ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 1(19), 2014, с. 91–110

УДК 004.657

К. Ю. Беседин, П. С. Костенецкий

Моделирование обработки запросов на гибридных вычислительных системах с многоядерными сопроцессорами и графическими ускорителями

Аннотация. Данная статья посвящена оценке эффективности применения графических ускорителей и многоядерных сопроцессоров в параллельных системах баз данных. Для этого был разработан эмулятор параллельной СУБД, позволяющий использовать вычислительный кластер, оснащенный графическими ускорителями NVIDIA и сопроцессорами Intel Xeon Phi. С помощью данного эмулятора был проведен ряд вычислительных экспериментов.

Ключевые слова и фразы: Параллельные СУБД, GPU, CUDA, Intel MIC, Intel Xeon Phi.

Введение

Применение вычислительных систем, построенных на базе многоядерных сопроцессоров и графических ускорителей, является одним из актуальных на сегодняшний день направлений исследований. Современные многоядерные сопроцессоры и графические ускорители могут быть с успехом применены для решения множества задач. Одним из примеров подобных задач являются базы данных [1]. Однако, графические ускорители и многоядерные сопроцессоры обладают целым рядом технических особенностей, что делает такое их использование довольно нетривиальной задачей, требующей модификации 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.

Работа выполнена при поддержке гранта РФФИ № 12-07-31082 (2012–2013 гг.), № 12-07-00443-а (2012–2014 гг.) и гранта Президента РФ № МК-3711.2013.9 (2013–2014 гг.)

На данный момент уже существуют работы, посвященные эффективной реализации специфичных для СУБД алгоритмов с использованием ГПУ. Так, [2, 3] описывают алгоритм работы с деревом индекса, учитывающий архитектурные особенности современных ГПУ и ЦПУ, который может быть использован для организации индекса. Работа [4] предлагает метод ускорения операций над индексом, эффективно использующий большое число вычислительных ядер ГПУ. Статьи [5,6] описывают алгоритмы сортировки, [7] описывает реализацию нескольких алгоритмов операции JOIN. В [8] рассматриваются реализации запросов SELECT и JOIN. Помимо этого, исследуются новые архитектурные решения, позволяющие наиболее эффективно использовать особенности графических ускорителей. В работе [9] 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). В статье [10] рассматривается группировка транзакций с целью их последующего выполнения на графическом ускорителе. Работа [11] предлагает использование ГПУ для оптимизации запросов. В техническом отчете [12] рассмотрена разработка прототипа СУБД, хранящей свои данные в памяти графического ускорителя. Некоторыми авторами проводится адаптация существующих СУБД для использования графических ускорителей и оценка эффективности такой адаптации. Так, в [13] поддержка вычислений с использованием ГПУ была интегрирована в СУБД Oracle 9, в [14] поддержка графических ускорителей была добавлена в WattDB [15], а в работе [16] использовалась SQLite.

Еще одной перспективной многоядерной архитектурой является разрабатываемая компанией Intel архитектура Intel MIC. Количество исследований, посвященных этой архитектуре, меньше, чем количество исследований, посвященных ГПУ. Это можно объяснить тем, что ускорители, построенные на основе этой архитектуры, лишь недавно были представлены широкой публике [17]. Однако уже проведено несколько исследований, посвященных использованию Intel MIC в базах данных. Так, в [18] 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 [2], 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 трудов 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.

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.

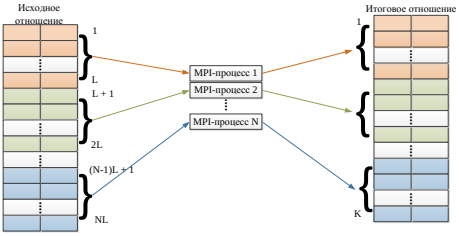

Рис. 1. Схема выполнения *ЦПУ SELECT*

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 этим накладных расходов. Конкретная реализация, 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;

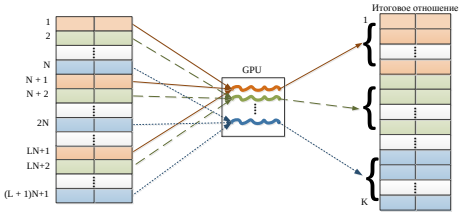

Рис. 2. Схема выполнения *ГПУ SELECT*

(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;

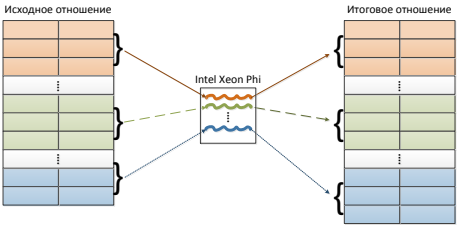

Рис. 3. Схема выполнения *Manycore SELECT*

(3) распределить тестируемое отношение между рабочими процессами;
(4) выполнить соединение;
(5) собрать результаты соединений и соединить их в итоговое отношение.

Для выполнения выборки используются алгоритмы вложенных циклов и Hash Join [19]. 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 расходов.

Алгоритм ЦПУ NESTED LOOPS JOIN последовательно проверяет каждый кортеж опорного отношения и каждый кортеж тестируемого отношение на возможность соединения. Схема выполнения ЦПУ NESTED LOOPS JOIN изображена на рис. 4. Ниже представлен алгоритм выполнения ЦПУ NESTED LOOPS 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 NESTED LOOPS JOIN использует графический процессор для вычисления размера итогового отношения и непосредственного выполнения выборки. Кортежи отношения делятся между потоками CUDA так, чтобы воспользоваться объединенным (coalesced)

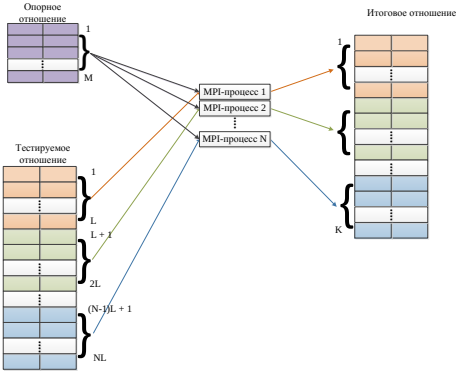

Рис. 4. Схема выполнения *ЦПУ NESTED LOOPS JOIN*

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 каждого 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 NESTED LOOPS 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

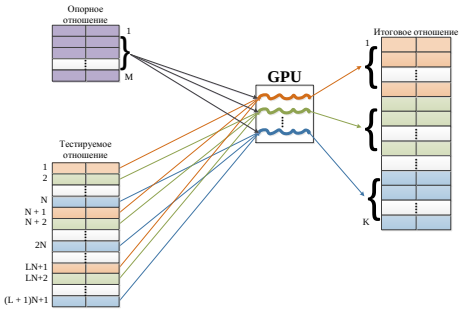

Рис. 5. Схема выполнения *ГПУ NESTED LOOPS JOIN*

OpenMP. Схема выполнения Manycore NESTED LOOPS JOIN изображена на рис. 6. Ниже представлен алгоритм Manycore NESTED LOOPS 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.

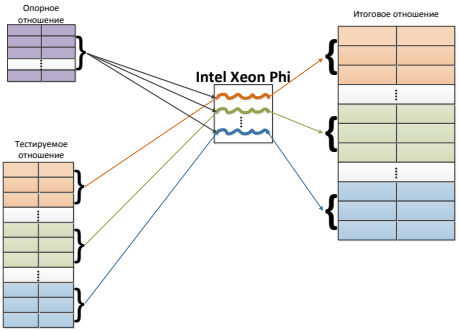

Рис. 6. Схема выполнения *Manycore NESTED LOOPS JOIN*

Таблица 1. Суперкомпьютер "Торнадо ЮУрГУ"

Узел	Процессор	Intel Xeon 5680 3.33 GHz
	Объем ОЗУ	24 / 48 GB
	Число процессоров	2
	Сопроцессор	Intel Xeon Phi 7110X
Число узлов		480
Число узлов с сопроцессорами		192

Таблица 2. Вычислительный кластер ННГУ

Узел	Процессор	Intel Xeon L5630 2.13 GHz
	Объем ОЗУ	24 GB
	Число процессоров	2
	GPU	NVIDIA Tesla X2070
	Число GPU	2
Число узлов с ускорителями		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

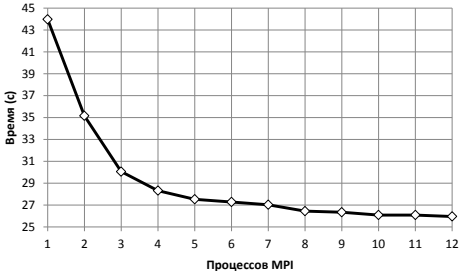

Рис. 7. Время выполнения алгоритма *ЦПУ SELECT* на одном вычислительном узле

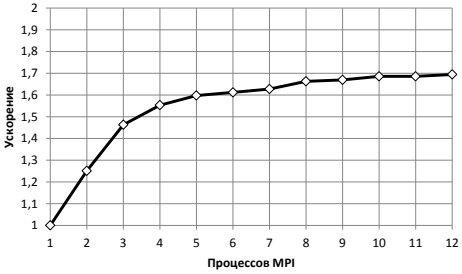

Рис. 8. Ускорение выполнения алгоритма *ЦПУ SELECT* на одном вычислительном узле

369098752 кортежей. Таким образом, примерный размер отношения — 5.5 GB.

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

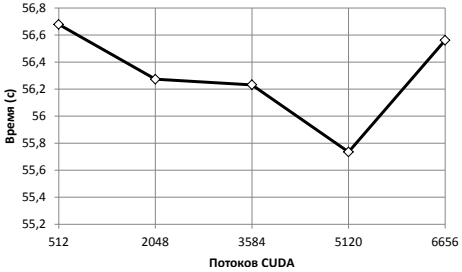

Рис. 9. Время выполнения алгоритма *ГПУ SELECT* на одном вычислительном узле

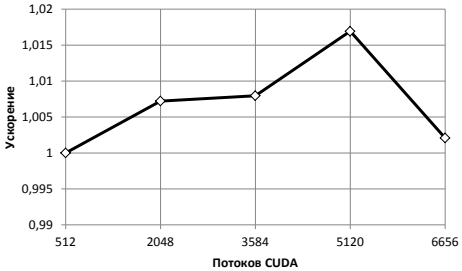

Рис. 10. Ускорение выполнения алгоритма *ГПУ SELECT* на одном вычислительном узле

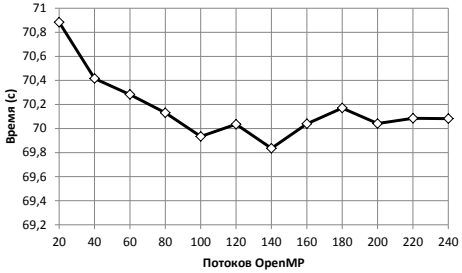

Рис. 11. Время выполнения алгоритма *Manycore SELECT* на одном вычислительном узле

связано с тем, что помимо передачи данных по сети, значительное время тратится на копирование отношения на ускоритель.

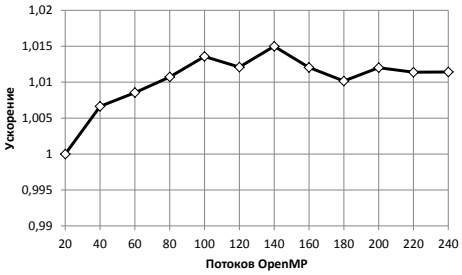

Рис. 12. Ускорение выполнения алгоритма *Manycore SELECT* на одном вычислительном узле

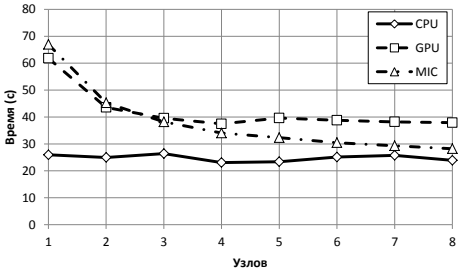

Рис. 13. Время выполнения алгоритма *SELECT* на нескольких вычислительных узлах

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

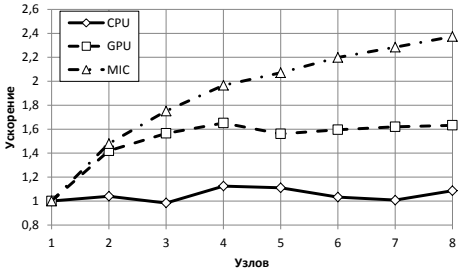

Рис. 14. Ускорение выполнения алгоритма *SELECT* на нескольких вычислительных узлах

с параметрами, дающими максимальную производительность в случае одного узла. На рис. 13 представлен график времени выполнения запроса *SELECT* для нескольких вычислительных узлов. На рис. 14 представлен график ускорения выполнения запроса *SELECT*. 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

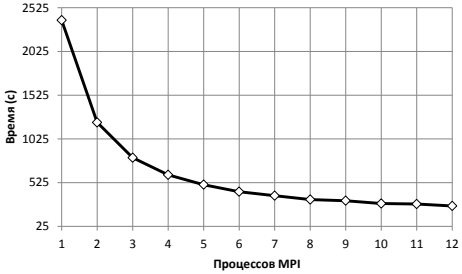

Рис. 15. Время выполнения алгоритма *ЦПУ JOIN* на одном вычислительном узле

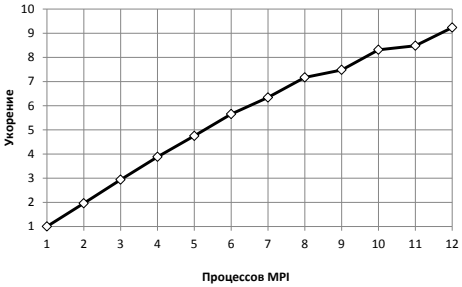

Рис. 16. Ускорение выполнения алгоритма *ЦПУ JOIN* на одном вычислительном узле

значениями атрибутов являются целые числа (`int64_t`). Таким образом, размер опорного отношения — примерно 500 KB, размер тестируемого — примерно 500 MB.

Производительность алгоритма *ЦПУ NESTED LOOPS JOIN* тестировалась (см. табл. 1) на вычислительном узле суперкомпьютера «Торнадо ЮУрГУ». 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.

Производительность алгоритма *ГПУ NESTED LOOPS 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.

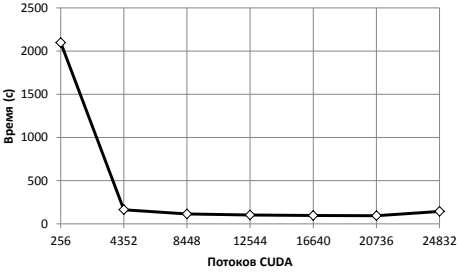

Рис. 17. Время выполнения алгоритма *ГПУ JOIN* на одном вычислительном узле

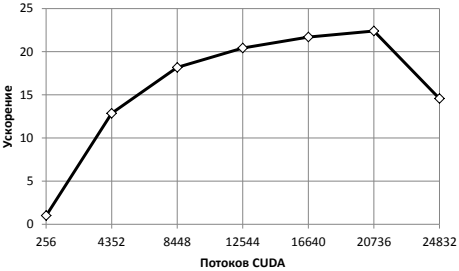

Рис. 18. Ускорение выполнения Время выполнения алгоритма *ГПУ JOIN* на одном вычислительном узле

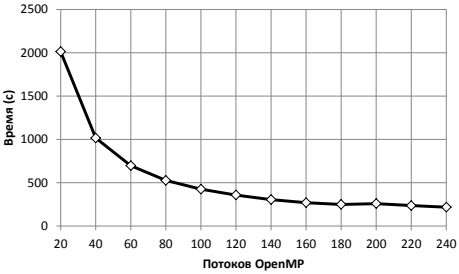

Рис. 19. Время выполнения алгоритма *Manycore JOIN* на одном вычислительном узле

Тестирование производительности при выполнении запроса *Manycore NESTED LOOPS JOIN*, использующего сопроцессор Intel Xeon Phi, выполнялось на вычислительном узле суперкомпьютера «Тор-

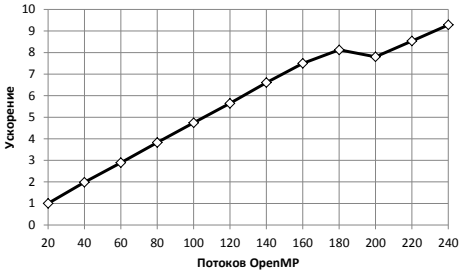

Рис. 20. Ускорение выполнения алгоритма *Manycore JOIN* на одном вычислительном узле

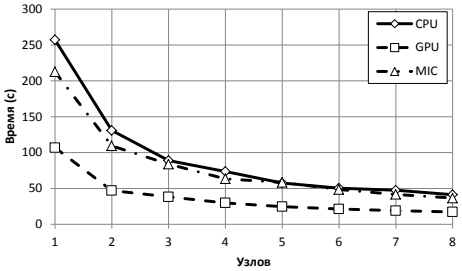

Рис. 21. Время выполнения алгоритма *JOIN* на нескольких вычислительных узлах

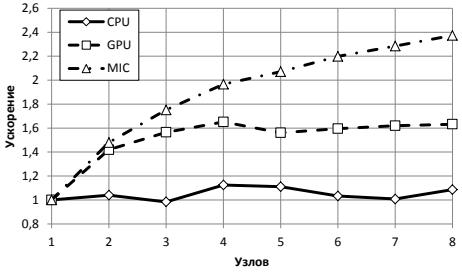

Рис. 22. Ускорение выполнения алгоритма *JOIN* на нескольких вычислительных узлах

надо 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.

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 авторами представлен 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 при передаче данных шина PCI Express является узким местом для обработки больших объемов данных, и при обработке простых реляционных запросов большую часть времени занимает передача данных на сопроцессор. В случае с запросами, обладающими высокой вычислительной сложностью, передача данных занимает меньше времени. В этом случае высокая производительность сопроцессоров позволяет им эффективно обрабатывать такие запросы. Объем передаваемых на сопроцессор данных можно значительно уменьшить, применив поколоночное хранение данных. Так, при обработке запросов можно передавать на сопроцессор только те колонки, которые непосредственно требуются для обработки запроса. Для еще большего снижения объема данных можно воспользоваться высокой эффективностью сжатия в поколоночных СУБД и передавать данные на сопроцессор в сжатом виде. При этом, за счет высокой производительности сопроцессоров,

время, требуемое на упаковку и распаковку данных, будет меньше времени, сэкономленного при передаче данных на сопроцессор за счет сжатия. Таким образом, в дальнейшем будут разрабатываться методы передачи данных на сопроцессор в сжатом виде и исследование эффективности различных алгоритмов сжатия для многоядерных сопроцессоров.

Список литературы

[1] A. D Blas, T. Kaldewey *Data Monster* // IEEE spectrum, 2009. Vol. **46**, no. 9. ↑91

[2] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt, P. Dubey *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. ↑92, 93

[3] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt, P. Dubey *FAST: fast architecture sensitive tree search on modern CPUs and GPUs* // ACM SIGMOD International Conference on Managment of data: Proceedings — Indianapolis, USA, June 6–10, 2010: ACM, 2010, p. 339–350. ↑92

[4] P. B. Volk, D. Habich, W. Lehner *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. ↑92

[5] D. G. Merrill, A. S. Grimshaw *Rewriting sorting for GPGPU stream architectures* // 19th international conference on Parallel Architectures and Compilation Techniques: Proceedings — Vienna, Austria, September 11–15, 2010: ACM, 2010, p. 545–546. ↑92

[6] N. Satish, C. Kim, J. Chhugani, A. D. Nguyen, V. W. Lee, D. Kim, P. Dubey *Fast sort on CPUs and GPUs: a case for bandwidth oblivious SIMD sort* // The 2010 ACM SIGMOD International Conference on Management of data: Proceedings — New York, USA, June 6–11, 2010: ACM, 2010, p. 351–362. ↑92

[7] B. He, K. Yang, R. Fang, M. Lu, N. K. Govindaraju, Q. Luo, P. V. Sander *Relational joins on graphics processors* // ACM SIGMOD international conference on Management of data: Proceedings — New York, USA, June 10–12, 2008: ACM, 2008, p. 511–524. ↑92

[8] П. С. Костенецкий *Обработка запросов на кластерных вычислительных системах с многоядерными ускорителями* // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика», 2012. Т. **47(306). Вып. 2**, с. 59–67. ↑92

[9] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, P. V. Sander *Relation query coprocessing on graphics processors* // ACM Trans. Database Syst, 2009. Vol. **34**, no. 4, p. 21:1–21:39. ↑92

[10] B. He, J. X. Yu *High-throughput transaction executions on graphics processors* // VLDB Endowment: Proceedings — Seattle, Washington, USA, August 29 – September 3, 2011: VLDB Endowment, 2011. Vol. **4**, no. 5, p. 314–325. ↑92

[11] M. Heimel, M. Volker *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. ↑92

[12] M. Christiansen, C. E. Hansen. *CUDA DBMS.*/ Aalborg University Denmark, Copenhagen, Aalborg University, 2009 ↑92

[13] N. Bandi, C. Sun, D. Agrawal, A. E. Abbadi *Hardware acceleration in commertial databases: a case study of spatial operations* // 30th international conference on Very Large Databases: Proceedings — Toronto, Canada, August 31 – September 3, 2004: VLDB Endowment, 2004. Vol. **30**, p. 1021–1032. ↑92

[14] U. R. Vitor, D. Schal. *A GPU-operations framework for WattDB.*/ University of Kaiserslautern Germany, Kaiserslautern, University of Kaiserslautern, 2012 ↑92

[15] Distributed database system WattDB, 30.10.2012, URL: http://wwwlgis.informatik.uni-kl.de/cms/dbis/projects/green/wattdb/. ↑92

[16] P. Bakkum, K. Skadron *Accelerating SQL Database Operations on a GPU with CUDA* // The 3rd workshop on General-Purpose Computation on Graphics Processing Units: Proceedings — Pittsburg, USA: ACM, March 14, 2010, p. 94–103. ↑92

[17] Intel Delivers New Architecture for Discovery with Intel Xeon Phi Coprocessors: Combination of Intel Xeon processors and Intel Xeon Phi coprocessors promises to drive high performance computing innovation, 6.05.2013, URL: http://newsroom.intel.com/community/intel_newsroom/blog/2012/11/12/intel-delivers-new-architecture-for-discovery-with-intel-xeon-phi-coprocessors. ↑92

[18] N. Satish, C. Kim, J. Chhugani, A. D. Nguyen, V. W. Lee, D. Kim, P. Dubey. *Fast sort on CPUs GPUs and Intel MIC architectures*: Intel Labs, 2010 ↑92

[19] Г. Гарсиа-Молина, Д. Ульман. Системы баз данных. Полный курс. Москва: Вильямс, 2003. — 1088 с. ↑96

Рекомендовал к публикации *Программный комитет*
Второго национального суперкомпьютерного форума ***НСКФ-2013***

Об авторах:

Константин Юрьевич Беседин
Студент ФГБОУ ВПО ЮУрГУ (НИУ).
e-mail: besedin.k@gmail.com

Павел Сегеевич Костенецкий

Руководитель Лаборатории Суперкомпьютерного Моделирования. Кандидат физ.-мат. наук. Доцент кафедры системного программирования факультета вычислительной математики и информатики ФГБОУ ВПО ЮУрГУ(НИУ).

e-mail: kostenetskiy@gmail.com

Образец ссылки на эту публикацию:

К. Ю. Беседин, П. С. Костенецкий. *Моделирование обработки запросов на гибридных вычислительных системах с многоядерными сопроцессорами и графическими ускорителями* // Программные системы: теория и приложения: электрон. научн. журн. 2014. Т. 5, № 1(19), с. 91–110.

URL: http://psta.psiras.ru/read/psta2014\protect\T2A\textunderscore1\protect\T2A\textunderscore91-110.pdf

K. Y. Besedin, P. S. Kostenetskiy. *Simulating of query processing on multiprocessor database systems with modern coprocessors.*

Abstract. This paper focuses on evaluation of database multiprocessor architectures with manycore coprocessors and GPUs. We implemented the emulator of parallel DBMS that uses computing cluster with NVIDIA GPUs or Intel Xeon Phi coprocessors for relational query processing. A number of experiments have been done using this emulator. (*in Russian*).

Key Words and Phrases: Parallel DBMS, GPU, CUDA, Intel MIC, Intel Xeon Phi.