Report

实验方法

该实验需要我们完成对稀疏矩阵乘法的加速。

在原实现 `spmm_ref.cu` 中，每个线程负责 A 矩阵的一整行的计算。由于每一行的非零元素的个数 可能大不相同，导致一个Warp内每个线程的工作量大不相同。由于一个Warp的运行时间由最慢的一个线程决定，这就产生了**Warp divergence**，使得程序的效率十分低下。

优化1 为了解决Warp divergence，我们需要考虑将一个Warp内的所有线程的工作量尽可能保持一致。我们可以想到，若一个Warp内是固定 A 中的一行，而每一个线程的区别是处理 B 矩阵中 32 列中的的不同列。由于 B 中每一列需要参与计算的元素是相同的，因此这就可以保证一个Warp内每个线程的工作量相近。我们设定一个Grid内的形状为 $(num\_v, \lceil feat\_in/32 \rceil)$ 一个线程块内的形状为 (32) 也即都在一个Warp中。那么每一个线程块内，就可以将对应 A 中的非零元的元素按照 32 个为一组储存进共享内存中，然后计算对应的 B 中每个线程对应的列中的元素与共享内存中的元素相乘，这样就大致解决了Warp divergence的问题。

在此基础之上，还可以做两个进一步的访存优化。

优化2 一是考虑一个Warp中，每个线程读取对应的 A 中的元素时，可以一次读取多个储存进共享内存，这样可以减少 `__syncthreads()` 等操作带来的性能影响，在代码中使用 `D` 全局变量来控制一批读取的数据个数为 $32 \times D$ 个，但该优化也可能带来一些访存不连续的问题，需要调整参数。

优化3 二是考虑一个Warp中，可以不仅仅计算 B 矩阵中的 32 列，在代码中使用 `C` 变量来控制，一个Warp计算 B 中的 $32 \times C$ 个列，这样线程块的数量变为原来的 $\frac{1}{C}$ 个，一个线程块内由于 A 中的元素已经储存在共享变量中，因此可以优化访存。但同样会导致访存不连续的问题，需要调参。

优化4 除了以上优化之外，由于 A 矩阵中每一行的元素个数不同，所以还存在着负载不均衡的问题。每一个线程块的运行时长大不相同，因此可以考虑如下优化：按照一个固定的大小（在代码中由全局变量 `SPLIT_SIZE` 控制）将矩阵 A 中每一行的非零元素进行一个分组，每组元素个数为 `SPLIT_SIZE` 。每一组记录对应的 `ptr` 位置以及行号于 `ptr_scheduled` 和 `target` 中，总计有 `num_target` 组，然后Grid的形状变为 $(num\_target, \lceil feat\_in/32/C \rceil)$ 。每一个线程块计算的不再是 A 中的一行而是之前分好的其中一组，这样就使得每个线程块的运行时间相近，使负载均衡。

优化效果

优化1：Warp divergence

以下为 `kLen=32` 时在 `arxiv, collab` 数据集上 `spmm_ref.cu` 的实现与使用优化1优化Warp divergence后的实现的运行时间对比。

Dataset	spmm_ref	spmm_opt	加速比
arxiv	0.0439093	0.00176163	24.925381606807332
collab	0.0203202	0.000663304	30.634822042381774

可以发现该优化使得效率获得了极大的提升

以下为 `kLen=32` 时所有数据集上 `spmm_cusparse.cu` 与 `spmm_opt.cu` 的运行时间比较以及加速比

Dataset	Cusparse Performance	Opt Performance	加速比
arxiv	0.000731204	0.00174855	0.41817734694461123
collab	0.00127019	0.000663248	1.9151056618338844
citation	0.0164478	0.00934392	1.7602676392777332
ddi	0.000640428	0.000338781	1.890389366581951
protein	0.0246531	0.00854603	2.884742974223119
ppa	0.0183809	0.0103841	1.7701004420219373
reddit.dgl	0.0485226	0.0221983	2.185870089150971
products	0.0558522	0.032131	1.7382652267280818
youtube	0.00364364	0.00459277	0.7933425797503467
amazon_cogdl	0.12536	0.0560563	2.2363231251438282
yelp	0.00657345	0.00379826	1.7306477176391295
wikikg2	0.00714571	0.00447128	1.5981352096044086
am	0.00374118	0.0211755	0.1766749309343345

可以发现已经可以在大部分数据集上都优于cusparse的实现

优化4：负载均衡

Dataset	Cusparse Performance	Opt1 Performance	S_Z=32	S_Z=64	S_Z=128	S_Z=256
arxiv	0.000770613	0.00174855	0.000401069	0.000362419	0.000361326	0.000374406
collab	0.00133035	0.000663248	0.000710231	0.000687731	0.000694615	0.000707043
citation	0.0164427	0.00934392	0.0100033	0.00999729	0.0099901	0.0100042
ddi	0.000640603	0.000338781	0.000289118	0.000237132	0.000234011	0.000240908
protein	0.0246571	0.00854603	0.0116737	0.00900886	0.00850541	0.0083887
ppa	0.0183678	0.0103841	0.0102205	0.0102782	0.0102203	0.0102111
reddit.dgl	0.0485409	0.0221983	0.0243041	0.0214908	0.0210961	0.021007
products	0.0558369	0.032131	0.0325955	0.0326952	0.0325724	0.0324766
youtube	0.00364393	0.00459277	0.00263449	0.00256862	0.00255539	0.00256195
amazon_cogdl	0.125351	0.0560563	0.0672361	0.054572	0.0516945	0.0514304
yelp	0.0065746	0.00379826	0.00363131	0.00356125	0.00356207	0.00357361
wikikg2	0.00713908	0.00447128	0.00483369	0.00482832	0.00482924	0.00483059
am	0.00374187	0.0211755	0.00274526	0.00240042	0.00230276	0.002272

可以发现在优化1后表现不是特别好的 `arxiv, youtube, am` 数据集上，使用优化4后效率得到了特别大的提升。推测是由于这三个数据集上 A 矩阵的每一行非零元素个数差距过大，导致负载不均衡。但略有遗憾的是在其他部分数据集上使用优化4却会不同程度地导致效率降低。因此，在 `arxiv, ddi, youtube, yelp, am` 五个数据集上我们使用优化4，并取参数 `SPLIT_SIZE=128` ，其他数据集上还是保持使用原来代码。则可得到如下结果：

Dataset	Cusparse Performance	Opt Performance	加速比
arxiv	0.000731516	0.000362669	2.0170348168715826
collab	0.00126994	0.000663914	1.912807984166624
citation	0.0164432	0.00933037	1.7623309686539765
ddi	0.000640139	0.000231061	2.7704329159832253
protein	0.0246538	0.00844323	2.919948882122126
ppa	0.0183589	0.0103727	1.7699248990137573
reddit.dgl	0.048546	0.022186	2.188136662760299
products	0.0558192	0.0321407	1.7367138861319136
youtube	0.00364352	0.00256387	1.4211016939236387
amazon_cogdl	0.125306	0.0560178	2.2368961294445695
yelp	0.00657554	0.00356428	1.8448438394290012
wikikg2	0.00714592	0.00448198	1.594366775398373
am	0.00373922	0.00230523	1.6220594040507887

可以发现在所有数据集上Opt的表现均优于Curparse

优化2：访存优化

在前面两个优化的基础上，根据实验方法中的优化2，我们取 $D = 1, 2, 4$ 可以得到如下结果：

Dataset	Cusparse Performance	D=1	D=2	D=4
arxiv	0.000731516	0.000362669	0.000361358	0.00049393
collab	0.00126994	0.000663914	0.000664861	0.00096082
citation	0.0164432	0.00933037	0.00935906	0.011866
ddi	0.000640139	0.000231061	0.000225548	0.000312457
protein	0.0246538	0.00844323	0.00825533	0.0141935
ppa	0.0183589	0.0103727	0.0102375	0.0129275
reddit.dgl	0.048546	0.022186	0.0216514	0.030666
products	0.0558192	0.0321407	0.0317988	0.0384717
youtube	0.00364352	0.00256387	0.00258859	0.00320652
amazon_cogdl	0.125306	0.0560178	0.054552	0.0766084
yelp	0.00657554	0.00356428	0.00354996	0.00459788
wikikg2	0.00714592	0.00448198	0.00452719	0.00583546
am	0.00373922	0.00230523	0.00232834	0.00278087

根据以上结果，我们做种选择取 $D = 2$

优化3：访存优化

在前三个优化的基础上，实现实验方法中的优化3，当 `kLen=32` 时该优化只有副作用，因此只有当 `kLen=256` 时我们使用该优化。

令 `kLen=256` ，我们取 $C = 1, 2, 4$ 可以得到如下结果：

Dataset	Cusparse Performance	C=1	C=2	C=4
arxiv	0.00299168	0.00287986	0.00263345	0.00276056
collab	0.00520264	0.00512518	0.004385	0.004561
citation	0.0790449	0.0809104	0.0691593	0.0700049
ddi	0.00155247	0.00168517	0.00149135	0.00143586
protein	0.0808578	0.0652005	0.0672139	0.0794952
ppa	0.0848833	0.0810183	0.0802434	0.0822321
reddit.dgl	0.202359	0.168885	0.181605	0.196823
products	0.25843	0.258061	0.245782	0.246705
youtube	0.0144381	0.0208401	0.0153589	0.0145374
amazon_cogdl	0.517907	0.406526	0.431299	0.468922
yelp	0.0300105	0.0290906	0.0277571	0.0290391
wikikg2	0.0167518	0.0356323	0.0205155	0.0162583
am	0.0134097	0.0185721	0.0134922	0.011978

因此，我们选择在 `youtube, wikikg2, am` 三个数据集上选择 $C = 4$ ，其他数据集上选择 $C = 2$

运行时间及加速比

综合以上四种优化，最终测试了一遍，我们得到如下的运行结果：

`kLen=32`

Dataset	Cusparse Performance	Opt Performance	加速比
arxiv	0.000753566	0.000368653	2.044106517511047
collab	0.00130332	0.000677276	1.9243558017706224
citation	0.0164425	0.0093671	1.7553458380928995
ddi	0.000640785	0.000228127	2.808895922008355
protein	0.0246446	0.00834096	2.9546479062362128
ppa	0.0183573	0.0102423	1.792302510178378
reddit.dgl	0.0485552	0.0216745	2.24019931255623
products	0.0558177	0.0318033	1.7550914527737687
youtube	0.00364432	0.00259258	1.4056731132694071
amazon_cogdl	0.125344	0.0545931	2.2959678054552684
yelp	0.00657464	0.00355674	1.848501717865236
wikikg2	0.00713817	0.00452314	1.5781448285925264
am	0.0037385	0.00233091	1.603880029687976

`kLen=256`

Dataset	Cusparse Performance	Opt Performance	加速比
arxiv	0.00299189	0.00263276	1.1364081800088122
collab	0.00520291	0.00438386	1.1868330649245187
citation	0.0789124	0.0691259	1.141575010234948
ddi	0.00155131	0.00149355	1.0386729603963711
protein	0.0808712	0.0671635	1.2040944858442457
ppa	0.084873	0.0802323	1.0578407947921222
reddit.dgl	0.202345	0.181591	1.1142898051114867
products	0.258316	0.245763	1.051077664253773
youtube	0.014433	0.0145371	0.9928390119074643
amazon_cogdl	0.517765	0.431213	1.2007175108357122
yelp	0.0299913	0.0277496	1.0807831464237323
wikikg2	0.0166421	0.0162442	1.0244948966400316
am	0.013387	0.0119741	1.1179963421050434