многоядерном ускорителе Intel Xeon Phi. Поскольку предложенная технология позволяет выполнять соединение при отсутствии обменов данных между процессорами, имеются все основания ожидать ускорение близкое к линейному на кластерных вычислительных системах с миллионами процессорных узлов, оснащенных многоядерными ускорителями.

6. Заключение

В статье было представлено описание колоночного сопроцессора баз данных для высокопроизводительных кластерных вычислительных систем с многоядерными ускорителями. Колоночный сопроцессор использует распределенные колоночные индексы с интервальной фрагментацией. Рассмотрена работа колоночного сопроцессора на примере параллельной декомпозиции операции естественного соединения. В рамках проведенных экспериментов было исследовано время выполнения операции естественного соединения двух отношений *R* и *S* с использованием колоночного сопроцессора. Исследовались зависимость времени выполнения естественного соединения от количества фрагментов при запуске 1, 2 и 4 нитей на каждом ядре Intel Xeon Phi и влияние количества фрагментов на баланс загрузки процессорных ядер Xeon Phi при равномерном и неравномерном распределении значений атрибута *S.B*. Проведенные исследования показывают, что предложенный подход на основе использования колоночного сопроцессора баз данных позволяет эффективно выполнять ресурсоемкую операцию естественного соединения двух таблиц на вычислительных кластерах с многоядерными ускорителями.

Литература

1. Соколинский Л.Б. Параллельные системы баз данных. М.: Издательство Московского университета, 2013. 184 с.
2. Sokolinsky L.B. Design and Evaluation of Database Multiprocessor Architecture with High Data Availability // Proceedings of the 12th International workshop on database and expert systems applications. IEEE Computer Society, 2001. P. 115–120.
3. Sokolinsky L.B. Operating System Support for a Parallel DBMS with an Hierarchical Shared-Nothing Architecture // Advances in Databases and Information Systems, Third East European Conference, ADBIS'99, Maribor, Slovenia, September 13-16, 1999, Proceedings of Short Papers. Maribor: Institute of Informatics. 1999. P. 38-45.
4. Соколинский Л.Б., Цымблер М.Л. Принципы реализации системы управления файлами в параллельной СУБД Омега для МВС-100 // Вестник Челябинского университета. Сер. 3. Математика, механика. 1999. No.2(5). С. 78-96.
5. Fang J., Varbanescu A.L., Sips H. Sesame: A User-Transparent Optimizing Framework for Many-Core Processors // Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid2013), May 13–16, 2013, Delft, Netherlands. IEEE, 2013. P. 70–73.
6. Scherger M. Design of an In-Memory Database Engine Using Intel Xeon Phi Coprocessors // Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'14), July 21–24, 2014, Las Vegas, USA. CSREA Press, 2014. P. 21–27.
7. Breß S., Beier F., Rauhe H., et al. Efficient Co-Processor Utilization in Database Query Processing // Information Systems. 2013. Vol. 38, No. 8. P. 1084–1096.
8. Беседин К.Ю., Костенецкий П.С. Применения многоядерных сопроцессоров в параллельных системах баз данных // Параллельные вычислительные технологии (ПаВТ'2013): труды международной научной конференции. Челябинск: Издательский центр ЮУрГУ, 2013. С. 583.
9. Khoshafian S., Copeland G., Jagodis T., Boral H., Valduriez P. A query processing strategy for the decomposed storage model // ICDE. 1987. P. 636–643.
10. Stonebraker M., Abadi D. J., Batkin A., Chen X., Cherniack M., Ferreira M., Lau E., Lin A., Madden S. R., O'Neil E. J., O'Neil P. E., Rasin A., Tran N., Zdonik S. B. C-Store: A Column-Oriented DBMS // VLDB. 2005. P. 553–564.
11. Boncz P., Zukowski M., Nes N. MonetDB/X100: Hyper-pipelining query execution // CIDR. 2005. P. 225-237.

12. Иванова Е.В., Соколинский Л.Б. Использование распределенных колоночных индексов для выполнения запросов к сверхбольшим базам данных // Параллельные вычислительные технологии (ПАВТ'2014). Труды международной научной конференции. Челябинск: Издательский центр ЮУрГУ, 2014. С. 270–275.
13. Иванова Е.В. Использование распределенных колоночных хеш-индексов для обработки запросов к сверхбольшим базам данных // Научный сервис в сети Интернет: многообразие суперкомпьютерных миров. Труды Международной суперкомпьютерной конференции. М.: Изд-во МГУ, 2014. С. 102–104.
14. Иванова Е.В., Соколинский Л.Б. Декомпозиция операций пересечения и соединения на основе доменно-интервальной фрагментации колоночных индексов // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. Т. 4. № 1. С. 44-56.
15. Гарсиа-Молина Г., Ульман Дж., Уидом Дж. Системы баз данных. Полный курс. М.: Издательский дом «Вильямс». 2004. 1088 с.
16. Прототип сопроцессора баз данных для распределенных колоночных индексов. URL: https://github.com/elena-ivanova/colomnindices/ (дата обращения: 11.06.2015)
17. Gray J., Sundaresan P., Englert S., Baclawski K., Weinberger P.J.: Quickly Generating Billion-record Synthetic Databases. In: SIGMOD'94, P. 243-252.
18. Roelofs G., Gailly J., Adler M. Zlib. URL: http://www.zlib.net/ (дата обращения: 11.06.2015)