Гипотетические принципы организации консервативных СУБД на кластерной платформе с графическими ускорителями

Аспирант Р.К. Классен

Научный руководитель:

д.ф.-м.н., профессор В. А. Райхлин

Введение

В современных вычислительных системах все активнее применяются графические ускорители. В частности, они применяются для ускорения работы СУБД (CoGaDb, расширение PG-Storm для Postgres и т.д.). Графический ускоритель (GPU) существенно ускоряет фрагментарную обработку, однако скорость обмена данными с ним сравнительно невелика. Именно по этой причине в работе [Rauhe H. Finding the Right Processor for the Job Co-Processors in a DBMS, Ilmenau University of Technology, Ilmenau, Dissertation urn:nbn:de:gbv:ilm1-2014000240, 2014] удалось достичь роста производительности сервера БД от использования GPU всего на 40%. Повышение эффективности консервативных СУБД на платформе GPU-кластеров требует особой организации хранения данных и работы таких СУБД.

Обзор существующих GPU СУБД

СУБД с GPU-ускорителями

Система	Институт	Год	Открытый исходный код	На сегодняшний день
CoGaDB	Universit¨ at Magdeburg	2013	yes	Развивается
GPUDB	Ohio State University	2013	yes	Комерческий продукт (kinetica)
GPUQP	Hong Kong University of Science and Technology	2007	yes	последнее обновление в 2009 году
GPUTx	Nanyang Technological University	2011	no	-
MapD	Massachusetts Institute of Technology	2013	no	Комерческий продукт
Ocelot	Technische Universit¨ at Berlin	2013	yes	последнее обновление в 2015 году
OmniDB	Nanyang Technological University	2013	yes	последнее обновление в 2014 году
Virginian	NEC Laboratories America	2012	yes	последнее обновление в 2012 году

Система хранения

DBMS	Storage System		Storage Model	
	Main-Memory Storage	Disk-based Storage	Column Store	Row Store
CoGaDB	X	-	X	×
GPUDB	X	-	X	×
GPUQP	X	X	X	×
GPUTx	X	-	X	×
MapD	X	X	X	×
Ocelot	X	-	X	×
OmniDB	X	-	X	×
Virginian	X	-	◦	◦

Модель обработки

DBMS	Operator-at-a-Time	Block-at-a-Time	Just-in-Time Compilation
CoGaDB	X	-	-
GPUDB	X	X	X
GPUQP	X	-	-
GPUTx	◦	◦	◦
MapD	X	X	X
Ocelot	X	-	-
OmniDB	X	X	-
Virginian	X	X	-

Поддержка нескольких графических ускорителей

DBMS	Single-Device Processing	Cross-Device Processing
CoGaDB	X	X
GPUDB	X	-
GPUQP	X	X
GPUTx	X	-
MapD	X	X
Ocelot	X	-
OmniDB	X	X
Virginian	X	-

Проблемы и ограничения использования графических ускорителей в СУБД

- Относительно малый объем глобальной памяти графического ускорителя
- Относительно медленная шина передачи данных (PCI-e)

Малый объем памяти

Серия	Год	Объем видеопамяти, Мб
GeForce 8 Series	2007	1024
GeForce 9 Series	2008	1536
GeForce 100 Series	2009	1536
GeForce 200 Series	2009	1792
GeForce 300 Series	2010	2048
GeForce 400 Series	2010	2048
GeForce 500 Series	2010	3072
GeForce 600 Series	2012	4096
GeForce 700 Series	2013	6144
GeForce 900 Series	2014	12288
GeForce 1000 Series	2016	12288

Малая пропускная способность PCI-e

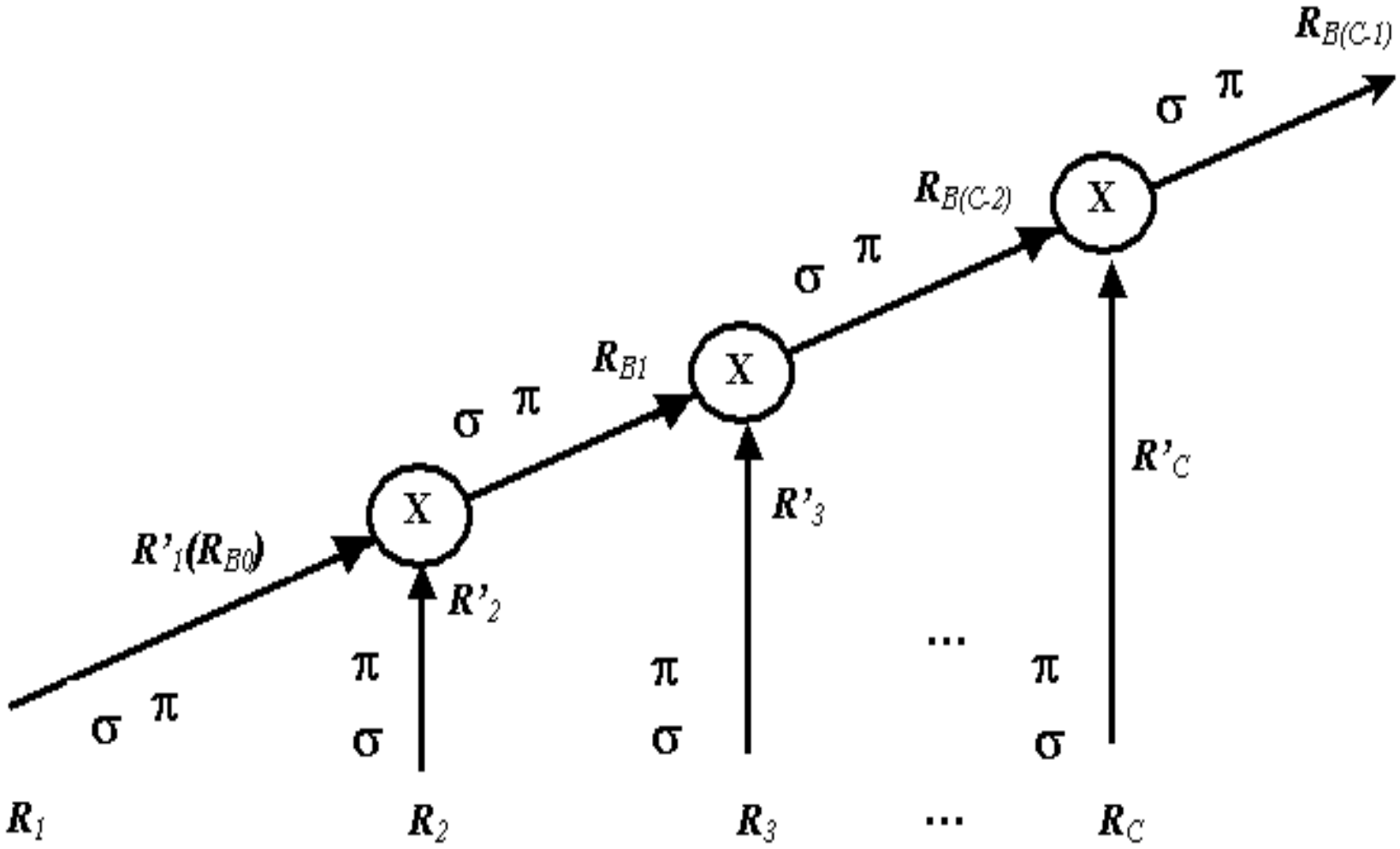
- Скорость чтения/записи для трёхканальной оперативной памяти типа DDR3-1600 составляет 38400 Мбайт/с, в то время как для шины PCI-e 2.0 x16 – 6,4 Гбайт/с (64 Гбит/с * 0.8, где 64 Гбит/с - пропускной способности шины в одну сторону, 0.8 - учёт избыточности 8b/10b для PCI-e 1.0 и 2.0; 0,985 — 128b/130b для PCI-e 3.0)

Связей	x1	x2	x4	x8	x12	x16	x32
PCIe 1.0	2/4	4/8	8/16	16/32	24/48	32/64	64/128
PCIe 2.0	4/8	8/16	16/32	32/64	48/96	64/128	128/256
PCIe 3.0	8/16	16/32	32/64	64/128	96/192	128/256	256/512
PCIe 4.0 (предварительно)	16/32	32/64	64/128	128/256	192/384	256/512	512/1024

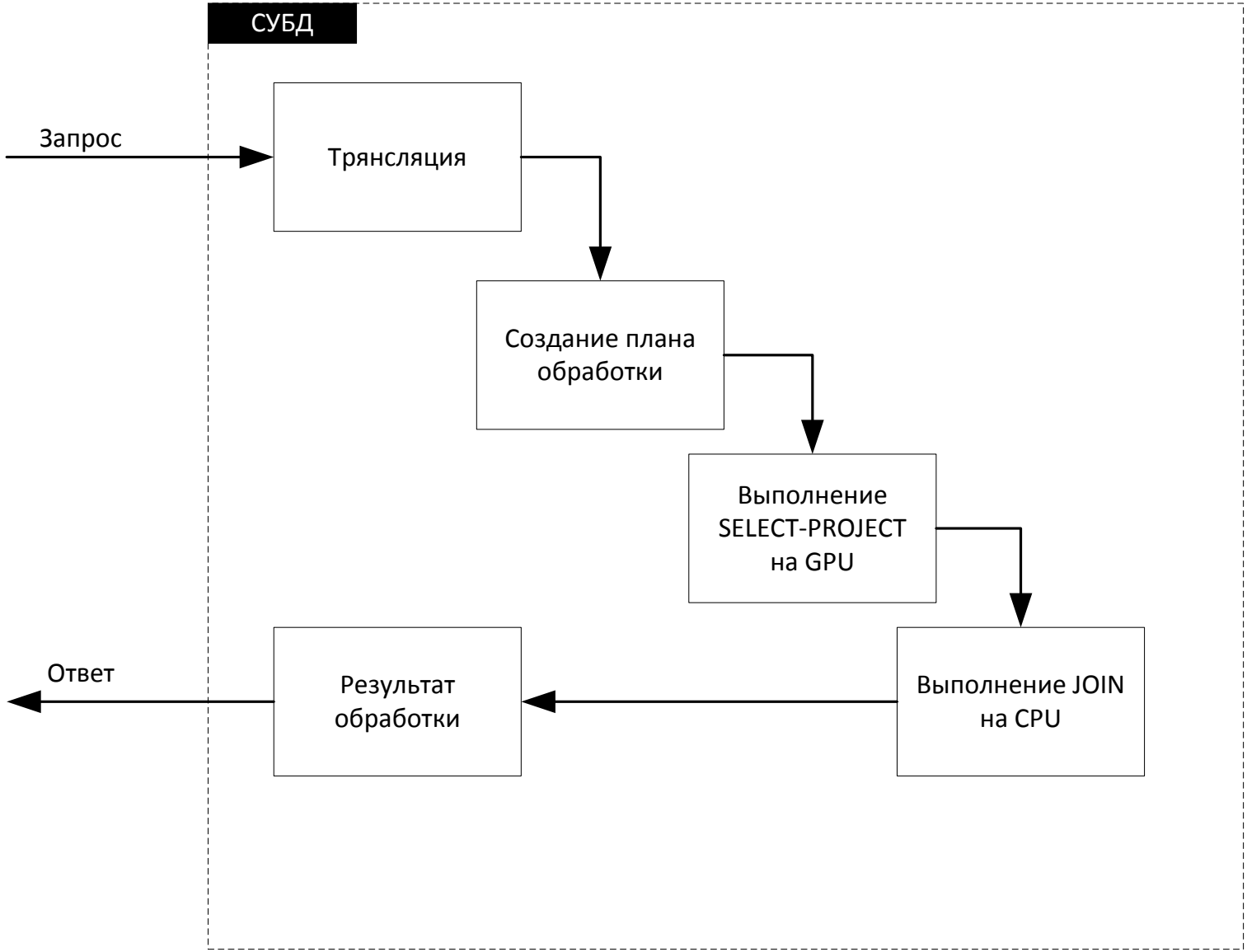
Предполагаемая организация работы СУБД с графическими ускорителями

Графические ускорители обладают большим количеством вычислительных ядер и быстрой памятью. Это позволяет быстро выполнять параллельные операции. В работе с БД представляется возможным параллельное выполнение всех операций SELECT-PROJECT-JOIN, но в связи с ограниченностью памяти графического ускорителя выполнение операции JOIN не всегда возможно. Сжатие данных позволяет минимизировать время передачи информации в графический ускоритель. Выполнение запроса происходит по регулярному плану. Операции SELECT-PROJECT выполняются на графических ускорителях. Операция JOIN выполняется на CPU.

Регулярный план обработки запросов



Обработка запроса

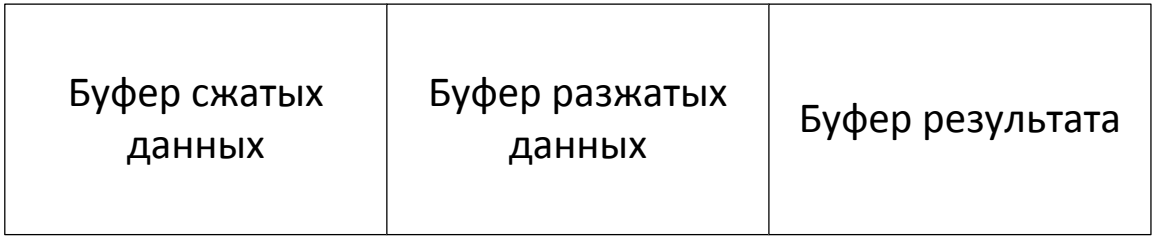


Предполагаемый конвейер выполнения операции SELECT-PROJECT

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
Передача данных	■		■		■		■		■		■	
Разжатие данных		■		■		■		■		■		■
Выполнение SELECT-PROJECT			■		■		■		■		■	
Пересылка результата из GPU				■		■		■		■		■

- t1 – поступление блока в GPU
- t2 – разжатие данных блока в GPU
- t3 – поступление следующего блока в GPU и выполнение операции SELECT-PROJECT
- t4 – разжатие данных нового блока в GPU и передача результата операции SELECT-PROJECT в хост
- t5 – поступление следующего блока в GPU и выполнение операции SELECT-PROJECT
- t6 – разжатие данных нового блока в GPU и передача результата операции SELECT-PROJECT в хост и т.д.

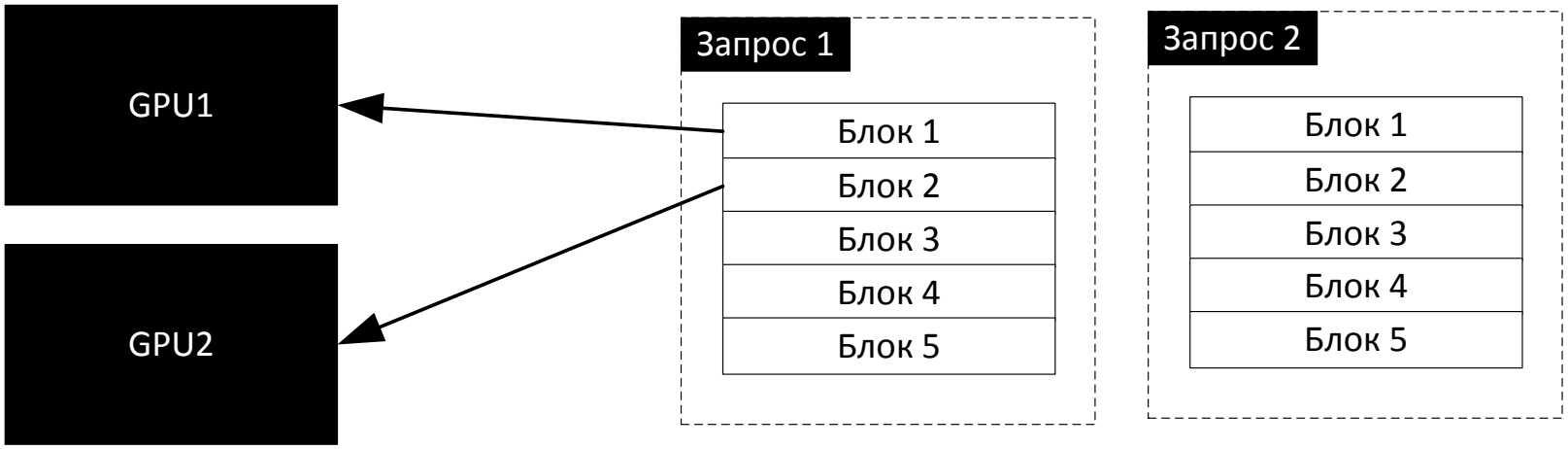
Разделение памяти для конвейерной обработки в GPU



На графическом ускорителе одновременного происходит обработка 2-х блоков. Т.к. данные блоков не должны пересекаться в памяти графического ускорителя, то память должна разделяться на 3 равные секции:

- Буфер сжатых данных – используется для пересылки сжатых данных с хоста.
- Буфер разжатых данных – используется выполнения операций SELECT-PROJECT
- Буфер результата – используется хранения результата SELECT-PROJECT и передачи результата в хост.

Использование нескольких GPU



В случае использования нескольких графических ускорителей последовательность блоков начинает обрабатываться параллельно. Распределение блоков происходит по алгоритму round-robin. Так при обработке 5 блоков на 2-х видеокартах получим следующее распределение: блоки под номерами 1,3,5 обрабатываются на первом графическом ускорителе, а блоки под номерами 2,4 – на втором.

Разбиение на блоки

Под блоком данных здесь понимается набор сжатых столбцов с общим объемом разжатых данных равным буферу разжатых данных. Алгоритм подготовки данных для сжатия следующий:

- Найти самую длинную запись в обрабатываемом отношении – RS
- Найти количество записей в отношении, которое гарантированно умещается в отведенной памяти, по формуле: RC=L/(RS*L/BS) с округлением выражения (RS*L/BS) в большую строну, где RC – количество записей для сжатия в одном блоке, L – количество строк в отношении, BS – объем памяти, отведенной для разжатых данных в графическом ускорителе.
- Выдавать данные для сжатия из отношения с шагом RC.

Например, требуется сжать отношение длинной 5 000 000 строк с 4-я колонками (1 – int, 2 – char, 3 – string(50), 4 – text) для буфера размером 100 Мбайт. Согласно алгоритму в начале будет выполнен поиск по отношению с целью выявления строки максимальной длины. Этот шаг необходим при наличии поля с типом text (т.к. оно хранит данные произвольной дины), во всех остальных случаях длина строки заранее известна. Предположим, максимальная длина нашлась равной 100 байт (4 байта – int, 1 байт – char, 50 байт – string, 55 байт – text), тогда согласно формуле второго пункта алгоритма количество строк в блоке будет равно: 5 000 000 / (100 * 5 000 000 / (100*1024*1024)) = 1 000 000. Количество сжатых блоков – 5. Объемы данных в одном блоке для каждого столбца равны: 1 – 4 000 000 байт, 2 – 1 000 000 байт, 3 – 50 000 000, 4 – до 55 000 000 байт.

int	char	String (50)	text	Блок 1
int	char	String (50)	text	Блок 2
int	char	String (50)	text	Блок 3
int	char	String (50)	text	Блок 4
int	char	String (50)	text	Блок 5

Завершение операции SELECT-PROJECT

После операции SELECT-PROJECT сжатие данных для передачи не происходит. Это связано с тем, что в результате выполнения операции SELECT-PROJECT объем данных существенно уменьшается. В таблице приведены объемы данных подлежащих обработке до выполнения операций SELECT-PROJECT и после для запросов из теста TPC-H при объеме БД равном 1 Гбайт.

Результат выполнения SELECT-PROJECT перемещается из памяти графического ускорителя в память хоста и готовится к операции JOIN

Объемы данных до и после выполнения операций SELECT-PROJECT для запросов теста TPC-H

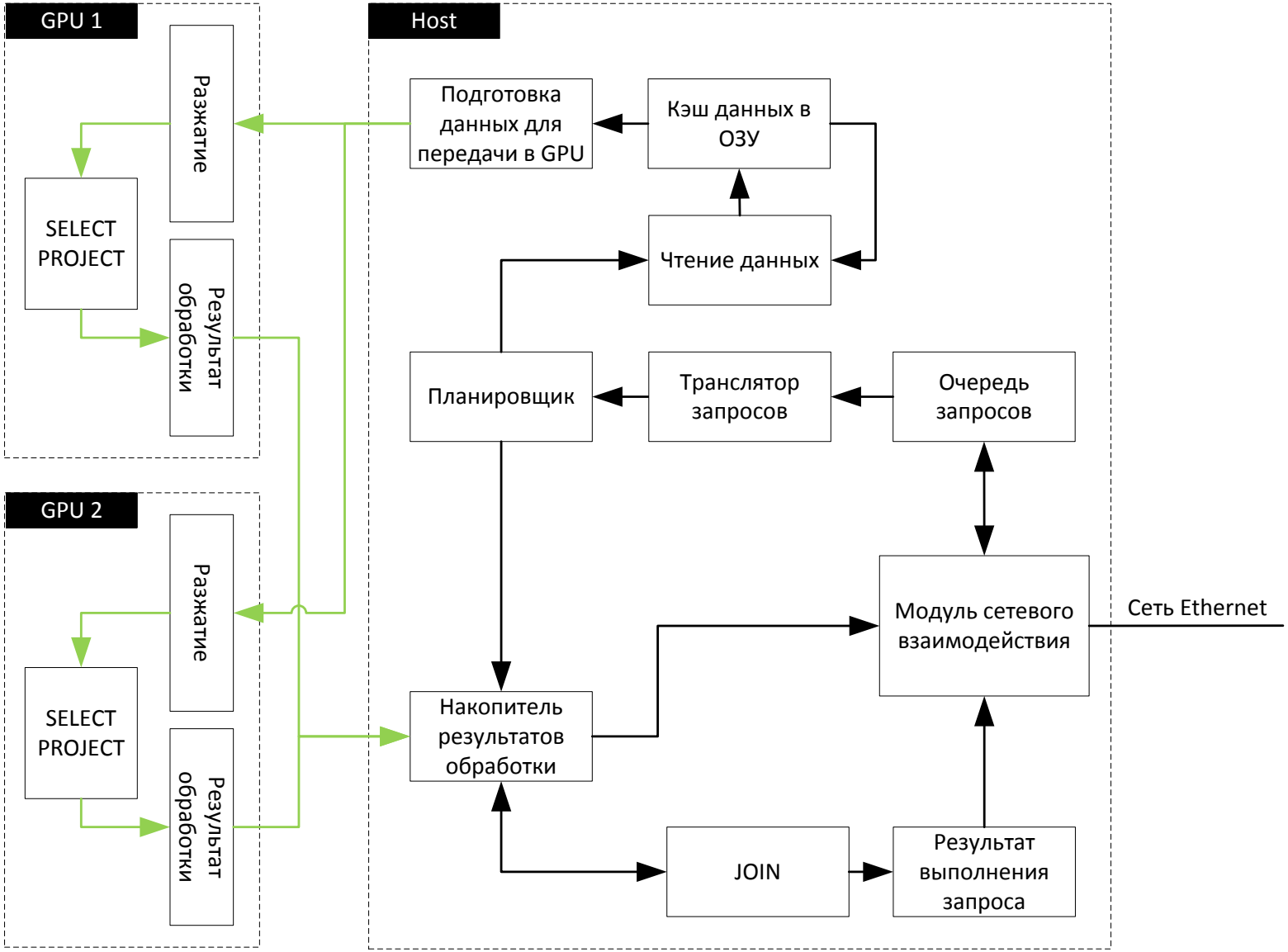
№ запроса	Объем данных, байт		Коэффициент уменьшения данных
	До SELECT-PROJECT	После SELECT-PROJECT	
1	675 846 277	134 255 929	5,03
2	137 651 393	13 526 703	10,18
3	855 794 582	76 991 966	11,12
4	832 798 438	428 049 230	1,95
5	857 126 234	139 324 211	6,15
6	675 846 277	1 339 256	504,64
7	857 125 865	78 759 670	10,88
8	879 261 359	179 338 247	4,90
9	970 449 462	234 598 226	4,14
10	855 796 681	49 964 804	17,13
11	342 555 947	27 528 586	12,44
12	832 798 438	23 140 549	35,99
13	179 948 305	18 706 950	9,62
14	697 981 402	6 548 108	106,59
Среднее	689 355 761	100 862 317	52,91

Операция JOIN

Операция JOIN выполняется при наличии как минимум 2-х результатов операций SELECT-PROJECT, а ее результат размещается либо вместо 2-х предыдущих результатов (если требуется выполнить еще одну операцию JOIN), либо в буфер конечного результата (если JOIN был последним для текущего запроса), откуда будет выполнена передача по сети.

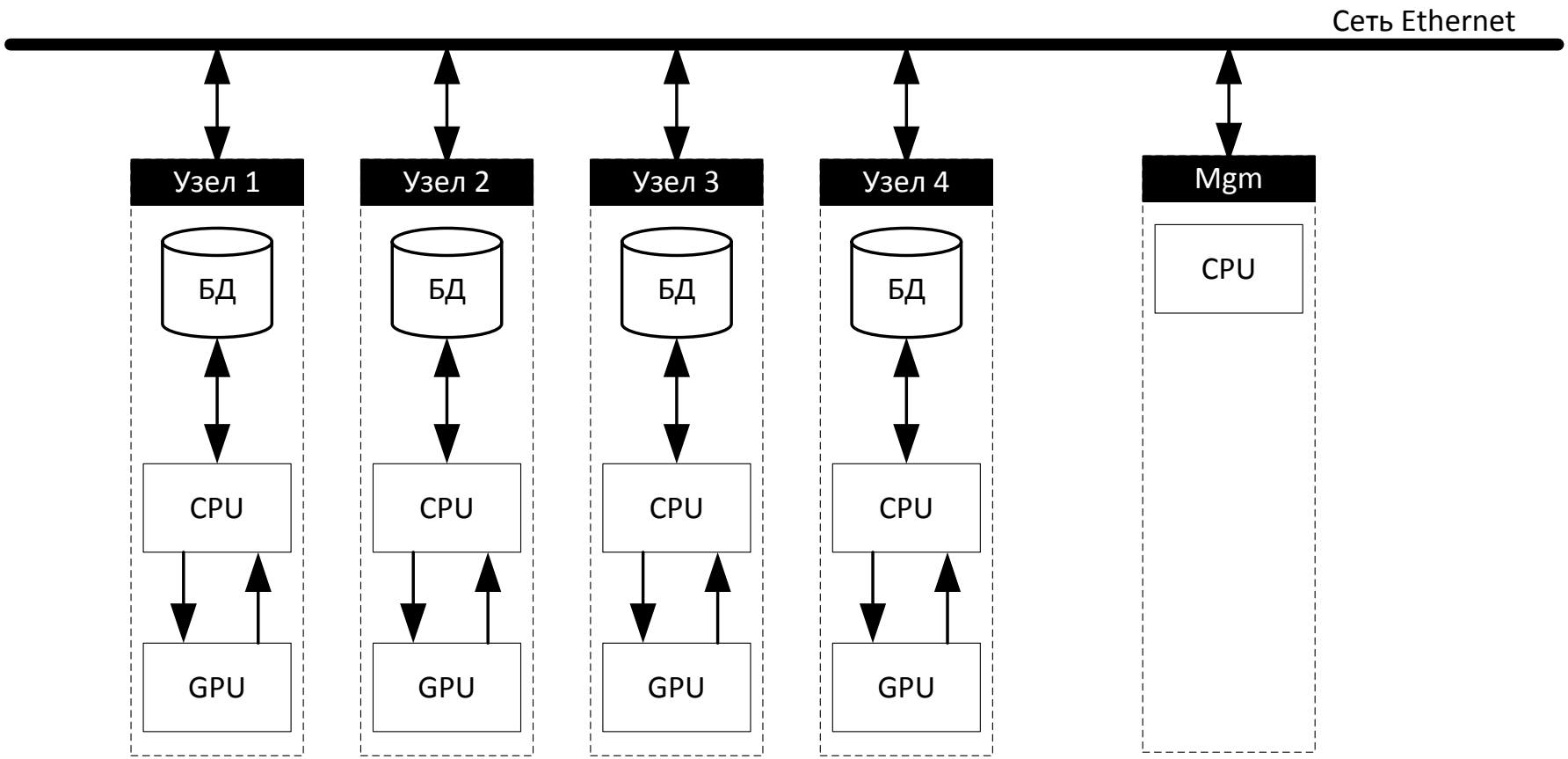
Каждая операция JOIN занимает одно ядро CPU, поэтому если в системе имеются данные для выполнения сразу двух операций JOIN (например, выполнены все операции SELECT-PROJECT для одного запроса и выполнено часть операций SELECT-PROJECT для другого запроса), то они должны выполняться параллельно.

Предполагаемая организация GPU СУБД



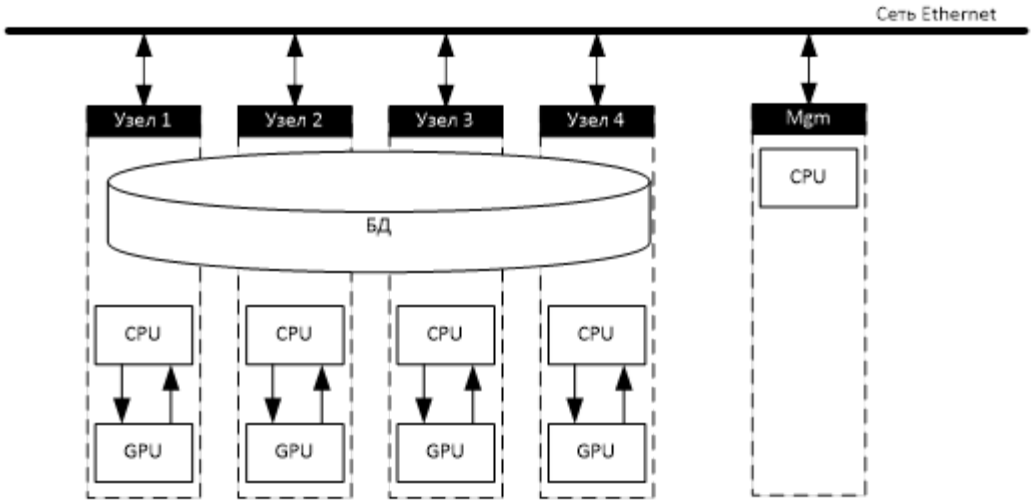
Узел на N/2+1 запросов

- Повторяет стратегию «узел на N+1 запросов» [Классен Р.К. Повышение эффективности параллельной СУБД консервативного типа на кластерной платформе с многоядерными узлами] с несколькими отличиями.
  - Во-первых, половина ядер здесь являются управляющими, а другая половина исполняющими.
  - Во-вторых запросы в очереди выполняются на графических ускорителях последовательно, и только операции JOIN выполняются параллельно.
- В этом режиме каждый узел имеет собственную очередь запросов длиной N/2+1 и полную копию БД. В одном узле может одновременно происходить M операций SELECT-PROJECT и N операций JOIN. Для балансировки нагрузки между узлами используется алгоритм роутеризации [Райхлин В.А., Минязев Р.Ш. Мультикластеризация распределенных СУБД консервативного типа].


Сеть Ethernet
Узел 1
Узел 2
Узел 3
Узел 4
Mgm
БД
БД
БД
БД
CPU
CPU
CPU
CPU
CPU
GPU
GPU
GPU
GPU

Множество узлов кластера на 1 запрос

Для организации работ с базами данных в сотни GB, можно попытаться развить стратегию СУБД Clusterix при существенно меньшей производительности по сравнению с предыдущим случаем. Исполняющие узлы реализуют только функции IO. Их работой полностью управляет управляющий узел: раздает задание, управляет передачей промежуточных данных, балансирует нагрузку между узлами, получает частичные SELECT-PROJECT - результаты



Отличия от СУБД Clusterix

- функции модуля IO здесь выполняет графический ускоритель, а узловые CPU занимаются только передачей сжатых блоков в GPU, получением от них результатов и перенаправлением их в узел Mgm;
- данная СУБД асинхронная, в то время как Clusterix – синхронная;
- операции JOIN выполнятся на CPU управляющего узла;
- очередь запросов находится на управляющем узле.
- после выполнения операций SELECT-PROJECT и передачи данных в операцию JOIN, система начинает обрабатывать следующий запрос. Запросы поступают до того момента как будут занят все процессорные ядра доступные для операции JOIN.

Обновление данных в БД

- полное перестроение БД. Это вполне возможно, т.к. БД консервативного типа обновляются редко, но в случае больших БД (несколько сотен гигабайт и более) такое перестроение может занимать очень длительное время.
- использовать специальный индексный файл для помечания удаленных и обновленных строк, добавление строк производить в конец последнего блока. В этом случае индексный файл работает следующим образом:
  - в случае удаления – добавляется запись о игнорировании записи в БД
  - в случае обновления – добавляется запись о игнорировании записи в БД и добавляется новая запись в конец последнего блока.

Организация хранения и обработки сжатых данных

2 типа СУБД

- СУБД ориентированные на хранение данных по строкам
- СУБД ориентированные на хранение данных по столбцам

Лучшие показания по сжатию возможно получить только в СУБД ориентированных на хранение данных по столбцам. Для этого есть ряд причин: во-первых, сжатие применяется для заранее определённого типа данных; во-вторых, для каждого столбца можно подобрать оптимальный алгоритм сжатия, который покажет наибольшую эффективность именно для данных в заданном столбце

Сжатие/разжатие данных на GPU

Данный вопрос уже исследован в [Wenbin F., Bingsheng H., Qiong L. Database Compression on Graphics Processors], где рассмотрено 5 алгоритмов сжатия и 4 вспомогательные схемы. Вспомогательные схемы позволяют добиться более высокого коэффициента сжатия при применении их перед применением алгоритма сжатия. Например, пусть дана последовательность 1,2,3,4,5, тогда при сжатии алгоритмом RLE получим удвоение исходного количества данных ([1,1], [2,1], [3,1], [4,1], [5,1]), однако если сначала применить вспомогательную схему DELTA (разность следующего и текущего элемента последовательности), то получим последовательность 1,1,1,1,1, которая успешно сжимается алгоритмом RLE: [1,5]. Т.е. для последовательности 1,2,3,4,5 выгодно применять цепочку алгоритмов: DELTA, RLE.

Алгоритмы сжатия

№ п/п	Обозначение	Название	Описание
1	NS	Null Suppression with Fixed Length	Сжатие путем отбрасывания впереди идущих неиспользуемых нулей (0x00000FFF -> 0x0FFF, int -> short)
2	NSV	Null Suppression with Variable Length	Сжатие путем отбрасывания впереди идущих неиспользуемых нулей с возможностью кодировать каждое значение индивидуально
3	DICT	Dictionary	Создание словаря значений и использование ссылок на него
4	BITMAP	Bitmap	Кодирует каждое значение как битовую строку. Подходит для малого количества уникальных значений в столбце
5	RLE	Run-Length-Encoding	Заменяющий повторяющиеся серии символов на один символ и число его повторов

Вспомогательные схемы для сжатия

№ п/п	Обозначение	Название	Описание
1	FOR	Frame-Of-Reference	Преобразует каждое значение как разность с базовым значением (как правило, минимальным значением в последовательности)
2	DELTA	Delta	Преобразует каждое значение как разность текущего и следующего элемента последовательности
3	SEP	Separate	Разделение поля на части с более простыми типами данных
4	SCALE	Scale	Преобразование нецелочисленных значений в целочисленные путем умножения на 10^n, где n – количество сохраняемых знаков после запятой.

Выбор алгоритма сжатия для столбца

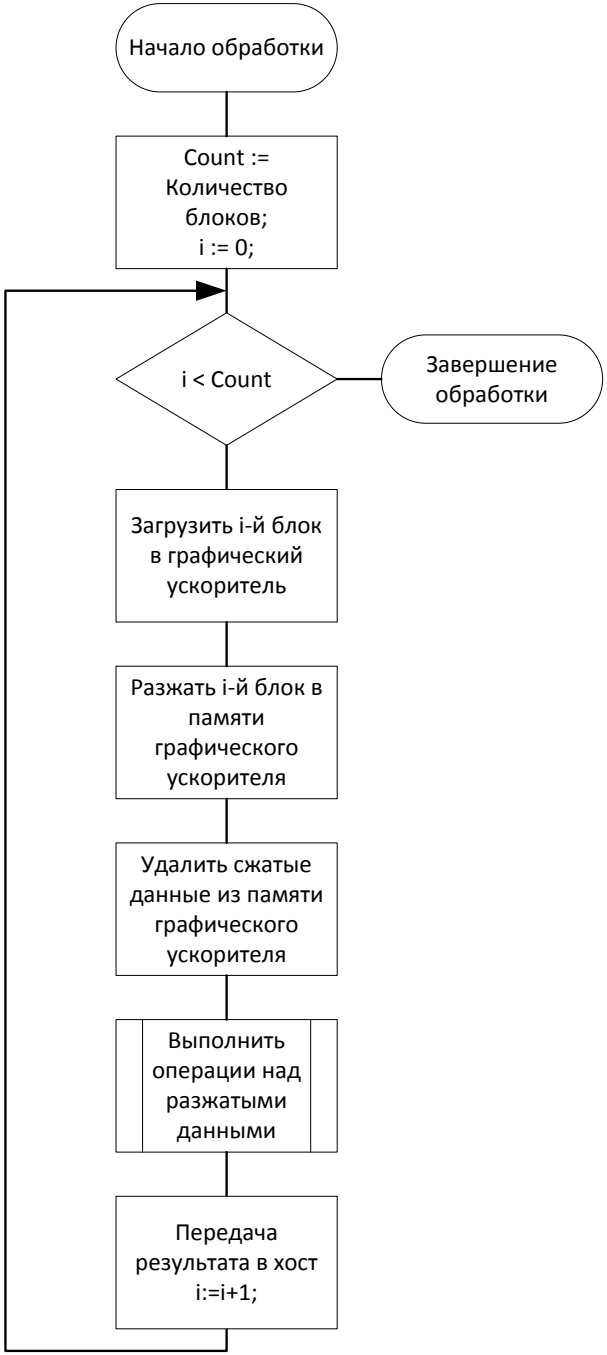
Выбор алгоритма сжатия в [Wenbin F., Bingsheng H., Qiong L. Database Compression on Graphics Processors] для той или иной колонки осуществляется с помощью планировщика по ее параметрам:

- наличие сортировки,
- средняя длина повторяющихся подряд значений,
- количество уникальных значений,
- максимальная длина значения в байтах,
- возможность дробления значений столбца на более простые значения (например, для поля даты хранить день, месяц, год отдельно).

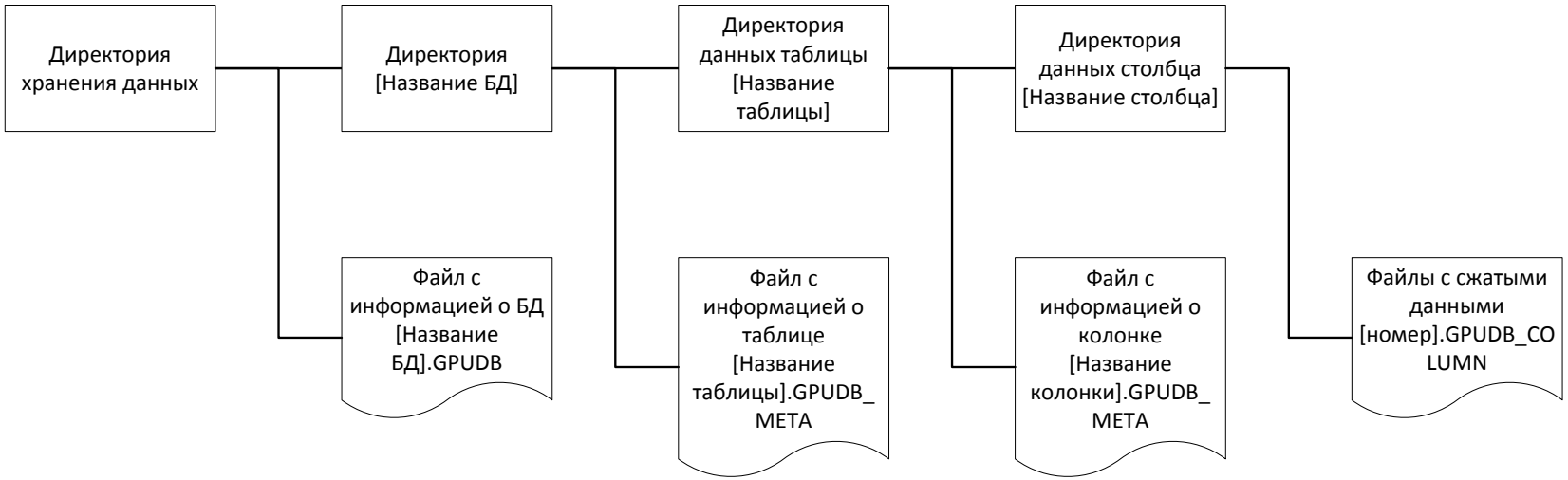
Причем осуществляется выбор сразу цепочки алгоритмов сжатия, применение которой дает достижение максимального коэффициента сжатия.

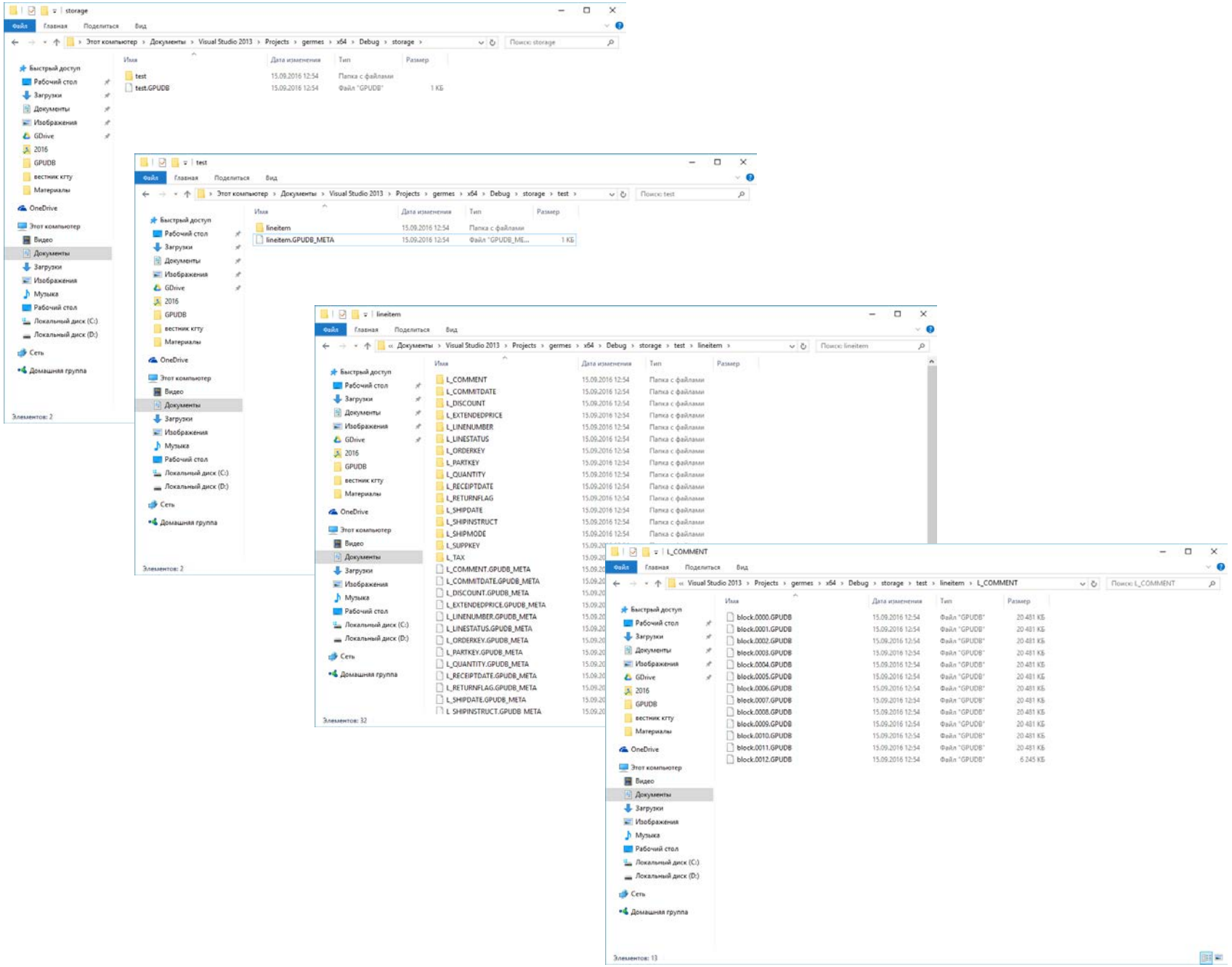
Разбиение на блоки

Разбиение на блоки позволяет итерационно (блок за блоком) обрабатывать объем информации, превышающий объем памяти графического ускорителя как показано на рисунке. При этом необходимо учитывать объемы разжатых и сжатых данных, т.к. они будут находится в памяти графического ускорителя одновременно. Поскольку в худшем случае данные не будут сжаты, то верным предполагается разделение памяти графического ускорителя на 2 равные части: часть сжатых данных и часть распакованных данных. В случае такого разделения данные загружаются в первую область, распаковываются во вторую. После распаковки данные в первой области уничтожаются и область готова к приему нового блока данных. Такое разделение памяти позволяет загружать порцию сжатых данных во время обработки распакованных данных и организовать конвейерную обработку.


Начало обработки
Count := Количество блоков; i := 0;
i < Count
Завершение обработки
Загрузить i-й блок в графический ускоритель
Разжать i-й блок в памяти графического ускорителя
Удалить сжатые данные из памяти графического ускорителя
Выполнить операции над разжатыми данными
Передача результата в хост
i:=i+1;

Хранение данных




storage
test
test.GPUDB
test
lineitem
lineitem.GPUDB_META
lineitem
L_COMMENT
L_COMMITDATE
L_DISCOUNT
L_EXTENDEDPRICE
L_LINENUMBER
L_LINESTATUS
L_ORDERKEY
L_PARTKEY
L_QUANTITY
L_RECEIPTDATE
L_RETURNFLAG
L_SHIPDATE
L_SHIPINSTRUCT
L_SHIPMODE
L_SUPPKEY
L_TAX
L_COMMENT.GPUDB_META
L_COMMITDATE.GPUDB_META
L_DISCOUNT.GPUDB_META
L_EXTENDEDPRICE.GPUDB_META
L_LINENUMBER.GPUDB_META
L_LINESTATUS.GPUDB_META
L_ORDERKEY.GPUDB_META
L_PARTKEY.GPUDB_META
L_QUANTITY.GPUDB_META
L_RECEIPTDATE.GPUDB_META
L_RETURNFLAG.GPUDB_META
L_SHIPDATE.GPUDB_META
L_SHIPINSTRUCT.GPUDB_META
L_COMMENT
block.0000.GPUDB
block.0001.GPUDB
block.0002.GPUDB
block.0003.GPUDB
block.0004.GPUDB
block.0005.GPUDB
block.0006.GPUDB
block.0007.GPUDB
block.0008.GPUDB
block.0009.GPUDB
block.0010.GPUDB
block.0011.GPUDB
block.0012.GPUDB

Применение сжатия для передачи информации в графический ускоритель

Сжатие как способ увеличения эффективности передачи информации в GPU

Компенсировать малую пропускную способность шины возможно с помощью применения сжатия к передаваемым данным. Для проверки данной гипотезы был проведен эксперимент с целью сравнения времени, затрачиваемого на простое копирование, с временем необходимым для копирования данных, сжатых по алгоритму RLE (Run Length Encoding), и их разжатия.

Суть алгоритма заключается в замене подряд идущих повторяющихся символов/чисел на один символ/число и число его повторов. Например, числовой ряд «1,1,2,2,2,3,4,4,4», состоящий из 9 чисел, будет закодирован в ряд «1,2,2,3,3,1,4,3», состоящий из 8 чисел, где выделенные числа – количество повторений предыдущего числа.

Постановка эксперимента

Эксперимент проводился на системе с конфигурацией: CPU – Core i5 4670K (4 ГГц), GPU – Nvidia GTX 770 (2 Гб GDDR5), RAM – 24 Гб (DDR3-1600 в двухканальном режиме).

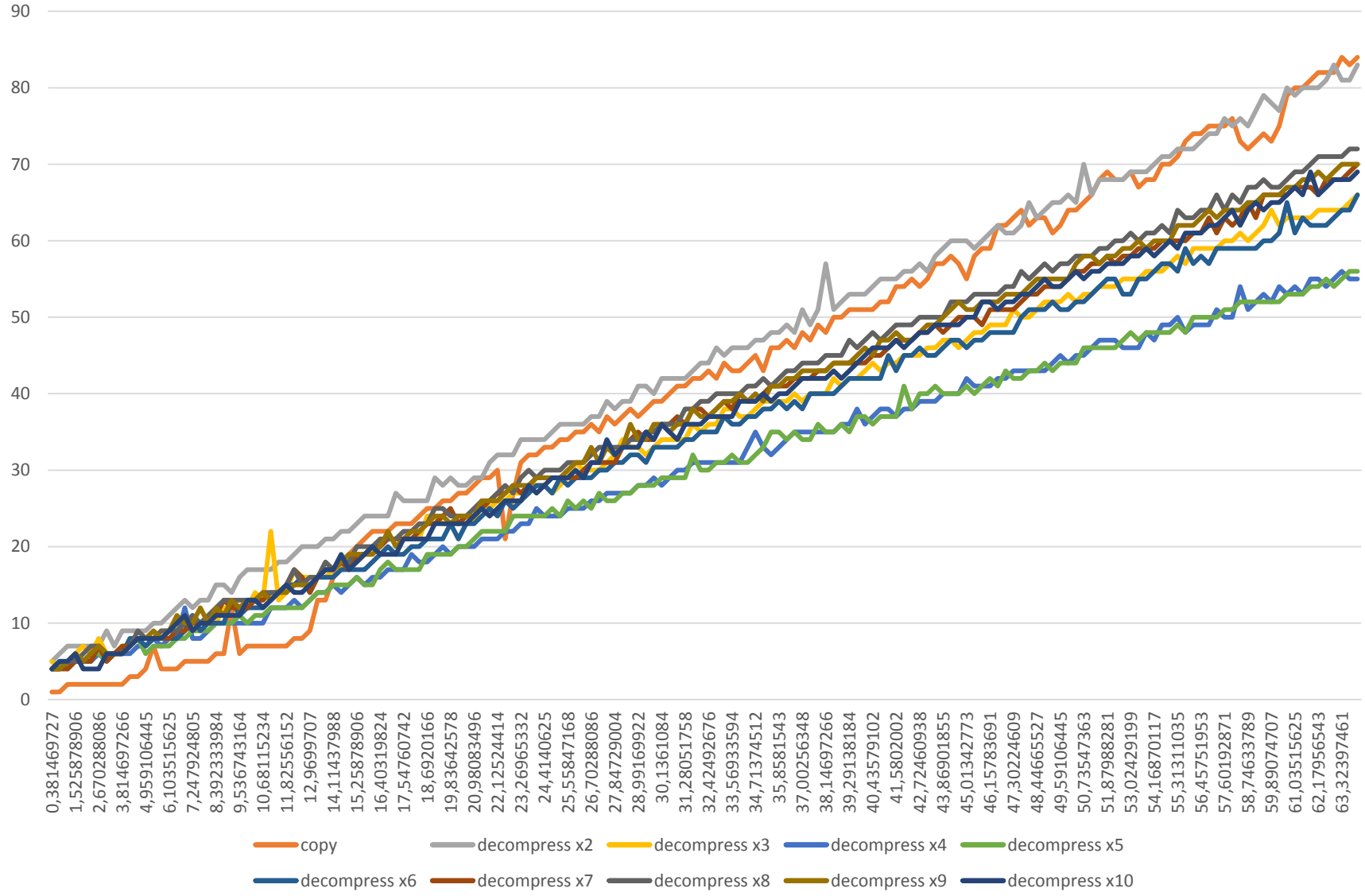
Данными для проведения эксперимента являются наборы рядов целых чисел, длиной L от 100000 (0,38 Мбайт) до 16800000 (64,09 Мбайт) с шагом 100000 (0,38 Мбайт) элементов.

Подготовка данных для эксперимента

Каждый набор генерировался таким образом, чтобы достигался заданный коэффициент сжатия по алгоритму RLE. На каждой итерации создавался новый набор, состоящий из 9 рядов сжатых данных с коэффициентами сжатия K от 2 до 10 и один несжатый ряд. Сжатые ряды представляют собой числовые массивы вида «1,2,2,3,3,2,4,2», где выделенные числа равны коэффициенту сжатия набора K, а не выделенные - инкрементируются до достижения значения равного L/K.

Несжатый ряд A формируется по формуле $A_i = A_{i-1} + 1$, где $i = \overline{2, L} \quad A_0 = 1$

Результаты эксперимента



Время копирования в память GPU и восстановления сжатой информации

Размер (Мбайт)	Копирование, мс	Копирование и восстановление сжатой информации, мс								
		x2	x3	x4	x5	x6	x7	x8	x9	x10
0,38	1	5	5	4	4	4	4	4	4	4
4,20	3	9	7	6	8	8	7	7	7	7
8,01	5	13	10	9	9	10	10	11	10	10
11,83	7	18	14	12	12	15	15	15	14	15
15,64	21	24	19	15	15	17	19	20	19	19
19,45	26	28	23	20	19	21	24	25	24	23
23,27	31	34	27	23	24	26	27	29	28	26
27,08	35	37	30	26	27	30	31	33	31	31
30,90	41	42	34	30	29	33	37	36	36	34
34,71	45	47	38	35	32	37	39	41	40	39
38,53	50	51	42	35	35	40	44	45	44	43
42,34	55	56	45	38	38	45	47	49	47	47
46,16	59	61	49	41	42	48	51	53	52	52
49,97	64	66	53	44	44	51	55	57	55	55
53,79	68	69	56	48	48	55	59	61	59	59
57,60	75	76	60	50	51	59	63	64	64	63
61,42	80	80	63	53	53	63	67	69	68	66

Увеличение эффективности передачи данных

Размер (Мбайт)	Копирование, мс	Копирование и восстановление сжатой информации, мс		Увеличение эффективности передачи, %	
		x4	x5	x4	x5
0,38	1	4	4	-300%	-300%
4,20	3	6	8	-167%	-100%
8,01	5	9	9	-80%	-80%
11,83	7	12	12	-71%	-71%
15,64	21	15	15	29%	29%
19,45	26	20	19	27%	23%
23,27	31	23	24	23%	26%
27,08	35	26	27	23%	26%
30,90	41	30	29	29%	27%
34,71	45	35	32	29%	22%
38,53	50	35	35	30%	30%
42,34	55	38	38	31%	31%
46,16	59	41	42	29%	31%
49,97	64	44	44	31%	31%
53,79	68	48	48	29%	29%
57,60	75	50	51	32%	33%
61,42	80	53	53	34%	34%

Заключение

Сжатие данных позволит не только более эффективно использовать графические ускорители для применения в консервативных СУБД, но хранить больше данных на тех же носителях. Предложенная организация СУБД на графических ускорителях учитывает особенности работы шины PCI-e и графических ускорителей фирмы Nvidia. Ожидается, что реализация изложенной концепции при использовании стратегии «1 узел на N/2 + 1 запросов», где N – число ядер в узле, может дать прирост производительности на потоке запросов не менее чем в 1,5 раза по сравнению с достигнутым в [Классен Р.К. Повышение эффективности параллельной СУБД консервативного типа на кластерной платформе с многоядерными узлами], т.е примерно на 33%, что существенно. Для варианта Clusterix предполагается не меньший выигрыш при существенном росте объемов обрабатываемых баз данных.