稀疏矩阵-矩阵乘 实验报告

管思源 2021012702

性能优化方式

> 为了逻辑组织的方便，以下我将依次叙述每次迭代所作出的优化，以及相应的加速效果（以 `len = 32` 为例）

在本次实验中，我总共尝试了两种优化思路。显然，第一种没能取得很好的效果，不过它依然为我理解SpMM问题的优化瓶颈提供了经验，因此我先简单地介绍一下第一种优化思路及其效果。

在拿到这个计算问题后，我判断主要的优化瓶颈是Warp divergence和负载不均衡，即一个thread需要处理稀疏矩阵的一整行非零元，而行与行之间非零元个数相差很大，导致一个thread拖慢整个warp、一个warp拖慢整个block、一个block拖慢最终计算时间的现象。

针对这个问题，一个很直觉的解决方法是使所有thread的工作量完全相同，即将**每个非零元素映射到一个thread**，这也是我一开始尝试的办法，优化效果如下：

	Arxiv	Collab	Citation	DDI	Protein	PPA	Reddit.dgl	Products	Youtube	Amazon_cogdl	Yelp	Wikig2	Am
用时(ms)	9.0	12.8	249.1	5.0	232.7	281.6	570.3	911.2	34.4	1954.2	88.6	22.5	59.2
较前次加速比	4.86	1.58	1.29	2.44	3.37	1.57	1.93	1.47	3.84	1.26	1.56	1.51	9.89

可以看到计算效率的确得到了提升。

接下来，由于我认为已经解决了Warp divergence的问题，又考虑到SpMM计算中会大量访问在Global Memory中的特征矩阵、访存带宽可能成为主要瓶颈，于是希望针对GPU访存行为进行优化。

先前每个非零元素映射到一个thread的策略使同一个warp中的thread访问的是特征矩阵的不同行。因为GPU访问Global Memory是以Sector (32 Bytes)为单位，所以这样做相当于同时访问这些行的前8个元素，受到了访问带宽的限制。

为了提高访存效率，又考虑到一个warp内的thread有合并访问的特性，于是我决定将**每个非零元素映射到32个thread**，每个thread处理特征矩阵对应行的1个（8个，对len=256）元素，优化效果如下：

	Arxiv	Collab	Citation	DDI	Protein	PPA	Reddit.dgl	Products	Youtube	Amazon_cogdl	Yelp	Wikig2	Am
用时(ms)	4.9	4.0	41.0	19.0	875.5	121.5	1273.1	369.5	18.3	3503.4	44.9	14.9	36.9
较前次加速比	1.83	3.23	6.07	0.26	0.27	2.32	0.45	2.47	1.88	0.56	1.97	1.51	1.60

测试发现，这种策略对大部分数据集都有速度提升，但对于DDI、Protein、Reddit.dgl、Amazon_cogdl四个数据集反而是逆优化。我对比了这些数据集的规模和非零元个数发现，这四个数据集对应的稀疏矩阵稠密度更高，即平均每行非零元个数更多。这意味着它们的计算负载更重、所需要的block更多。当我们将每个非零元素映射到32个thread时，每个block承担的计算负载同时变轻了，所以在需要大量block的这些数据集上，block的调度成本和占有率就成为了瓶颈。

	Arxiv	Collab	Citation	DDI	Protein	PPA	Reddit.dgl	Products	Youtube	Amazon_cogdl	Yelp	Wikig2	Am
M	169343	235868	2927963	4267	132534	576289	232965	2449029	1138499	1569960	716847	2500604	881680
NNZ	1166243	2358104	30387995	2135822	79122504	42463862	114615891	123718280	5980886	264339468	13954819	16109182	5668682
Avg(NNZ)	6.9	10.0	10.4	500.5	597.0	73.7	492.0	50.5	5.3	168.4	19.5	6.4	6.4

在无法解决这个问题的情况下，我决定着手解决局部性问题。我认为局部性问题主要还是出在对特征矩阵的访问上，即对特征矩阵哪一行的访问完全是随机的，访问一次就要丢弃、很少复用，无法利用内存的时间局部性。

我的解决方案是对稀疏矩阵非零元的顺序进行预处理，将**以行为主序转变为以列为主序**（从CSR转为COO天然是以行为主序的）。这样一来，连续的同一列的非零元将会访问特征矩阵中的同一行，从而提高内存的时间局部性命中，优化效果如下：

	Arxiv	Collab	Citation	DDI	Protein	PPA	Reddit.dgl	Products	Youtube	Amazon_cogdl	Yelp	Wikig2	Am
用时(ms)	1.3	2.3	27.4	2.0	69.4	37.5	99.9	110.4	9.2	Timeout	12.3	15.0	15.7
较前次加速比	3.66	1.76	1.50	9.62	12.62	3.24	12.75	3.35	1.99	/	3.66	1.00	2.36

可以看到计算效率的确得到了较大提升。

此后，我做了诸多尝试（如使用共享内存、进一步处理非零元顺序等），但均以失败告终。而此时我的用时仍在Cusparse的2~3倍左右，无法通过常规优化途径达到预期效果，只能抛弃既定的优化思路另寻它法。

在分析这一次尝试的优化思路的失败时，我认为犯了两点错误：

一是对SpMM计算任务中的”串行“部分和并行部分认识不足。访存和矩阵元乘法是可并行的，但加法和写回操作是难以并行的。当我将每个非零元映射到一个或多个thread时，就不得不面对同一行非零元之间的加法和写竞争/写冲突的问题，从而必须执行原子加法（AtomicAdd）。我后来意识到，这个原子加法产生的等待很可能是制约我进一步优化的瓶颈，但我无能为力。

二是我只考虑了特征矩阵的访存优化，而忽略了稀疏矩阵本身也是需要从Global Memory中读取的、也是需要访存优化的。

在寻找其他优化思路的过程中，我搜索到了一片参考文献（G. Huang, G. Dai, Y. Wang and H. Yang, "GE-SpMM: General-Purpose Sparse Matrix-Matrix Multiplication on GPUs for Graph Neural Networks"），以下优化也是基于文献中叙述的思路展开的。

简而言之，在回归以一个稀疏矩阵行为单位的基础上，我做了两大优化：

1. **并行地访问稀疏矩阵系数（ptr，idx，val），并缓存在Shared Memory中共用；**
2. **每一行对应一个或多个warp，warp中的每个thread负责特征矩阵的不同列，消除warp divergence。**

优化效果如下：

	Arxiv	Collab	Citation	DDI	Protein	PPA	Reddit.dgl	Products	Youtube	Amazon_cogdl	Yelp	Wikig2	Am
用时(ms)	1.5	0.7	9.5	0.3	8.8	10.4	22.2	32.4	3.8	60.5	3.8	3.4	17.1
较前次加速比	0.88	3.05	2.88	6.16	7.88	3.60	4.50	3.40	2.44	/	3.20	4.47	0.91

除了Arxiv和Am数据集（数据结构不规则、更有利于第一种优化思路中的完全平均负载），速度均有较大提升。

接下来，我在这种优化思路上进行了一些小改进。先是为了消除负载不均衡，**将一个稀疏矩阵行映射到一个block**，而非一个block负责多行，从而降低block的调度成本和内部负载不均情况，优化效果如下：

	Arxiv	Collab	Citation	DDI	Protein	PPA	Reddit.dgl	Products	Youtube	Amazon_cogdl	Yelp	Wikig2	Am
用时(ms)	1.52	0.66	9.15	0.29	8.20	10.29	21.84	31.98	3.88	54.79	3.63	4.51	17.07
较前次加速比	1.002	1.123	1.039	1.100	1.073	1.015	1.016	1.014	0.971	1.105	1.055	0.743	1.004

可以看到有略微提升，但效果不明显。

此外，我还将每个load 32个非零元改为64个非零元，意在减少thread同步造成的等待，优化效果如下：

	Arxiv	Collab	Citation	DDI	Protein	PPA	Reddit.dgl	Products	Youtube	Amazon_cogdl	Yelp	Wikig2	Am
用时(ms)	1.34	0.65	9.17	0.27	8.09	10.17	21.33	31.72	3.63	53.41	3.59	4.50	15.73
较前次加速比	1.136	1.011	0.997	1.079	1.014	1.012	1.024	1.008	1.068	1.026	1.012	1.003	1.086

也是略有提升。

进一步，我又尝试了一些常规优化，包括提取公共计算、提取常量等，效果如下：

	Arxiv	Collab	Citation	DDI	Protein	PPA	Reddit.dgl	Products	Youtube	Amazon_cogdl	Yelp	Wikig2	Am
用时（ms）	1.17	0.64	9.11	0.28	8.07	10.11	21.13	31.69	3.16	52.65	3.53	4.46	13.34
较前次加速比	1.149	1.028	1.007	0.977	1.002	1.005	1.010	1.001	1.150	1.015	1.019	1.008	1.179

也是略有提升。

至此，我基本已经穷尽了常规优化的点子，但距离性能线仍有一段距离，特别是Arxiv、Am数据库。在往年实现的启发下，我决定尝试预处理稀疏矩阵。

我首先尝试了Metis，但是它要求待处理的是无向图、且在部分数据集上会出现超时的情况，因此放弃。

于是我决定**自己实现简单的预处理功能**，其基本思想如下：

1. 先置换系数矩阵的列，使得每一行的非零元在列数上尽可能靠近；
2. 再置换系数矩阵的行，使得每一列的非零元在行数上尽可能靠近；
3. 最后逆变换第一步中的列置换。

我一开始只采用了第2步，因为我认为只需要提高访问特征矩阵的局部性即可，写回结果矩阵的局部性不影响速度。这样优化的效果如下：

	Arxiv	Collab	Citation	DDI	Protein	PPA	Reddit.dgl	Products	Youtube	Amazon_cogdl	Yelp	Wikig2	Am
用时（ms）	1.23	0.60	8.58	0.25	9.55	7.00	14.12	20.98	3.18	39.16	3.24	4.73	14.07
较前次加速比	0.948	1.057	1.061	1.114	0.845	1.445	1.497	1.510	0.992	1.345	1.087	0.944	0.948

可以看到一部分数据集在预处理后得到了提升，而另一部分则反而变慢了。这一方面可能是因为我实现的算法实际没有做到使得每一列的非零元在行数上尽可能靠近，而是简单地以每行首个非零元的列数进行排序，反而是非零元之间离得更远了；另一方面可能是因为预处理后需要在计算时额外访问逆变换向量，从而增大了访存时间。

后来，由于性能不达预期，我还是尝试了第1、3步（如果没有第3步，那么需要在循环里访问列置换的逆变换向量，开销太大，所以在预处理时直接做逆变换），效果如下：

	Arxiv	Collab	Citation	DDI	Protein	PPA	Reddit.dgl	Products	Youtube	Amazon_cogdl	Yelp	Wikig2	Am
用时（ms）	1.14	0.55	7.55	0.26	9.14	6.33	14.83	19.18	3.23	39.81	3.02	4.48	13.23
较前次加速比	1.084	1.097	1.138	0.972	1.045	1.106	0.952	1.094	0.985	0.983	1.075	1.055	1.064

同样是一部分数据集在预处理后得到了提升，而另一部分则反而变慢了。

最后，我通过测试结果决定每个数据集是否使用预处理，即为最终优化结果（如下）。

最终测试结果

```
len = 32
```

数据集	用时（us）	吞吐量（1e9）	较性能线加速比	较 Cusparse 加速比	较朴素实现加速比
Arxiv	1171.4	0.996	0.299	0.659	37.354
Collab	549.1	4.294	1.129	2.427	37.022
Citation	7544.6	4.028	1.180	2.180	42.502
DDI	247.7	8.624	1.009	2.583	49.182
Protein	8078.6	9.794	1.015	3.052	97.114
PPA	6341.5	6.696	1.388	2.895	69.875
Reddit.dgl	14094.9	8.132	1.206	3.444	77.927
Products	19107.9	6.475	1.675	2.922	69.889
Youtube	3137.9	1.906	0.892	1.163	42.098
Amazon_cogdl	39033.6	6.772	1.127	3.210	63.189
Yelp	3020.8	4.620	1.126	2.176	45.903
Wikig2	4475.3	3.600	0.983	1.595	7.605
Am	13228.7	0.429	0.174	0.283	44.254

`len = 256`

数据集	用时（us）	吞吐量（1e8）	较性能线加速比	较 Cusparse 加速比	较朴素实现加速比
Arxiv	3522.2	3.311	0.710	0.850	108.880
Collab	3127.5	7.540	1.439	1.663	61.597
Citation	50621.1	6.003	1.383	1.562	54.473
DDI	1489.8	14.337	1.007	1.043	65.360
Protein	95073.6	8.322	1.367	0.850	/
PPA	60079.4	7.068	1.332	1.413	62.046
Reddit.dgl	152202.0	7.531	1.051	1.329	/
Products	168139.0	7.358	1.487	1.537	/
Youtube	13166.5	4.543	1.215	1.096	88.740
Amazon_cogdl	412657.0	6.406	0.969	1.254	/
Yelp	25009.0	5.580	1.080	1.200	50.474
Wikig2	13819.3	11.657	1.809	1.213	32.081
Am	26471.3	2.141	0.491	0.507	/