单卡大图采样

GPU unified memory

> 如果不用`cudaMemAdvise`，用UM支持大图采样会非常慢

对UM的使用根据图数据存在哪可以分成3种情况：

- GPU内存
- GPU内存+CPU内存
- CPU内存

三种情况预期的采样时间应该会逐渐变长。

对于第二个情况，实现时将前半段放入GPU内存，剩余部分放入CPU内存：

```
CUDA_CALL(cudaMemAdvise(ret, device_nbyte,
    cudaMemAdviseSetPreferredLocation, ctx.device_id));
CUDA_CALL(cudaMemAdvise(ret, device_nbyte,
    cudaMemAdviseSetAccessedBy, ctx.device_id));

CUDA_CALL(cudaMemAdvise(ret + device_nbyte, host_nbyte,
    cudaMemAdviseSetPreferredLocation, cudaCpuDeviceId));
CUDA_CALL(cudaMemAdvise(ret + device_nbyte, host_nbyte,
    cudaMemAdviseSetAccessedBy, ctx.device_id));
```

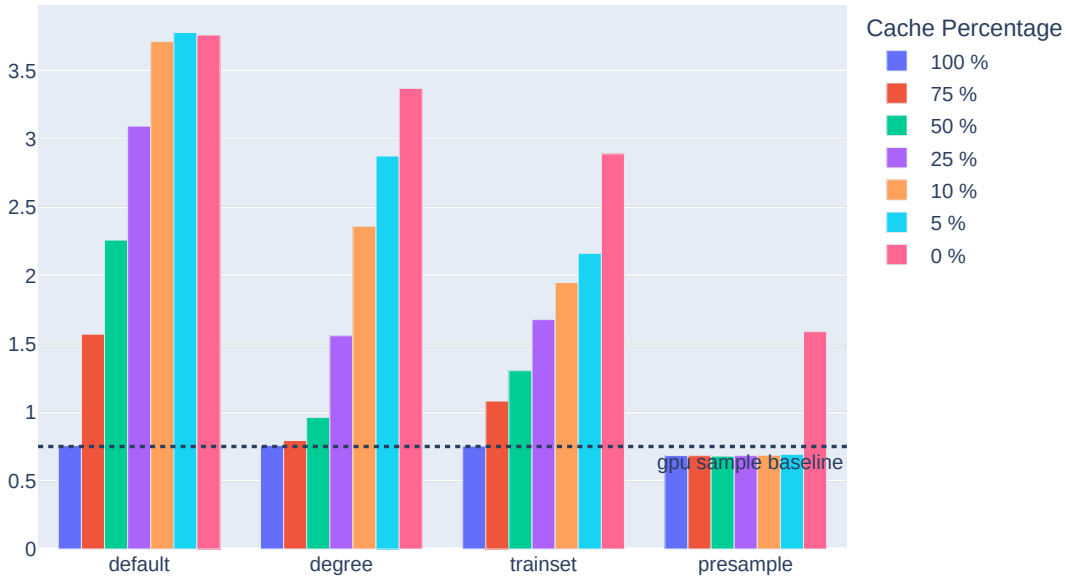
因此不同的图数据顺序可能会影响采样时间，这里测试了4种策略：

```
enum class UMPolicy {
  kDegree = 0, // 度数优先
  kTrainset,   // 训练集中的节点优先
  kRandom,
  kPreSample,  // 和缓存特征情况类似，根据presample结果得到ranking
  kDefault,    // 默认的图数据
};
```

根据不同策略得到的节点rank，重新排序csr图数据

测试

papersM100

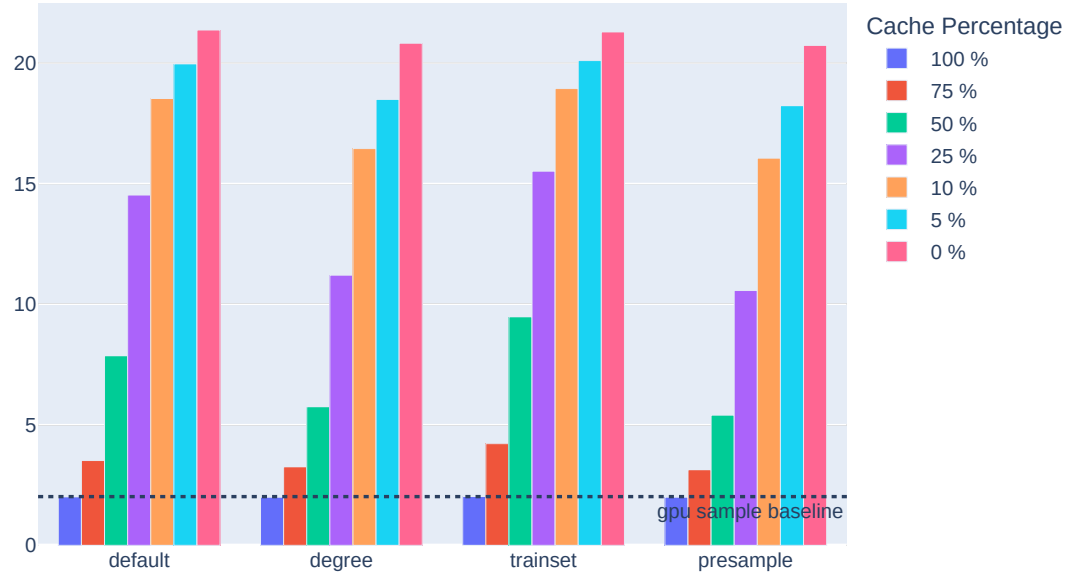


使用GPU采样的速度要比CPU快很多，CPU采样时间是14.42 sec。

正常GPU采样、用UM但图存在GPU内存，两个的采样时间应该是基本相等的。

将图数据存在CPU内存的采样时间，也会受到csr图数据的顺序的影响。而且，将图全存在GPU的采样时间，可能也会受到csr顺序的影响，但这个影响似乎不是很大。

friendster



虽然也能看到和papersM100相似的趋势，但是不同策略之间差的不是很多

partition

目前的方法是，将csr分成多个csr数据，在采样时依次加载到GPU采样。

一种实现：

```
for(IdType i = 0; i < partition.Size(); i++) {
    Timer t0;
    Load(i, indptr_buf[0], indices_buf[0], cu_stream);
    sample_device->StreamSync(ctx, cu_stream);
    partition_load_time += t0.Passed();
    IdType offset = h_partition_offset[i];
    size_t num_input = h_partition_input_size[i];
    Timer t1;
    size_t num_tiles = (num_input + Constant::kCudaTileSize + 1) /
Constant::kCudaTileSize;
    const dim3 grid(num_tiles);
    const dim3 block(Constant::kCudaBlockSize);
    partition_sample_khop0<Constant::kCudaTileSize><<<grid, block, 0,
cu_stream>>>(
      indptr_buf[0], indices_buf[0],
      partition_input + offset, num_input,
      partition.GetNodeIdRMap(i), partition_node_pos_rmap + offset,
      fanout, tmp_src, tmp_dst, random_states->GetStates(), random_states-
>NumStates(),
      RunConfig::partition_check);
    sample_device->StreamSync(ctx, cu_stream);
    partition_sample_time += t1.Passed();
}
sample_device->StreamSync(ctx, cu_stream);
```

但是这样有两个问题：

- 每次将图加载到GPU用时很长
- csr数据相比之前的csr的节点数更少，并行分配的线程比原来少

第二个可能可以每个点分配一个warp，增加使用的线程。

```
const int WARP_SIZE = 32;
const int BLOCK_WARP = Constant::kCudaBlockSize / WARP_SIZE;
const int TILE_SIZE = BLOCK_WARP;
const int num_tiles = (num_input + TILE_SIZE - 1) / TILE_SIZE;
const dim3 grid(num_tiles);
const dim3 block(WARP_SIZE, BLOCK_WARP);
partition_sample_khop0<WARP_SIZE, BLOCK_WARP, TILE_SIZE><<<grid, block, 0,
cu_stream>>>(
    indptr_buf[0], indices_buf[0],
    partition_input + offset, num_input,
    partition.GetNodeIdRMap(i), partition_node_pos_rmap + offset,
    fanout, tmp_src, tmp_dst, random_states->GetStates(), random_states-
>NumStates(),
    false);
```

第一个问题，似乎可以和采样overlap，但是在增加分配的线程数量后，采样kernel的时间减少很多，将每个csr加载到GPU占了主要的时间。

测试

reddit

	epoch sample time	task sample time
GPU采样	0.0968	0.0048
partition	2.5	0.1264

```
partition task sample time: 0.1264
|
+--GPUPartitionSampleKHop0主要采样时间: 0.1205
|  |
   +--加载csr数据时间: 0.1120
   |
   +--cuda kernel时间: 0.0084
```

papers100M

	epoch sample time	task sample time
GPU采样	0.76	0.0051
partition	252.6351	1.6731

```
partition task sample time: 1.6731
|
+--GPUPartitionSampleKHop0主要采样时间: 1.6658
|  |
   +--加载csr数据时间: 1.6630
   |
   +--cuda kernel时间: 0.0027
```

另一个方法是采用PaGraph的partition，减少数据H2D的次数

Algorithm 1: Computation-balanced and cross-partition access free graph partition

Input: graph $\mathcal{G}$, train vertex set TV, number of hops L, partition number K

Output: graph partition $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_k$

1 **for** $i \leftarrow 1$ *to* K **do**

2 $\quad \mathcal{G}_i \leftarrow \varnothing$ `// Initialization`

3 **for** *each train vertex* $v_t \in TV$ **do**

4 $\quad IN(v_t) \leftarrow$ IN-NEIGHBOR$(v_t, \mathcal{G}, L)$

5 $\quad score_{v_t} \leftarrow$ SCORE$(v_t, IN(v_t))$

6 $\quad ind \leftarrow \arg\max_{i \in [1,K]} \{score_{v_t}^{(i)}\}$

7 $\quad \mathcal{G}_{ind} \leftarrow \mathcal{G}_{ind} \cup \{v_t\} \cup \{IN(v_t)\}$

8 return $\{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_K\}$

但这个一个非常耗时的算法，在com-friendster可能会跑很久。