Atricle ID:1000-5641(2014)05-0240-12

Co-OLAP: Research on cooperated OLAP with star schema benchmark on hybrid CPU&GPU platform

ZHANG Yu[1,2], ZHANG Yan-song[1,2,3], ZHANG Bing[1,2], CHEN Hong[1,2], WANG Shan[1,2]

(1. *DEKE Lab*, *Renmin University of China*, *Beijing* 100872, *China*;
2. *School of Information*, *Renmin University of China*, *Beijing* 100872, *China*;
3. *National Survey Research Center at Renmin University of China*, *Beijing* 100872, *China*)

Abstract: Nowadays GPUs have powerful parallel computing capability even for moderate GPUs on moderate servers. Opposite to the recent research efforts, a moderate server may be equipped with several high level CPUs and a moderate GPU, which can provide additional computing power instead of more powerful CPU computing. In this paper, we focus on Co-OLAP (Cooperated OLAP) processing on a moderate workstation to illustrate how to make a moderate GPU cooperate with powerful CPUs and how to distribute data and computation between the balanced computing platforms to create a simple and efficient Co-OLAP model. According to real world configuration, we propose a maximal high performance data distribution model based on RAM size, GPU device memory size, dataset schema and special designed AIR(array index referencing) algorithm. The Co-OLAP model distributes dataset into host and device memory resident datasets, the OLAP is also divided into CPU and GPU adaptive computing to minimize data movement between CPU and GPU memories. The experimental results show that two Xeon six-core CPUs slightly outperform one NVIDA Quadra 5 000 GPU with 352 cuda cores with SF = 20 SSB dataset, the Co-OLAP model can assign balanced workload and make each platform simple and efficient.

Key words: GPU; OLAP; Co-OLAP; AIR

CLC number: TP393 Document code: A **DOI**:10.3969/j.issn.1000-5641.2014.05.021

Co-OLAP: CPU&GPU 混合平台上面向星形模型基准的协同 OLAP

张 宇[1,2], 张延松[1,2,3], 张 兵[1,2], 陈 红[1,2], 王 珊[1,2]

(1. 中国人民大学 DEKE 实验室, 北京 100872; 2. 中国人民大学 信息学院,北京 100872;

收稿日期:2014-06

基金项目:中央高校基本科研业务费专项资金(12XNQ072, 13XNLF01)

第一作者:张宇,女,博士研究生,研究方向为 GPU、OLAP. E-mail: zyszy511@hotmail.com.

通信作者:张延松,男,博士后,讲师,研究方向为内存数据库、OLAP. E-mail: zhangys_ruc@hotmail.com.

3. 中国人民大学 中国调查与数据中心,北京　100872)

摘要: 当前 GPU(图形处理器),即使是中端服务器配置的中端 GPU 也拥有强大的并行计算能力.不同于近期的研究成果,中端服务器可能配置有几块高端 CPU 和一块中端 GPU,GPU 能够提供额外的计算能力而不是提供比 CPU 更加强大的计算能力.本文以中端工作站上的 Co-OLAP(协同 OLAP)为中心,描述如何使中端 GPU 与强大的 CPU 协同以及如何在计算均衡的异构平台上分布数据和计算以使 Co-OLAP 模型简单而高效.根据实际的配置,基于内存容量,GPU 显存容量,数据集模式和订制的 AIR(数组地址引用)算法提出了最大高性能数据分布模型. Co-OLAP 模型将数据划分为驻留于内存和 GPU 显存的数据集,OLAP 计算也划分为 CPU 和 GPU 端的自适应计算负载来最小化 CPU 和 GPU 内存之间的数据传输代价.实验结果显示,在 SF = 20 的 SSB(星形模型基准)测试中,两块至强六核处理器的性能略优于一块 NVIDA Quadra 5 000 GPU(352 个 cuda 核心)的处理性能, Co-OLAP 模型可以将负载均衡分布在异构计算平台并使每个平台简单而高效.

关键词: GPU(图形处理器); OLAP(联机分析处理); Co-OLAP(协同 OLAP); AIR(数组地址引用)

0　Introduction

Databases severely suffer from performance issues for the data- and computing-intensive workloads such as data warehousing and OLAP. Performance-critical designed IMDBs (In-memory Database) such as MonetDB, Vectorwise, HANA, Hyper, IWA, have gained orders of magnitude improvement than the traditional disk resident databases, but performance is still the most critical issue for a big in-memory dataset due to memory bandwidth bottleneck and limited processing cores. Nowadays graphic processing units (GPUs) have been designed to be powerful parallel computing processors with much more cores than CPUs (thousands of cores vs. tens of cores), and GPU memory has much higher memory bandwidth than DRAM(300 + GB/s vs. 80 GB/s). The parallel programming models, such as CUDA and OpenCL further push GPUs as practical candidate for parallel computing-intensive engine beyond multicore CPUs. Database community has made significant approaches in optimizing query processing on GPUs in recent years[1-13], co-processor framework becomes main stream in high performance computer. Even moderate computers are commonly configured with powerful GPU which can not only provide graphic processing but also additional computing power.

In this paper, we focus on a case study on a moderate workstation configuration with two Xeon 6-core CPUs(12 cores) and one NVIDA Quadra 5 000 GPU(11 multiprocessors, 352 cuda cores) for standard SSB OLAP optimizations. On data-intensive OLAP workload, we find that the processing capability relies more on amount of multiprocessors than amount of streaming cores. The Xeon CPU platform outperforms GPU platform in our workstation, and it is common scenario for typical moderate servers. So it is impossible to use GPU as major accelerator as some researches (commonly configured with low level

CPU and high level GPU), we have to rethink the hybrid CPU/GPU computing framework. we first design an array oriented storage and processing engine for cooperated CPU and GPU computing, AIR (array index referencing) algorithm to replace traditional hash join operator to make star-join efficient both on CPU and GPU platforms, then we assign GPU as additional star-join engine and CPU as predicate processing and aggregating engines to cooperate for the whole OLAP processing.

The related work are presented in Sec. 1. Co-OLAP is discussed in Sec. 2. Sec. 3 shows the results of experiments. Finally, Sec. 4 summarizes the paper.

1 Related Work

GPUs are employed as co-processors for CPUs[1,2], GDB[2] gives an architecture of relational query Co-Processing on graphic processors. According to cost model for CPU and GPU, workload is assigned to different processors. Relational operators are optimized for GPU processing such as GPU joins[3,4], GPU hashing[5], GPU compression[6,7], GPU sorting[8,9], GPU memory contention[10], data movement optimizations[11], GPU transaction execution[12], GPU cost model[13], etc. These significant efforts have verified that GPUs are good candidate for high performance relational processing due to massive parallel cores. The major differences between GPU and CPU focus on two aspects: (1) CPU optimizations are cache-conscious designed, the key idea is to fully utilize cache hierarchy to make frequent dataset closer to the cores, the LLC(last level cache) becomes larger and larger with increasing cores; GPU equips with the small size of shared memory(32 KB-48 KB), without hardware shared memory management like cache, GPU optimizations need more programming techniques, and the small size of shared memory is difficult to optimize strong data locality processing like hash join; GPU commonly relies on hardware threading mechanism to overlap device memory accessing latency while CPU majorly relies on large cache. (2) nowadays PCIe bandwidth is much lower than memory bandwidth, we must either improve PCIe transmission performance (e. g., multiple memory channels, DMA (Direct Memory Access), pinned memory transmission[4]) or data distribution to make computing-intensive workload GPU memory resident.

For OLAP workload, the key issue is the star-join performance. [14,15] discussed the performances of different multicore parallel hash join algorithms, the results show that simple nopartitioned hash algorithm is adaptive to star schema with skewed dataset between fact table and dimension tables. Invisible-join[16] can even improve star-join efficiency by using bitwise operation. Current GPU efforts commonly rely on hash structures for hash join or hash aggregation, while managing hash structure in GPU is not as efficient as in CPU. So GPU OLAP algorithms rely on two main considerations, one is to choose high performance hash join algorithms, the other is to tune algorithms to be adaptive to GPU's memory management and hardware architecture. One common barrier is that memory efficient hash table needs pointers to manage dynamic chained buckets, while it is less effi-

cient in GPU for fixed memory management. DDTA-JOIN[17] is a tailored join algorithm by using foreign key columns as native join indexes for the star-joins. Hash tables are removed from OLAP processing, so they are more adaptive to be used in GPU.

In general, desktop GPU outperforms CPU. But for moderate server, multi-way CPUs may outperform GPU in typical configuration. For balanced CPU and GPU platforms, it is not necessary for GPU to realize all the query processing operators, GPU acts as an additional computing engine instead of a major computing engine, we should focus on how to distribute data and computing in CPU and GPU memories and cooperate the computing of each platform.

2 Co-OLAP Designs

Co-OLAP model is designed for a distributed OLAP model for GPU and CPU platforms, data distribution strategy is processor-conscious to guarantee minimal data movement overhead and make computing adaptive to processor features.

2.1 Data distribution Model

Star schema is a formula schema for data warehousing, and star schema is widely studied by commercial databases and academic researches. We focus on star schema optimizations for Co-OLAP in this paper, and snow-flake schema like TPC-H will be studied in future work.

1. Data distribution of SSB

SSB(star schema benchmark) comprises one fact table and four dimension tables, fact table comprises four foreign key columns and thirteen measure columns. Dimension tables are small in sizes with predicate processing on various data types, foreign key columns in fact table are relative small but frequently accessed for multidimensional queries(star-joins), and measure columns may be very large in enterprise dataset, but each multidimensional query commonly locates very small dataset (aggregate on dataset with very low multidimensional selectivity and few measure attributes). So we can first consider the small dimension tables and foreign key columns to be GPU memory resident candidates.

Dimension tables are small in sizes but with various data types, complex data type management is adaptive to CPU. Moreover, dimension tables involve many update operations, CPU can do updates more efficient than GPU. According to these considerations, we only assign foreign key columns GPU memory resident for the simple data types and computing-intensive workload of star-join.

2. Data access latency of data distribution

Fig. 1 illustrates a workstation configuration. The device memory of GPU is 2.5 GB, the main memory is 12 GB, the memory bandwidthes of CPUs are 51.2 GB/s (four memory channels with each 12.8 GB/s bandwidth), the bandwidth of GPU device memory is 120 GB/s, and the PCIe bandwidth is maximal 6 GB/s with pinned memory access. The local memory accesses are efficient for both GPU and CPU, but data movement between

host memory and device memory is quite slow. So we must minimize data movement between CPU and GPU during query processing. We design a vector oriented OLAP algorithm to reduce PCIe transmission overhead by only swapping small dimension vectors and fact vectors. The detailed design will be illustrated in the following sections.

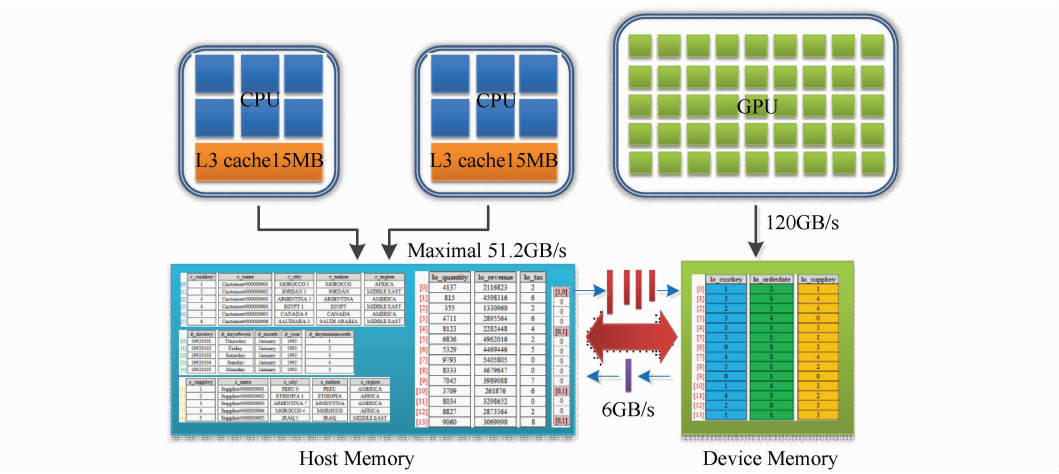

Fig. 1 Data distribution

3. Maximal high performance data distribution model

In ideal data distribution model, all the foreign key columns can be GPU memory resident for CPU bound star-join processing, CPU only responses for predicate processing and aggregation. To gain this target, GPU memory should contain all the foreign key columns, so we can model the maximal high performance dataset size as:

$S = (S_{GPU} - S_{DimVecs} - S_{FactVec}) / R_{FKs/Dataset}$, in which,

S denotes the total dataset size with GPU memory resident computing;

$R_{FKs/Dataset}$ denotes the proportion of foreign key columns size in whole dataset;

S_{GPU} denotes the size of GPU device memory;

$S_{DimVecs}$ denotes the size of a dimension vector;

$S_{FactVec}$ denotes the size of a fact vector.

With this data distribution model, we can support maximal S dataset to perform completely cooperated computing with CPU and GPU according to specified GPU memory size. If we have to deal with larger dataset, we can upgrade hardware platform by add more GPUs or use higher level GPU with larger GPU memory.

2.2 Array store and AIR algorithm

Storing columns in arrays is widely adopted in main-memory databases(e.g., MonetDB) and GPU database researches. We adopt array store technique, the dimensional columns and measure columns are stored as page-able arrays, the foreign key columns are stored as pinned arrays to accelerate the transmission performance through PCIe bus.

For dimension table, array index can be considered as virtual primary key(VPK). If a foreign key is assigned with dimensional array index, the join between foreign key column

and primary key column can be simplified as foreign key directly accessing dimensional item with stored dimensional array index. We define this join as Array Index Referencing (AIR). This new operator requires an additional constraint for star schema that primary key of dimension table must be defined as 1,2,3… which is widely used in SSB and TPC-H as default (such as *part*, *supplier*, *customer*, *date* (the 19920101, 19920102, … format primary keys can be simply used as array index by current date minus the first date) dimension tables in SSB and *part*, *supplier*, *customer* dimension tables in TPC-H). We can also update foreign key columns for existed dataset as an ETL process.

Invisible-join[14] is a column based OLAP algorithm like MonetDB with improvement on star-join by bitmap bitwising operator, and invisible-join algorithm is also adopted by[13] GPU algorithm. To make AIR algorithm comparable with invisible-join, we inherit the query example style algorithm description like[17], we will illustrate how AIR works with Q3.1 from star schema benchmark, and compare the detailed process stages with invisible-join.

```
SELECT c.nation, s.nation, d.year, sum(lo.revenue) as revenue
FROM customer AS c, lineorder AS lo,supplier AS s, dwdate AS d
WHERE lo.custkey = c.custkey
    AND lo.suppkey = s.suppkey
    AND lo.orderdate = d.datekey
    AND c.region = 'ASIA'
    AND s.region = 'ASIA'
    AND d.year >= 1992 and d.year <= 1997
GROUP BY c.nation, s.nation, d.year
ORDER BY d.year asc, revenue desc;
```

In this query, dimension tables response for providing predicate filters (opposite to hash tables in[16]) and groups for aggregation. For Q3.1, the first stage is to apply predicates and *GroupBy* clauses on dimension tables to generate star-join filters. In Fig.3, invisible-join only applies predicates on dimension tables and uses hash table as star-join filters, the *GroupBy* clauses are processed in the end, so dimension tables are accessed twice in the whole query processing. AIR uses vector as an early-materialized grouping filter. According to predicates on dimension table, filtered *GroupBy* keys are organized as array dictionaries. In dimension vector, positions 1 and 3 are filled with *GrpCode* array indexes, position 2 does not satisfy the predicate and is filled with 0. The dimension vector is small (length of dimension table rows) even for large dataset, foreign key can directly access dimension vector to probe whether current fact tuple can go through dimension filter. Moreover, we can pre-generate a multidimensional array $Agg[D_1]\cdots[D_n]$ as *GroupBy* container, where $D_i(1<=i<=n)$ denotes the cardinality of each dimensional array dictionary in each dimension vector. For example, Fig.2 can use $Agg^{[2][1][1]}$ for aggregation.

In star-join stage, invisible-join performs hash table oriented column joins and uses bitmaps as join results, finally a bitwise AND operator is invoked for star-join result.

There are two important issues we should pay attentions: one issue is that OLAP query commonly has high selectivity(maximal 6/7 in SSB opposite to Fig. 2 example with very few filtered dimensional tuples) in single dimension table, hash join between foreign key column and dimension table has high overhead; the other issue is that bitwise operation overhead for big bitmaps is also high and bitmaps consume large space for a big dataset.

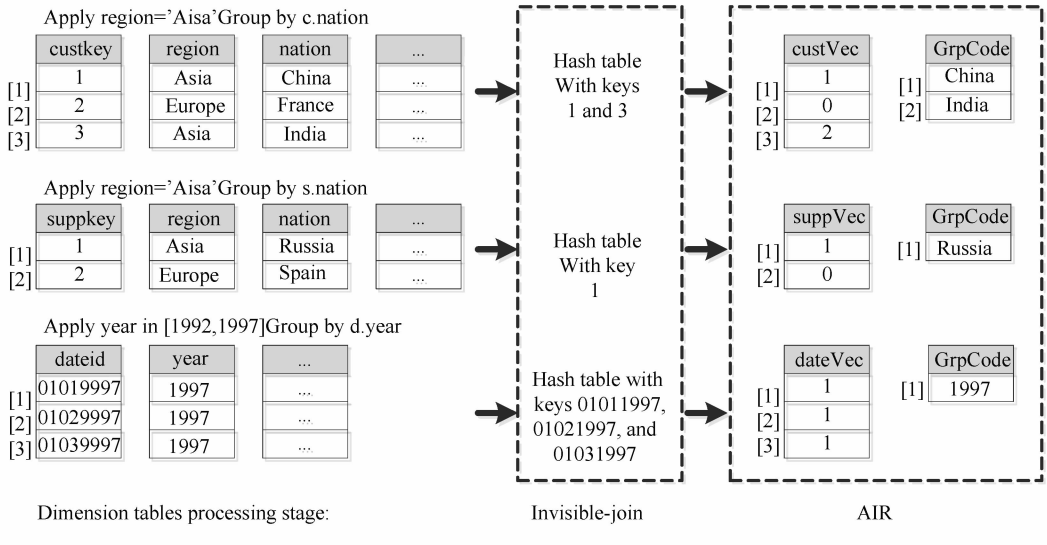

Fig. 2 The first stage of predicate processing on sample data

In Fig. 3, AIR algorithm makes star-join even simple and efficient. Assume that *orderdate* column can be on-the-fly converted with array index of *date* table. We use a fact vector as star-join filter. As scanning *custkey* column, each key is mapped to *custVec* to update relative fact vector item with *custkey* value. For *suppkey* column, we perform a positional scan according to fact vector's non-zero positions, and then updating fact vector

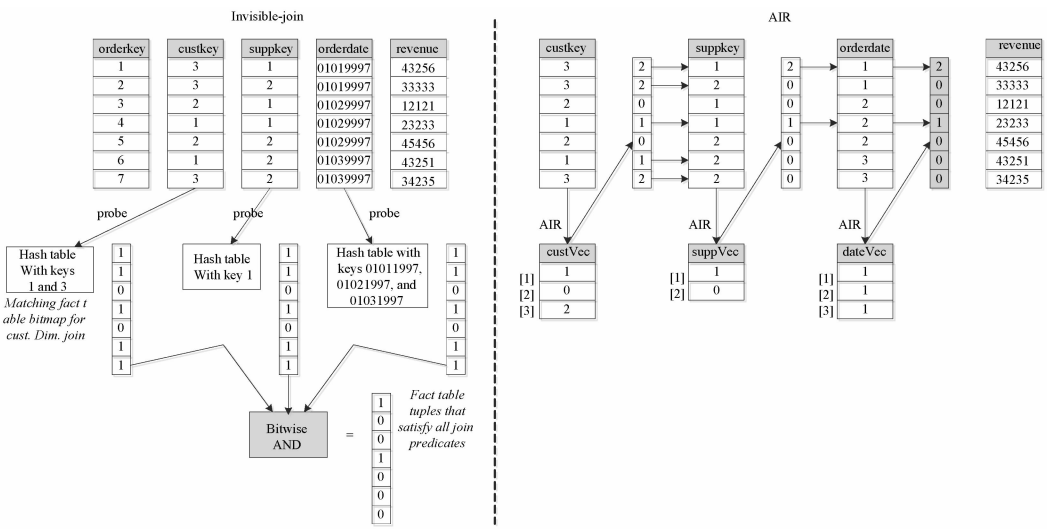

Fig. 3 The second stage of star-join on sample data

according to *suppVec* mapping. The fact vector is iteratively updated with *GroupBy* multi-dimensional array indexes(mapping 3-D array index to 1-D array index), when finishing all the foreign key scans, the fact vector can identify which fact rows are to be output for aggregation and the aggregation unit address for each fact tuple.

With AIR algorithm, hash tables are replaced with simple vectors, array index referencing on small dimension vectors is very efficient for cache locality.

Column store commonly employs late-materialization strategy. Invisible-join has to access foreign key columns twice, one for foreign key join, one for extracting final foreign key and joining with dimensional column for *GroupBy* values as shown in Fig. 4.

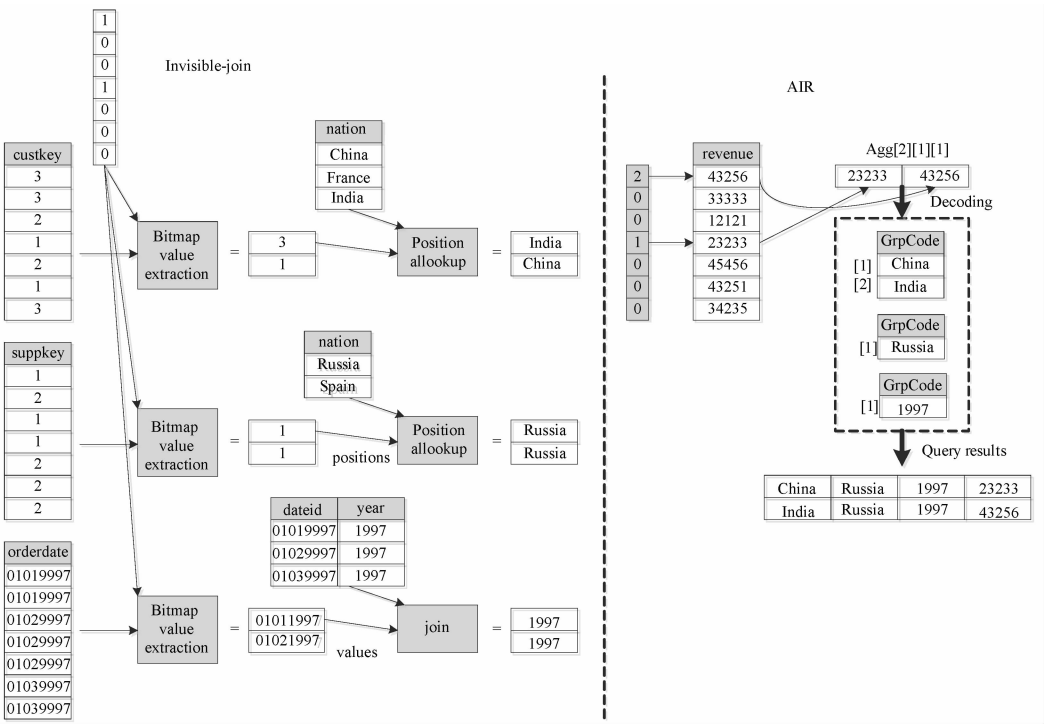

Fig. 4　The third stage of aggregation on sample data

Opposite to invisible-join, AIR employs early- materialization strategy. In dimension processing stage, *GroupBy* values are encoded into dimension vectors and pre-generate multidimensional array for final aggregation. The star-join stage iteratively refreshes fact vector with pre-assigned multidimensional array addresses, so the aggregation stage can be independently performed without accessing dimensional columns or foreign key columns again. Fig. 4 shows the aggregation stage of AIR, a positional lookup according to fact vector is performed on measure columns, the value is extracted and pushed to aggregation array $Agg[2][1][1]$ for aggregating. Finally, aggregation array is converted into aggregating results by mapping array indexes to dimension array dictionary to extract each grouping attributes.

In AIR algorithm, a standard OLAP query processing is divided into three independ-

ent stages, the *input* and *output* data are all small. Star-join is a computing- intensive workload with fixed columns and can be assigned to GPU; aggregation involves large dataset but simple processing, this data-intensive workload is CPU adaptive.

The Co-OLAP model can be illustrated in Fig. 1. All the data are memory resident, and foreign key columns are GPU memory resident. CPU processes query and generates dimension vectors, the small vectors are transferred to GPU memory through PCIe bus, GPU performs star-join with dimension vectors and generates fact vector, the fact vector is transferred back to CPU memory, then CPU performs a positional scan on measure columns for aggregation.

For CPU programming, we can use a dynamic *vector* template class as fact vector to only store filtered fact tuple positions and *GroupBy* addresses. For high selective queries, the dynamic *vector* oriented fact vector needs less space and avoids scanning the whole fact vector. For GPU programming, we maintain a pre-allocated pinned array as a fact vector. Star-join is multiple-pass vector processing between equal length arrays, and it is adaptive to be programmed with CUDA. We did not use dynamic *vector* like CPU because GPU is less efficient than CPU in dynamic memory management, fixed length fact vector can satisfy all the queries with different selectivity. Star-join in CPU has better code efficiency than in GPU, but GPU has much more processing cores and higher device memory bandwidth than CPU, the overall performance is convincing.

3 Case Studies

In this paper, we design experiments in a moderate workstation to exploit how to maximize the hardware performance. Our experiments are conducted on a Lenovo workstation with two IntelR XeonR Processor E5-2667 (6-core, 15M Cache, 2.90 GHz), 12 GB memory, maximal memory bandwidth 51.2 GB/s, PCIe channel bandwidth 6 GB/s with pinned memory. The GPU type is NVIDA Quadro 5000, the configurations are as follows: 352 cuda cores(11 multiprocessors), 2.5 GB GDDR5, 320 bit bus width, 120 GB/s bandwidth. The prices of two CPUs and one GPU are equal (~20,000 RMB). The OS version is ubuntu-12.04 (precise)64-bit, kernel Linux 3.8.0-29-generic, CUDA version is 5.5. We use star schema benchmark(SSB) dataset of SF = 20 with standard SSB data generator.

3.1 GPU memory resident

For Co-OLAP model, the foreign key columns are GPU memory resident, small dimension vectors are on-the-fly transferred from host memory to device memory for each ad-hoc query, and only fact vector needs to be transferred from device memory to host memory. The total size of four foreign key columns plus one fact vector is about 2.39GB, 95.6% GPU memory is utilized. The GPU memory is maximal utilized and we can support maximal SF = 20 dataset for GPU memory resident Co-OLAP.

3.2 CPU memory resident

The average predicate processing time is 10.24 ms in CPU, dimension vectors are transferred from host to device memory with average 0.23 ms under 6GB/s pinned transfer bandwidth. In star-join stage, each foreign key column is parallel scanned with AIR algorithm on dimension vectors and updating fact vector. The star-join execution time(average 73% in total execution time) is influenced by dimension vector size, selectivity and amount of foreign key columns. The fact vector is transferred from device to host memory with about 20.89 ms. With the fact vector, aggregation is executed efficiently in CPU on large measure columns with average 5.33 ms. In the whole Co-OLAP processing stages, star-join is computing-intensive workload on GPU memory resident foreign key columns. For big dataset, dimension vectors usually exceed the small shared memory size(48 KB), array index referencing involves many global memory access latency. In general, shared memory can hardly contain strong locality dataset such as dimension vectors or hash table, GPU's SIMT(Single Instruction Multiple Threads) mechanism uses hardware threads to overlap device memory access latency.

For further analysis on Co-OLAP, we compare SSB performance of CPU AIR algorithm, Co-OLAP model(CPU&GPU memory resident AIR algorithm) and open-source column-oriented MonetDB with version Feb2013-SP6 (http://www.monetdb.org/downloads/deb/). We execute each query for 3 times and use the minimal time as execution time to eliminate I/O overhead for MonetDB. In our 12-core workstation, shown as Fig. 5, the average execution time of MonetDB is 571.87 ms. The average execution time of Co-OLAP is 136.36 ms. CPU AIR algorithm outperforms both Co-OLAP and MonetDB, the average execution time is 89.06 ms. Co-OLAP model is 4.2 X faster than MonetDB with GPU accelerator and 6.4 X faster than MonetDB with multicore CPUs.

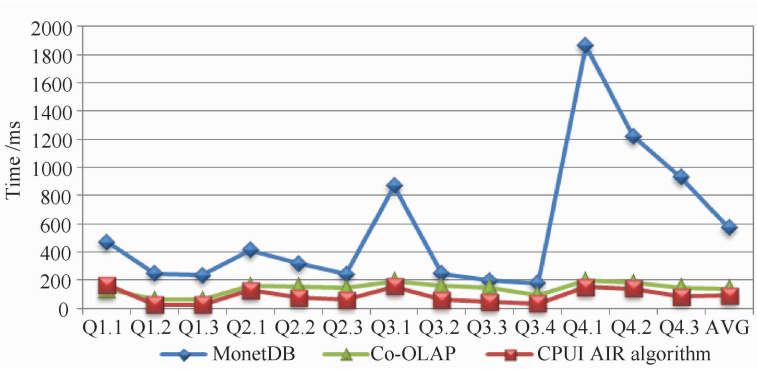

Fig. 5　Performance comparison for different Co-OLAPs

GPU OLAP commonly employs hash join algorithms and the overall performance is usually lower than column based MonetDB[13], the performance gains rely on GPU's hardware performance opposite to algorithm efficiency. AIR algorithm is special designed for GPU vectorized processing and the algorithm efficiency is also higher than conventional hash join algorithms and MonetDB on multicore platform.

CPU AIR algorithm still outperforms Co-OLAP even if we do not consider transfer cost, the pure star-join execution time for CPU and GPU platform is 89.06 *ms* and 99.67 *ms*, the main reasons lie in two aspects:

• Dimension vectors in SSB dataset(SF = 20) amount to 1.62 MB which are far less than CPU's L3 cache size but far larger than GPU's shared memory size, so star-join stage gains better cache locality in CPU than in GPU.

• CPU AIR algorithm employs dynamic *vector* as fact vector to avoid sequential scan on fact vector, programming for GPU is less efficient than for CPU.

As a summary, one GPU's processing performance is approximately equal to processing performance of two CPUs in our experiments. For concurrent queries, half workloads can be assigned to CPUs and the remainder workload can be assigned to GPU and CPUs with Co-OLAP model, the server's performance can be doubled.

4 Conclusion

Different from many researches, we first focus on GPU-aware OLAP framework research instead of GPU-able relational operator optimizations, AIR algorithm is superior to in-memory database on both CPU platform and GPU platform. Co-OLAP model focuses on GPU memory resident computing strategy to maximize GPU computing power and minimize PCIe bus transmission overhead by assigning computing-intensive workload for GPU. Co-OLAP can also model the platform processing power by either configuring GPUs according to dataset size or give the maximal high performance Co-OLAP dataset size according to GPU memory size.

[References]

[1] GOVINDARAJU N K, LLOYD B, WANG W, et al. Fast computation of database operations using graphics processors [C]//SIGMOD Conference. 2004.

[2] HE B, LIU M, YANG K, et al. Relational query coprocessing on graphics processors[J]. ACM Transactions on Database Systems, 2009, 34(4).

[3] HE B, YANG K, FANG R, ey al. Relational joins on graphics processors[C]//SIGMOD, 2008:511 - 524.

[4] PIRK H, MANEGOLD S, KERSTEN M. Accelerating foreign-key joins using asymmetric memory channels [C]//ADMS, 2011.

[5] ALCANTARA D A, SHARF A, ABBASINEJAD F, et al. Real-time parallel hashing on the gpu[J]. ACM Trans Graph, 2009,28(5).

[6] AO N, ZHANG F, WU D, et al. Efficient parallel lists intersection and index compression algorithms using graphics processing units[J]. PVLDB, 2011.

[7] FANG W, HE B, LUO Q. Database compression on graphics processors[C]//VLDB, 2010.

[8] GOVINDARAJU N, GRAY J, KUMAR R, et al. Gputerasort: high performance graphics co-processor sorting for large database management[C]//SIGMOD, 2006.

[9] SATISH N, KIM C, CHHUGANI J, et al. Fast sort on cpus and gpus: a case for bandwidth oblivious simd sort [C]//SIGMOD, 2010.

[10] SITARIDI E, ROSS K. Ameliorating memory contention of olap operators on gpu processors[C]//DaMoN, 2012: 39-47.

[11] WU H, DIAMOS G, CADAMBI S, et al. Kernel weaver: automatically fusing database primitives for efficient gpu computation[C]//MICRO-45. 2012.

[12] HE B, YU J X. High-throughput transaction executions on graphics processors[J]. PVLDB, 2011.

[13] YUAN Y, LEE R B, ZHANG X D. The yin and yang of processing data warehousing queries on GPU devices[J]. PVLDB, 2013, 6(10): 817-828.

[14] BLANAS S, LI Y, PATEL J. Design and evaluation of main memory hash join algorithms for multi-core cpus [C]//SIGMOD, 2011: 37 - 48.

[15] BALKESEN C, TEUBNER J, ALONSO G, et al. Main-memory hash joins on multi-core cpus: Tuning to the underlying hardware[C]//ICDE, 2013.

[16] ABADI D J, MADDEN S, HACHEM N. Column-stores vs. row-stores: how different are they really? [C]// SIGMOD Conference. 2008: 967-980.

[17] ZHANG Y S, WANG S, LU J H. Improving performance by creating a native join-index for OLAP[J]. Frontiers of Computer Science in China, 2011, 5(2): 236-249.

(责任编辑　李　艺)