PA4 稀疏矩阵-矩阵乘（SpMM）实验报告

1 实现方法

由于 CSR 的储存格式特点，容易想到的基本做法是，每个线程块将稀疏矩阵的某一行与稠密矩阵相乘，得到结果矩阵的某一行。在此基础上，为了提高效率、均衡负载，使用了以下优化：

1. 在共享内存中储存稀疏行，每个线程块分若干轮计算，一轮读入 32 个稀疏行元素，将它们与稠密矩阵中的 32×32 的块进行乘法
2. `kLen=32` 时，只要每个线程块 32 个线程即可

   `klen=256` 时，只要每个线程块 32 个线程，每个线程计算两个数，即计算规模为 1×32 与 32×64 的矩阵乘法。相比于直接进行规模为 1×32 和 32×256 的矩阵乘法，这种分配虽然需要四个线程块来完成，增加了从全局内存到共享内存的内存负担，但是减小了计算的粒度，总体上提高了效率
3. 调整行的计算顺序，按照非零元数量从大到小排序，并将非零元多的行拆成多个任务计算

2 具体优化效果分析

在使用优化2之前， `kLen=256` 时， `arxiv` 上仅有 0.75 倍加速比，在 `ddi` 上仅有 0.81 倍加速比；优化后， `arxiv` 上达到了 1.13 倍加速比， `ddi` 上达到了 1.02 倍加速比。

在使用优化3之前， `kLen=32` 时， `arxiv` 上仅有 0.68 倍加速比，在 `am` 上仅有 0.30 倍加速比；优化后， `arxiv` 上达到了 2.07 倍加速比， `ddi` 上达到了 1.65 倍加速比。这两个数据集上，非零元分布非常不均匀，极少几行有超过 10000 个非零元，大部分行非零元都不超过 10 个，优化3在均衡负载上起到了显著的效果。

此外，注意到 `kLen=256` 时，加速比显著降低了，这说明在优化2之上仍然有更好的负载均衡方法。

	num_vertex	num_edge	nnz/row
arxiv	169343	1166243	6.89
collab	235868	2358104	10.00
citation	2927963	30387995	10.38
ddi	4267	2135822	500.54
protein	132534	79122504	597.00
ppa	576289	42463862	73.69
reddit.dgl	232965	114615891	491.99
products	2449029	123718280	50.52
youtube	1138499	5980886	5.25
amazon_cogdl	1569960	264339468	168.37
yelp	716847	13954819	19.47
wikikg2	2500604	16109182	6.44
am	881680	5668682	6.43

3 测试结果

3.1 kLen = 32

dataset	cuSparse	spmm_opt	speedup
arxiv	0.77	0.37	2.07
collab	1.33	0.62	2.15
citation	16.45	8.95	1.84
ddi	0.64	0.25	2.55
protein	24.61	8.14	3.02
ppa	18.36	9.35	1.96
reddit.dgl	48.67	18.51	2.63
products	55.81	31.90	1.75
youtube	3.64	2.82	1.29
amazon_cogdl	125.34	47.69	2.63
yelp	6.57	3.50	1.88
wikikg2	7.15	4.39	1.63
am	3.74	2.27	1.65

3.2 kLen = 256

dataset	cuSparse	spmm_opt	speedup
arxiv	3.00	2.65	1.13
collab	5.20	4.50	1.15
citation	78.99	75.79	1.04
ddi	1.55	1.52	1.02
protein	80.86	105.28	0.77
ppa	84.99	74.44	1.14
reddit.dgl	202.32	160.69	1.26
products	258.49	248.71	1.04
youtube	14.43	16.46	0.88
amazon_cogdl	517.27	401.14	1.29
yelp	29.99	26.55	1.13
wikikg2	16.76	24.56	0.68
am	13.42	12.93	1.04