CPU-GPU 混合动态图更新和处理机制

开题报告

U201614531　CS1601　刘本嵩

主要内容

GPU 已被广泛用于加速图计算。然而，在现实世界中，诸如万维网、社交网和用户交易网是经常发生动态变化的。然而，现有 GPU 上的的动态图处理系统主要使用纯 GPU 进行处理。对于动态图的更新和处理，都没有充分的利用计算机的所有计算资源。因此，如何有效在 CPU-GPU 混合系统上，同时利用 CPU 和 GPU 的可用资源，进行动态图更新和处理是一个急需解决的问题。此研究主要考虑如何合理地调度混合 CPU-GPU 的计算资源来提高动态图更新的速度，并且获得 CPU 和 GPU 上的高利用率。

目的要求

1. 熟悉 Linux 操作系统、图算法、图计算系统和 GPU 的基础知识,通过毕业设计加深对现有系统上动态图更新和处理机制的理解.

2. 熟悉 C++,CUDA 编程,学习并了解动态图系统上目前典型图存储方式和处理模型.

3. 在开源系统上测试现有典型动态图系统的扩展性等特征,实现一套同时利用 CPU 和 GPU 进行动态图更新的策略。提高系统的计算资源利用率,并设计实验检测所提方法的效果.

4. 严格遵守毕设时间计划,完成论文以及相关工作,符合答辩要求.

现有研究成果

在纯 GPU 的高性能动态图更新算法中,VLDB2017 中提出的 GPMA+算法在纯 GPU 的动态图更新方面性能较好.经测试, GPU 核利用率可以达到 70%左右 由于同时期的 DCSR 不能支持高效的删除和搜索, 因此没有进行比较. 我正在完成的系统中 GPU 侧的数据结构, 可能会参考 GPMA+的设计.

在对动态图进行分区处理的分区策略方面, VLDB2019 的一篇论文讨论了包含 1D, 2D 和其他策略等多种分区方式, 它主要对于多 GPU 的情形, 论证了 CVC 这种分区方式更有利于不同运算单元间的高效交流, 其中的思想对新的系统设计也有很大参考价值.

在 PPOPP2016 和 PPOPP2017 中, 我注意到了两个关于多 GPU 图处理任务的库, 分别是 Gunrock 和 Groute. Gunrock 对图操作进行了抽象, 提供了更简单的 API, 并针对 GPU 计算进行了一些优化. 而 Groute 在 Gunrock 的基础上, 针对多 GPU 系统的通信优化问题提供了对应的高层抽象, 以及 Distributed Worklists, Pipelined Operation 等优化, 大大降低了 GPU 间和 CPU 间的通信延迟带来的性能损失. 这些设计方案同样可以在我们的 CPU/GPU 混合系统上实现.

在 TACO2016 中, 注意到动态图中不同 Snapshots 的相似性, 提出了 Fetch Amortization 和 Processing Amortization 的方法, 来降低 IO 负载和减少不必要的计算. 后来, GraPU 在此基础上, 针对快速更新的动态图提出了在 Buffer Node 对 Updates 进行 PreCompute, 以及通过划分 Component 与 Subgraph 来平衡不同计算单元 workload 的方法.

要改进和解决的问题

在 GPMA 和 GPMA+算法中, 都是使用纯 GPU 来进行计算的. 尽管 GPMA+已经针对多核进行了充分的优化, 但 CPU 到 GPU 的数据传输延迟仍然对性能造成主要影响. 如果能够将部分数据交由 CPU 和主存进行处理, 就可以降低任务延迟, 减少计算过程中 IO 等待时间的浪费, 并因此达到更高的吞吐量.

同时, 在我们的测试中发现, GPMA+的 GPU 核心使用率可以达到 70%左右, 但 CPU 却只有一个核心在工作.如果能够同时发挥 CPU 和 GPU 的计算能力, 就能够更充分的利用 CPU 靠近主存的优势, 进一步提升动态图更新任务的性能.

预计方案

将动态图按照一定的策略进行分区, 一部分保存在 GPU 的 GDDR6 显存中, 另一部分保存在 CPU 能直接访问的 DDR4 主存中. 对动态图进行增加和删除操作时, CPU 先按照既定的分区策略将 Updates 进行分类, 将 GDDR 负责的分区通过 PCIE 交给 GPU 的核心进行处理, 同时 CPU 的核心对 CPU 所负责的分区进行处理.

同时, 当主存和显存的 workload 有明显不平衡时, 需要调整分区大小来进行 re-balance. 具体的分区策略需要通过后续的实验来确定. 考虑到 GPU 和 CPU 之间的 Proxy Node 会造成在 GPU 和 CPU 间大量消息交换, 分区策略也会避免将入度和出度特别多的节点放置在分界线上.

数据结构方面, GPU 显存和 CPU 主存方面都将使用基于 GPMA+的数据结构和算法来处理单个计算单元对动态图的更新. GPMA+算法非常适合 GPU 上极多线程的环境(往往有几千个可用线程), 在 CPU 上由于线程数较少, TryInsert+附近的并行循环逻辑可能会根据届时性能测试的结果进行修改.

因此在以上主要工作完成之后, 我会尝试继续优化数据通信的策略, 这对多 GPU 加多 CPU 的系统也会有帮助. GPU 中会尝试 Garaph 论文中提到的 GPU Multi-Streaming 方法或 Groute 论文中提到的 Distributed Worklist 方法, 将负责与主存进行信息交流的 GPU 线程与负责数据处理的 kernel 线程分开, 以便减少与主存的信息 IO 造成 GPU 计算线程等待的情况.

主要参考文献

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