МОДЕЛИРОВАНИЕ ОБРАБОТКИ ЗАПРОСОВ НА ГИБРИДНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ С МНОГОЯДЕРНЫМИ СОПРОЦЕССОРАМИ И ГРАФИЧЕСКИМИ УСКОРИТЕЛЯМИ

К.Ю. Беседин, П.С. Костенецкий

Введение

Использование гибридных вычислительных систем, построенных на базе многоядерных сопроцессоров и графических ускорителей, является одним из актуальных на сегодняшний день направлений исследований. В свою очередь, системы баз данных являются одной из областей, где подобные вычислительные системы могут быть успешно применены[5]. В то же время целый ряд технических особенностей графических ускорителей делают такое использование довольно нетривиальной задачей, требующей адаптации существующих либо разработки новых алгоритмов и архитектурных решений. Цель данной работы заключается в оценке эффективности применения графических ускорителей и многоядерных сопроцессоров в параллельных системах баз данных.

На данный момент уже существуют работы, посвященные эффективной реализации специфичных для СУБД алгоритмов с использованием ГПУ. Так, работы [13, 14] описывают алгоритм работы с деревом индекса, учитывающий архитектурные особенности современных ГПУ и ЦПУ, который может быть использован для организации индекса. Работа [19] предлагает метод ускорения операций над индексом, эффективно использующий большое число вычислительных ядер ГПУ. Работы [16, 15] описывают алгоритмы сортировки [9] описывает реализацию операции JOIN. В работе [2] рассматриваются реализации запросов SELECT и JOIN. Также, существуют работы, которые посвящены поиску архитектурных решений, позволяющих наиболее эффективно использовать особенности графических ускорителей. В работе [8] описывается такое совместное использование ГПУ и ЦПУ при обработке запросов, при котором запрос выполняется на ГПУ в тех случаях, когда стоимость выполнения запроса на нем ниже, чем на центральном процессоре (стоимость выполнения запроса вычисляется динамически на основе плана запроса). Работа [10] рассматривает группировку транзакций с целью их последующего выполнения на графическом ускорителе. Работа [11] предлагает использование ГПУ для оптимизации запросов. В работе [6] рассмотрена разработка прототипа СУБД, хранящей свои данные в памяти графического ускорителя. В некоторых работах проведена адаптация существующих СУБД для использования графических ускорителей и оценка эффективности такой адаптации. Так, в работе [4]поддержка вычислений с использованием ГПУ была интегрирована в СУБД Oracle 9, в работе [18] поддержка графических ускорителей была добавлена в WattDB [7], а в работе [3] использовалась SQLite. Еще одной перспективной многоядерной архитектурой является разрабатываемая компанией Intel архитектура Intel MIC. Количество работ, посвященных этой архитектуре меньше чем количество работ, посвященных ГПУ. Это можно объяснить тем, что ускорители, построенные на основе этой архитектуры лишь недавно были представлены широкой публике [12]. Однако, уже существует несколько работ, посвященных использованию Intel MIC в базах данных. Так, в работе [17] было проведено сравнение реализаций алгоритма поразрядной сортировки (Radix Sort) для Intel Core i7, NVIDIA GTX 280 и Knights Ferry. Согласно результатам исследования, производительность Intel MIC оказалась в 2.2 раза выше, чем производительность ЦПУ и в 1.7 раз выше, чем производительность ГПУ. В работе [13], помимо ЦПУ и ГПУ подробно рассматривается реализация предложенного алгоритма поиска в дереве индекса на Intel MIC (Knights Ferry). Для маленьких деревьев (64 тыс. ключей) MIC показал в 2.4 раза более высокую производительность, чем ЦПУ и в 4.4 раза чем ГПУ. Для больших деревьев (16 млн. ключей) производительность MIC оказалась в 3 раза выше, чем ЦПУ и в 1.8 раз выше, чем ГПУ.

В подавляющем большинстве перечисленных выше работах рассматривается использование одного вычислительного узла с многоядерным сопроцессором либо графическим ускорителем. Работ, посвященных использованию ГПУ и MIC в параллельных системах баз данных еще нет. В связи с этим было принято решение разработать эмулятор параллельной СУБД, использующий вычислительные кластеры с гибридными вычислительными узлами, и при помощи данного эмулятора оценить производительность подобных вычислительных систем на приложениях баз данных.

В рамках данной работы был разработан и спроектирован эмулятор параллельной СУБД, позволяющий использовать графические ускорители и NVIDIA и многоядерные сопроцессоры Intel Xeon Phi при обработке реляционных запросов Select и Join. Эмулятор написан на языке Си и использует технологии OpenMP, MPI и NVIDIA CUDA. Реализация алгоритмов была выполнена в трех вариантах с учетом архитектуры используемых сопроцессоров. Для достижения высокой степени параллелизма при обработке запроса используется фрагментация отношений по вычислительным узлам. Выполняемые запросы делятся на подзапросы, поровну распределяемые между рабочими процессами (параллельными агентами). Для коммуникации процессов между собой используется интерфейс MPI.

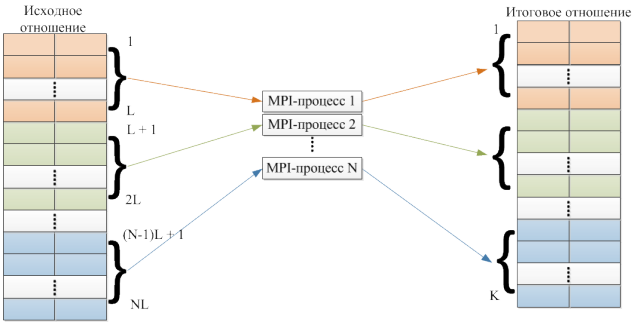

Рис. 1. Схема выполнения запроса ЦПУ SELECT

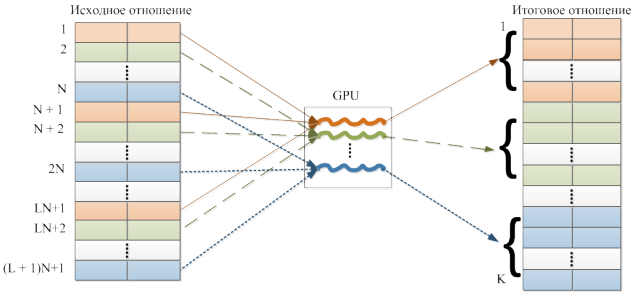

Рис. 2. Схема выполнения запроса ГПУ SELECT

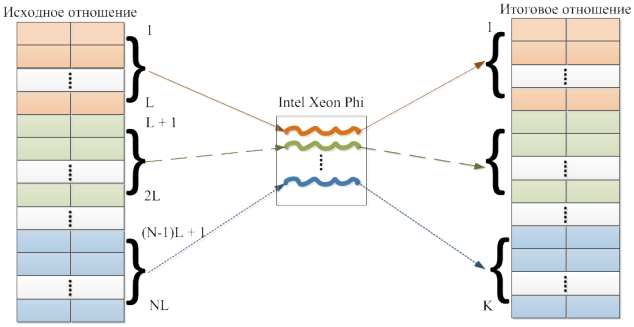

Рис. 3. Схема выполнения запроса Manycore SELECT

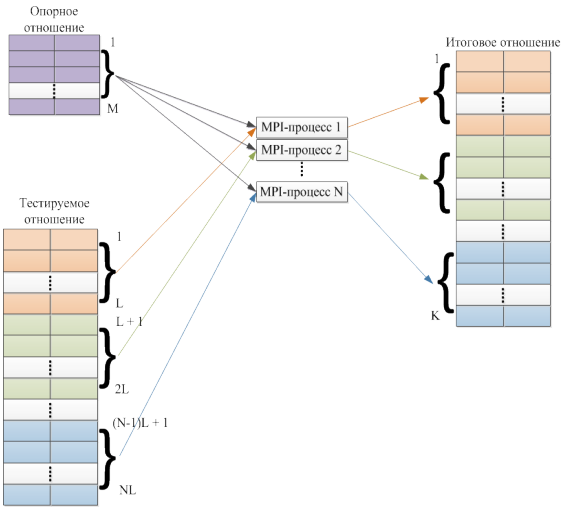

Рис. 4. Схема выполнения запроса ЦПУ JOIN

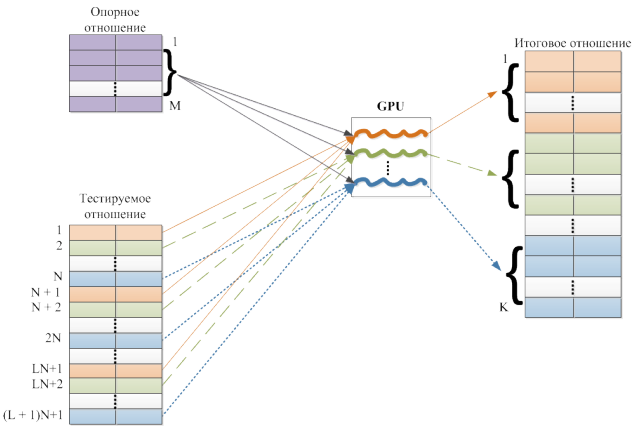

Рис. 5. Схема выполнения запроса ГПУ JOIN

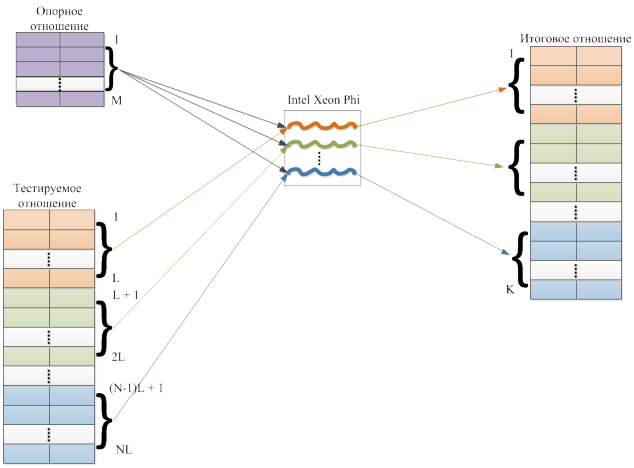

Рис. 6. Схема выполнения Manycore JOIN

Методы организации запросов к базе данных

Общий алгоритм выполнения запроса SELECT состоит из следующих шагов:

1. загрузить отношение в основную память;
2. распределить отношение между рабочими процессами;
3. выполнить выборку;
4. собрать результаты выборок и соединить их в итоговое отношение.

Алгоритм выполнения выборки состоит в просмотре всех кортежей отношения и проверке их на соответствие заданному предикату. Выборка производится в два прохода: во время первого прохода определяется количество отбираемых кортежей, а во время второго прохода заполняется итоговое отношение.

Это позволяет избежать использования сложных структур данных и связанных с ними накладных расходов. Конкретная реализация, однако, несколько отличается в зависимости от версии эмулятора.

Алгоритм ЦПУ SELECT последовательно обрабатывает каждый кортеж полученного фрагмента исходного отношения, проверяя для него условие выборки. Схема выполнения запроса SELECT представлена на рис. 1. Ниже представлен алгоритм выполнения ЦПУ SELECT:

1. последовательно просканировав фрагмент отношения, определить размер результата выборки;
2. выделить память для хранения результата выборки;
3. последовательно просканировав фрагмент отношения, сформировать результат выборки.

Алгоритм ГПУ SELECT. В данном алгоритме кортежи фрагмента исходного отношения распределяются по потокам CUDA так, чтобы был возможен соединенный (coalesced) доступ к глобальной памяти. Для каждого обрабатываемого кортежа проверяется условие выборки. Схема выполнения ГПУ SELECT изображена на рис. 2. Ниже представлен алгоритм выполнения ГПУ SELECT:

1. определить число потоков в блоке и число используемых блоков;
2. подготовить фрагмент отношения в памяти ускорителя;
3. использовать ускоритель для определения размера результата выборки для каждого потока CUDA;
4. вычислить общий размер результата выборки;
5. для каждого потока CUDA вычислить смещения в результате выборки для вставки кортежей;
6. подготовить память для хранения результатов выборки на хосте и на ГПУ;
7. использовать ГПУ для формирования результата выборки;
8. скопировать результат выборки в память хоста;
9. Освободить используемые ресурсы ускорителя

Алгоритм Manycore SELECT использует для выполнения сопроцессоры Intel Xeon Phi. Кортежи фрагмента исходного отношения автоматически распределяются по потокам OpenMP. Схема выполнения Manycore SELECT изображена на рис. 3. Ниже представлен алгоритм Manycore SELECT:

1. подготовить фрагмент отношения в памяти сопроцессора;
2. используя сопроцессор, определить размер результата выборки для каждого потока OpenMP;
3. вычислить общий размер результата выборки;
4. для каждого потока OpenMP вычислить смещения в результате выборки для вставки кортежей;
5. сформировать выборку на сопроцессоре;
6. скопировать результат в память хоста;
7. освободить ресурсы сопроцессора.

Общий алгоритм выполнения запроса JOIN состоит из следующих шагов:

1. загрузить отношения в основную память;
2. отправить каждому рабочему процессу копию опорного отношения;
3. распределить тестируемое отношение между рабочими процессами;
4. выполнить соединение;
5. собрать результаты соединений и соединить их в итоговое отношение.

Для выполнения выборки используется алгоритм вложенных циклов [1]. Соединение производится в два прохода: во время первого прохода определяется количество соединенных кортежей, а во время второго прохода заполняется итоговое отношение. Это позволяет избежать использования сложных структур данных и связанных с ними накладных.

Алгоритм ЦПУ JOIN последовательно проверяет каждый кортеж опорного отношения и каждый кортеж тестируемого отношение на возможность соединения. Схема выполнения ЦПУ JOIN изображена на рис. 4. Ниже представлен алгоритм выполнения ЦПУ JOIN:

1. последовательно выполнив алгоритм вложенных циклов, определить размер результата соединения;
2. выделить память для хранения результата соединения;
3. последовательно выполнив алгоритм вложенных циклов, заполнить результат соединения.

Алгоритм ГПУ JOIN использует графический процессор для вычисления размера итогового отношения и непосредственного выполнения выборки. Кортежи отношения делятся между потоками CUDA так, чтобы воспользоваться объединенным (coalesced) доступом к памяти. Чтобы уменьшить число обращений к глобальной памяти ускорителя интенсивно используется разделяемая память блоков CUDA. На рис. 5 изображена схема выполнения JOIN для ГПУ, а ниже представлен его алгоритм:

1. определить число потоков в блоке и число используемых блоков;
2. подготовить опорное отношение в памяти ускорителя;
3. подготовить фрагмент тестируемого отношения в памяти ускорителя;
4. использовать ускоритель для определения размера результата соединения для каждого потока CUDA;
5. вычислить общий размер результата соединения;
6. для кажлого потока CUDA вычислить смещения в результате соединения для вставки кортежей;
7. подготовить память для хранения результатов соединения на хосте и на ГПУ;
8. использовать ГПУ для формирования результата соединения;
9. скопировать результат в память хоста;

10. освободить используемые ресурсы ускорителя.

Алгоритм Manycore JOIN. Версия алгоритма JOIN для Intel Xeon Phi использует сопроцессор в offload-режиме для выполнения описанных выше проходов. Для распределения нагрузки по ядрам сопроцессора итераций циклов, используется технология OpenMP. Схема выполнения Manycore JOIN изображена на рис. 6. Ниже представлен алгоритм Manycore JOIN:

1. подготовить опорное отношение в памяти сопроцессора;
2. подготовить фрагмент тестируемого отношения в памяти сопроцессора;
3. используя сопроцессор, определить размер результата соединения для каждого потока OpenMP;
4. вычислить общий размер результата выборки;
5. для каждого потока OpenMP вычислить смещения в результате соединения для вставки кортежей;
6. сформировать соединение на сопроцессоре;
7. скопировать результат в память хоста;
8. освободить ресурсы сопроцессора.

Вычислительные эксперименты

Разработанный эмулятор параллельной СУБД был запущен на суперкомпьютере «Торнадо ЮУрГУ», для проведения экспериментов с центральными процессорами Intel Xeon и с многоядерным ускорителем Intel Xeon Phi и на вычислительном кластере ННГУ, для проведения экспериментов на графических ускорителях NVIDIA Tesla. Характеристики узлов данных вычислительных систем приведены в табл.1, 2.

Таблица 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
Число вычислительных узлов с сопроцессорами		192

Таблица 2. Вычислительный кластер ННГУ

Вычислительный узел	Процессор	Intel Xeon L5630 2.13 GHz
	Объем ОЗУ	24 GB
	Число процессоров	2
	Графические ускорители (ГПУ)	NVIDIA Tesla X2070
	Количество ГПУ	2
Число вычислительных узлов		16

Исследование эффективности аппаратных архитектур

Моделирование запроса SELECT. Моделировался простейший вариант запроса SELECT:

SELECT: SELECT * FROM Relation WHERE Attribute = Value;

Для тестирования производительности было использовано отношение, состоящее из двух целочисленных атрибутов и содержащее 369098752 кортежей. Таким образом, примерный размер отношения – 5.5GB.

Производительность запроса ЦПУ SELECT тестировалась на вычислительном узле суперкомпьютера «Торнадо ЮУрГУ». Во время тестирования число MPI-процессов эмулятора варьировалось от 1 до 12.

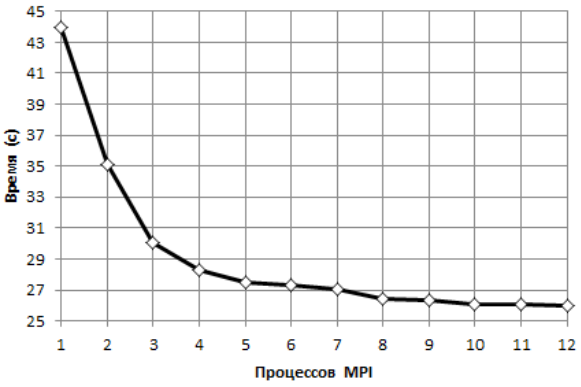

Рис. 7. Время выполнения алгоритма *ЦПУ SELECT* на одном вычислительном узле

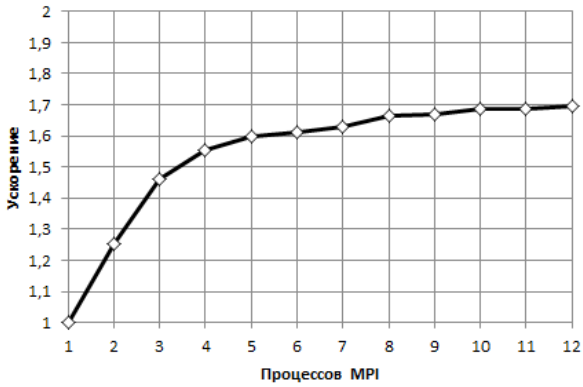

Рис. 8. Ускорение выполнения алгоритма *ЦПУ SELECT* на одном вычислительном узле

На рис. 7 представлена зависимость времени выполнения запроса от числа используемых процессов MPI. На рис. 8 изображено полученное ускорение. Малое ускорение объясняется большими накладными расходами при передаче данных по сети.

Производительность запроса ГПУ SELECT тестировалась на вычислительном узле кластера ННГУ. Во время тестирования число потоков CUDA, используемых эмулятором варьировалось от 512 до 6656. Графики времени выполнения и ускорения представлены на рис. 9, и рис. 10 соответственно. Малое снижение времени и ускорение связано с тем, что помимо передачи данных по сети, значительное время тратится на копирование отношения на ускоритель.

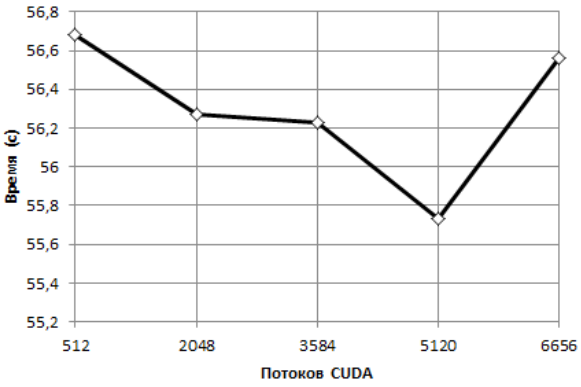

Рис. 9. Время выполнения алгоритма *ГПУ SELECT* на одном вычислительном узле

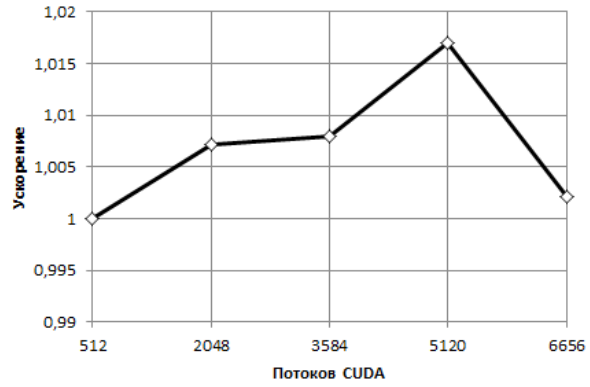

Рис. 10. Ускорение выполнения алгоритма *ГПУ SELECT* на одном вычислительном узле

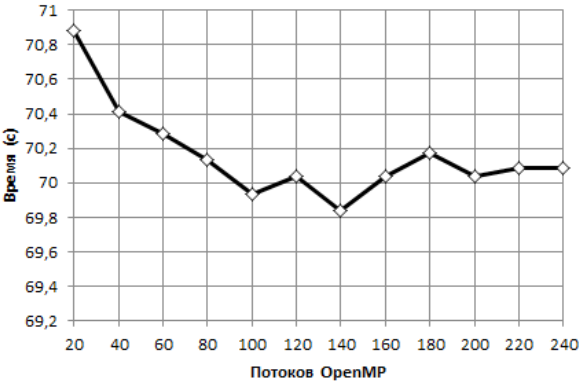

Рис. 11. Время выполнения алгоритма Manycore SELECT на одном вычислительном узле

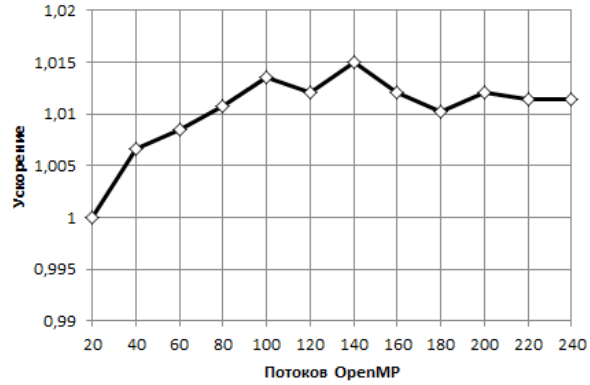

Рис. 12. Ускорение выполнения алгоритма Manycore SELECT на одном вычислительном узле

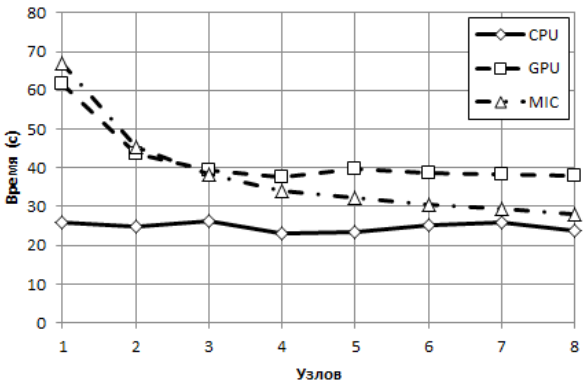

Рис. 13. Время выполнения алгоритма SELECT на нескольких вычислительных узлах

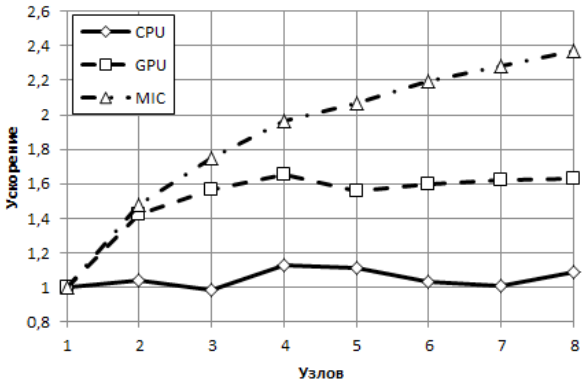

Рис. 14. скорение выполнения алгоритма SELECT на нескольких вычислительных узлах

Тестирование производительности алгоритма Manycore SELECT выполнялось на суперкомпьютере «Торнадо ЮУрГУ». Во время тестирования число потоков OpenMP варьировалось от 20 до 240. На рис. 11 представлен график времени выполнения запроса. Здесь, как и в случае с графическими ускорителями, к большим накладным расходам на передачу отношений по сети добавляется время на копирование отношений на сопроцессор. На рис. 12 изображен график ускорения выполнения запроса SELECT в зависимости от числа потоков OpenMP. Тестирование производительности при выполнении запроса SELECT в случае использования нескольких вычислительных узлов для ЦПУ и Intel Xeon Phi проводилось на суперкомпьютере «Торнадо ЮУрГУ», а для ГПУ – на вычислительном кластере ННГУ. Во время тестирования число вычислительных узлов изменялось от 1 до 8. На каждом из узлов был запущен разработанный эмулятор с параметрами, дающими максимальную производительность в случае одного узла. На рис. 13 представлен график времени выполнения запроса SELECT для нескольких вычислительных узлов.

Из рис. 14 видно, что наилучшее ускорение показал Intel Xeon Phi . Это связано с тем, что передача отношения на сопроцессор занимает значительную долю от общего времени выполнения запроса: уменьшение обрабатываемых отдельным сопроцессором фрагментов отношения по мере увеличения числа используемых узлов, позволяет получить некоторое ускорение.

Вне зависимости от числа используемых узлов кластера, время выполнения запроса с использованием ГПУ и Intel Xeon Phi превышает время выполнения запроса и использованием только центрального процессора. Таким образом, использование сопроцессоров для смоделированного варианта запроса SELECT менее эффективно, чем использование центральных процессоров. Это связано с тем, что имеющиеся накладные расходы значительно превышают сокращение непосредственного времени (то есть, без учета времени передачи данных по сети и на сопроцессор/ГПУ) выполнения запросов, вызванного использованием сопроцессоров или графических ускорителей.

Для тестирования производительности при выполнении алгоритма JOIN использовалось опорное отношение, состоящее из двух атрибутов и содержащее 33521 кортежей и тестируемое отношение из двух атрибутов и содержащее 33521000 кортежей. В обоих случаях значениями атрибутов являются целые числа (long int). Таким образом, размер опорного отношения – примерно 500 KB, размер тестируемого – примерно 500 MB.

Производительность алгоритма ЦПУ JOIN тестировалась на вычислительном узле суперкомпьютера «Торнадо ЮУрГУ» (см. табл. 1). Во время тестирования число MPI-процессов эмулятора варьировалось от 1 до 12. Графики времени выполнения алгоритма и его ускорения представлены на рис. 15 и рис. 16 соответственно.

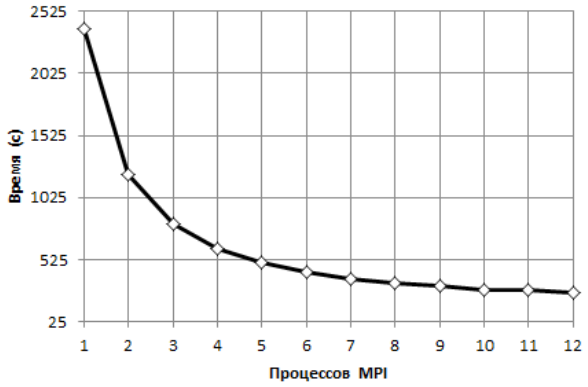

Рис. 15. Время выполнения алгоритма *ЦПУ JOIN* на одном вычислительном узле

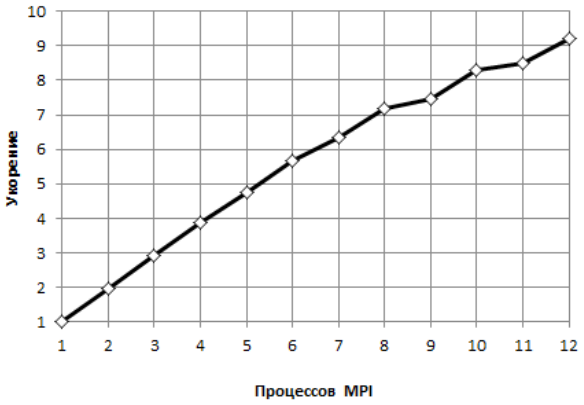

Рис. 16. Ускорение выполнения алгоритма *ЦПУ JOIN* на одном вычислительном узле

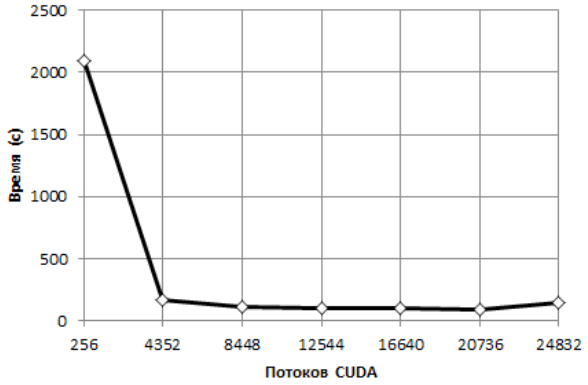

Рис. 17. Время выполнения алгоритма *ГПУ JOIN* на одном вычислительном узле

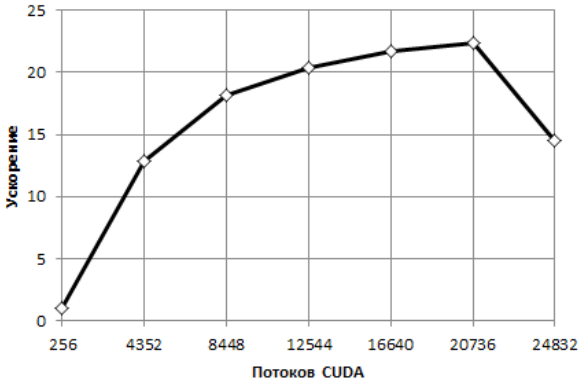

Рис. 18. Ускорение выполнения алгоритма *ГПУ JOIN* на одном вычислительном узле

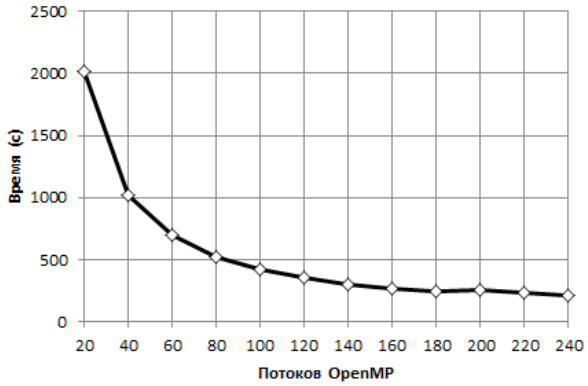

Рис. 19. Время выполнения алгоритма *Manycore JOIN* на одном вычислительном узле

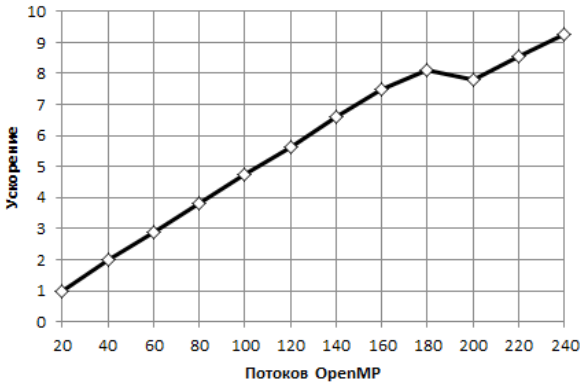

Рис. 20. Ускорение выполнения алгоритма *Manycore JOIN* на одном вычислительном узле

Производительность алгоритма ГПУ JOIN тестировалась на вычислительном узле кластера ННГУ (см. табл. 2). Во время тестирования число потоков CUDA, используемых эмулятором варьировалось от 256 до 24832. На рис. 17 показано время выполнения данного алгоритма,а на рис. 18 его ускорение.

Тестирование производительности при выполнении запроса Manycore JOIN, использующего сопроцессор Intel Xeon Phi выполнялось на вычислительном узле суперкомпьютера «Торнадо ЮУрГУ»(см табл. 1). Во время тестирования число потоков OpenMP варьировалось от 20 до 240.

Результаты данного тестирования представлены на рис. 19 и рис. 20.

Тестирование производительности при выполнении запроса JOIN в случае использования нескольких вычислительных узлов для ЦПУ и Intel Xeon Phi проводилось на суперкомпьютере «Торнадо ЮУрГУ», а для

ГПУ – на вычислительном кластере ННГУ. Использовалось от 1 до 8 вычислительных узлов кластера. На каждом из узлов был запущен разработанный эмулятор с параметрами, дающими максимальную производительность в случае одного узла.

На рис. 21 показана зависимость времени выполнения запроса JOIN в зависимости от числа используемых узлов кластера. Как видно из графика на рис. 22, все версии эмулятора показывают близкий к линейному рост ускорения по мере увеличения количества используемых вычислительных узлов кластера.

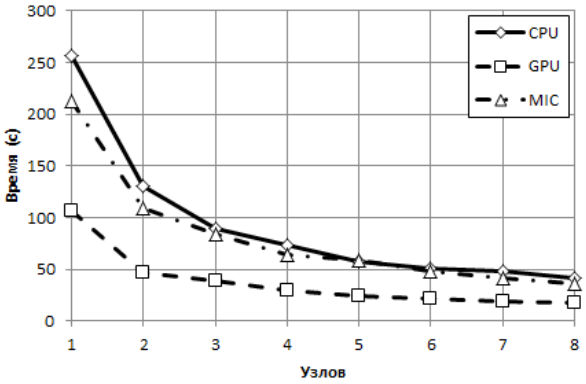

Рис. 21. Время выполнения алгоритма *JOIN* на нескольких вычислительных узлах

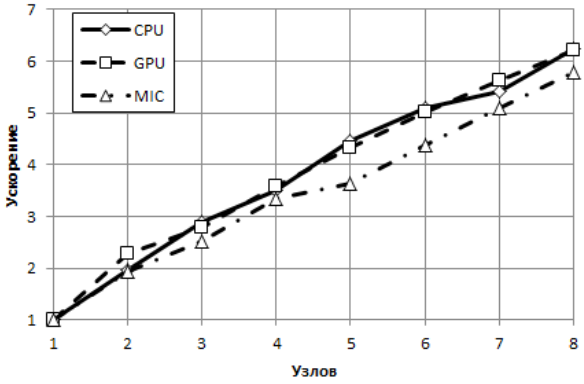

Рис. 22. Ускорение выполнения алгоритма *JOIN* на нескольких вычислительных узлах

Графические ускорители NVIDIA, и сопроцессоры Intel Xeon Phi позволяют выполнять соединение отношений быстрее, чем центральные процессоры, обладая схожими ускорениями при использовании нескольких вычислительных узлов. Отсюда можно сделать вывод о том, что они могут быть эффективно использованы для выполнения запросов INNER JOIN в параллельных СУБД.

Заключение

В рамках данной работы разработан эмулятор параллельной СУБД, использующий вычислительный кластер для выполнения запросов SELECT и JOIN. Разработаны версии запросов для ЦПУ, ГПУ и многоядерных сопроцессоров. Проведен рад вычислительных экспериментов, в которых выяснено, что как ГПУ, так и многоядерные сопроцессоры могут быть эффективно использованы для выполнения запросов JOIN.

Направлениями дальнейших исследований будут:

- исследование производительности гетерогенных вычислительных систем на приложениях баз данных;
- реализация и сравнение производительности более сложных алгоритмов выполнения запроса JOIN и SELECT на ЦПУ, ГПУ и многоядерных сопроцессорах;
- разработка алгоритмов, позволяющих одновременно задействовать центральный процессор и сопроцессор или графический ускоритель для выполнения запросов.

Работа выполнена при поддержке гранта РФФИ № 12-07-31082 (2012-2013 гг.) и гранта Президента РФ № МК-3711.2013.9 (2013-2014 гг.).

ЛИТЕРАТУРА:

1. Гарсиа-Молина Г., Ульман Д. Системы баз данных. Полный курс. Вильямс, 2003. 1088 с.
2. Костенецкий П.С. Обработка запросов на кластерных вычислительных системах с многоядерными ускорителями // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". 2012. № 47(306). Вып. 2. С. 59-67.
3. Bakkum P., Skadron K. Accelerating SQL Database Operations on a GPU with CUDA // The 3rd workshop on General-Purpose Computation on Graphics Processing Units, Pittsburg, USA, March 14, 2010, Proceedings. ACM, 2010 P. 94–103.
4. Bandi N., Sun C., Agrawal D., et al. Hardware acceleration in commertial databases: a case study of spatial operations // 30th international conference on Very Large Databases, Toronto, Canada, August 31 – September 3, 2004, Proceedings. VLDB Endowment, 2004 Vol 30, P 1021–1032
5. Blas A. D., Kaldewey, T. Data Monster // IEEE spectrum 2009. Vol. 46, No. 9.
6. Christiansen M., Hansen C. E. CUDA DBMS. Technical report. Denmark, Copenhagen: Aalborg University, 2009
7. Distributed database system WattDB: [wwwlgis.informatik.uni-kl.de/cms/dbis/projects/green/wattdb/], 30.10.2012
8. He B., Lu M., Yang K., et al. Relation query coprocessing on graphics processors // ACM Trans. Database Syst. 2009 Vol 34, No 4. P 21:1–21:39
9. He B., Yang K., Fang R., et al., Relational joins on graphics processors, // ACM SIGMOD international conference on Management of data, New York, USA, June 10–12, 2008, Proceedings. ACM, 2008 P. 511–524.

10. He B., Yu J. X. High-throughput transaction executions on graphics processors // VLDB Endowment, Seattle, Washington, USA, August 29 – September 3, 2011, Proc. VLDB Endowment 2011 Vol 4, No 5. P 314–325
11. Heimel M, Volker M. A first step towards GPU-assisted query optimizations // Third International workshop on accelerating data managment systems using modern processor and storage architectures in conjunction with VLDB, Istanbul, Turkey, August 27, 2012, P. 1–12
12. Intel Delivers New Architecture for Discovery with Intel® Xeon Phi™ Coprocessors [http://newsroom.intel.com/community/intel_newsroom/blog/2012/11/12/intel-delivers-new-architecture-for-discovery-with-intel-xeon-phi-coprocessors], 6.05.2013
13. Kim C., Chhugani J., Satish, N. et al., Designing fast architecture-sensitive tree search on modern multicore/many-core processors, // ACM Trans. Database Syst. 2011 Vol 36, No 4., P 22:1–22:34
14. Kim C., Chhugani J., Satish N., et al. FAST: fast architecture sensitive tree search on modern CPUs and GPUs // ACM SIGMOD International Conference on Managment of data, Indianapolis, USA, June 6–10, 2010, Proceedings. ACM, 2010 P. 339–350
15. Merrill D. G., Grimshaw A.S. Rewriting sorting for GPGPU stream architectures // 19th international conference on Parallel Architectures and Compilation Techniques, Vienna, Austria, September 11–15, 2010, Proceedings. ACM, 2010 P. 545–546 37
16. Satish N., Kim C., Chhugani J., et al., Fast sort on CPUs and GPUs: a case for bandwidth oblivious SIMD sort // The 2010 ACM SIGMOD International Conference on Management of data, New York, USA, June 6–11, 2010, Proceedings. ACM, 2010 P. 351–362. 36
17. Satish N., Kim C., Chhugani J., et al. Fast sort on CPUs GPUs and Intel MIC architectures // Intel Labs, 2010
18. Vitor U.R., Schal D. A GPU-operations framework for WattDB. Technical report. Germany, Kaiserslautern: University of Kaiserslautern // University of Kaiserslautern, 2012
19. Volk P .B., Habich D., Lehner W. GPU-based speculative query processing for database operations // First International workshop on accelerating data managment systems using modern processor and storage architectures in conjunction with VLDB, Singapure, September 13, 2010