Hybrid CPU-GPU dynamic graph updating and processing mechanism

Proposals

U201614531   CS1601   Bensong Liu <recolic@hust.edu.cn>

Backgrounds

GPUs have been widely used to accelerate graph calculations. However, in the real world, such as the World Wide Web, social networks, and user trading networks are often dynamically changing. However, the dynamic graph processing system on the existing GPU usually uses GPU-ONLY method. For the updating and processing of dynamic graphs, some computing resources of the computer are still not fully utilized. Therefore, how to effectively utilize the available resources of the hybrid CPU-GPU system to update and process dynamic graphes is an urgent problem. This study mainly considers how to reasonably schedule the computing resources of a hybrid CPU-GPU to increase the speed of dynamic graph updates and obtain high utilization on both the CPU and GPU.

Goals

1. Increase the basic knowledge of Linux operating system, graph algorithm, graph computing system and GPU. Graduation design will deepen the understanding of dynamic graph update and processing mechanism on existing systems.

2. Increase the mastery of C ++ and CUDA programming, learn and understand the current typical graph storage methods and processing models on dynamic graph systems.

3. Test the performance of the existing dynamic graph system on open source systems, and implement a set of strategies that use both CPU and GPU for dynamic graph updates. Improve the system's computing resource utilization and design experiments to test the effectiveness of the proposed method.

4. Strictly abide by the set time plan, complete the thesis and related work, and meet the defense requirements.

Existing researches

Among the high-performance dynamic graph update algorithms of pure GPU, the GPMA+ algorithm proposed in VLDB2017 performs pretty well. After my testing, the GPU core utilization rate can reach about 70%. Since DCSR can neither handle deletions nor support efficient searches, it is in-comparable to GPMA+. The data structure on the GPU side in the system I am completing may refer to the design of GPMA +.

Regarding the partitioning strategy for partitioning dynamic graphs, a paper from VLDB2019 discussed multiple partitioning methods including 1D, 2D and other strategies. It is mainly for the case of multiple GPUs/CPUs, and it proves that CVC is the best for dynamic graph partitioning. For efficient communication between different computing units, the ideas therein also have great reference value for my new system design.

In PPOPP2016 and PPOPP2017, I noticed two libraries for multi-GPU graph processing tasks, namely Gunrock and Groute. Gunrock abstracted graph operations, provided a simpler API, and optimized some GPU computing. On the basis of Gunrock, Groute provides a better high-level abstractions for communication optimization problems of multi-GPU systems, as well as optimizations such as Distributed Worklists, Pipelined Operation, etc., which greatly reduces the performance loss caused by the communication latency between GPUs and between CPUs. These The design scheme can also be implemented on our hybrid CPU / GPU system.

In TACO2016, researcher noticed the similarity of different snapshots in the dynamic graph, and proposed the methods of Fetch Amortization and Processing Amortization to reduce the IO load and reduce unnecessary calculations. Later, GraPU based on this, and aimed at fast-updated dynamic graphs. A method for PreComputing the Updates in the Buffer Node and a method to balance the workload of different computing units by dividing Component and Subgraph are proposed.

Problems to solve

In both GPMA and GPMA + algorithms, pure GPUs are used for calculations. Although GPMA + has been fully optimized for multi-core GPUs, the CPU-to-GPU data transfer latency still has a major impact on performance. By assigning a graph partition to the CPU and RAM for processing, we can reduce task latency, reduce waste of IO wait time during calculation, and thus achieve higher throughput.

At the same time, in our tests, we found that the GPU core utilization of GPMA + can reach about 70%, but the CPU has only one core utilized. If we can use the computing power of the

CPU and GPU at the same time, we will take full advantage of the CPU's proximity to main memory to further improve the performance of dynamic graph update tasks.

Proposed Plan

Firstly, we partition the dynamic graph according to a certain strategy. One part is stored in the GDDR6 video memory of the GPU, and the other is stored in the DDR4 main memory that the CPU can directly access. When adding and deleting edges on the graph, the CPU classifies these Update basing on the partitioning strategy, and the graph partitions in GDDR are handed to the GPU through PCIE for processing, while the core of the CPU processes the partitions responsible for the CPU.

At the same time, when the workload of main memory and video memory is obviously unbalanced, the partition size needs to be adjusted to re-balance. The specific partition strategy needs to be determined through subsequent experiments. Considering that the Proxy Node between GPU and CPU will cause a large number of GPU-CPU message exchanges, the partitioning strategy will also avoid placing nodes with particularly high degrees of ingress and egress on the partition border.

In terms of data structure, GPU video memory and CPU main memory will use GPMAPlus-based data structures and algorithms to handle the update of dynamic graphs within a single computing unit. GPMA+ algorithm is very suitable for the extremely multi-threaded environment on GPUs (often there are thousands of available Threads), due to the small number of threads on the CPU, the parallel loop logic near TryInsert+ may be modified based on the results of the future performance profiling.

Therefore, after the above main work is completed, I will try to continue to optimize the data communication strategy. This will also help our systems with multiple GPUs and multiple CPUs. I will try the GPU Multi-Streaming method mentioned in the Garaph paper or the Distributed Worklist method mentioned in Groute paper, to separate the transmission thread and the computation thread, in order to reduce the CPU-GPU communication overhead.

References

1. Accelerating Dynamic Graph Analytics on GPUs, VLDB 2017

2. Garaph:Efficient GPU-accelerated Graph Processing on a Single Machine with Balanced Replication , ATC 2017

3. CGraph: A Correlations-aware Approach for Efficient Concurrent Iterative Graph Processing, ATC 2018

4. Dynamic Sparse-Matrix Allocation on GPUs. ISC 2016

5. GraPU: Accelerate Streaming Graph Analysis through Preprocessing Buffered Updates. SoCC 2018

6. A Study of Partitioning Policies for Graph Analytics on Large-scale Distributed Platforms, VLDB 2019

7. Gunrock: a high-performance graph processing library on the GPU. PPOPP 2016

8. Groute: An Asynchronous Multi-GPU Programming Model for Irregular Computations. PPOPP 2017

9. Synergistic Analysis of Evolving Graphs. TACO 2016

10. Chronos: a graph engine for temporal graph analysis. EuroSys 2014