lab4

PB21051012 刘祥辉

算法设计

1.串行代码

spmm算法

```
for (int i = 0; i < N; i++) // 遍历稠密矩阵 A 的行
    for (int j = 0; j < K; j++) // 遍历稀疏矩阵 B 的列
    {
        float acc = 0; // 累积变量，用于存储当前行和列的点积
        for (int l = rowPtr[j]; l < rowPtr[j + 1]; l++) // 遍历稀疏矩阵 B 的非零元素
        {
            int idx = index[l]; // 当前非零元素的行索引
            float val = value[l]; // 当前非零元素的值
            acc += A[i][idx] * val; // 计算点积的一部分
        }
        C[i][j] = acc; // 将点积结果存储到结果矩阵 C 中
    }
```

1.1 问题分析

并行化部分：

- 外层循环（i 循环）可以被并行化，因为每次循环之间不存在依赖关系，可以同时处理不同的行。
- 内层循环（j 循环）在一定程度上也可以被并行化，因为每个 j 对应的计算之间也没有依赖关系，但需要注意到行指针（rowPtr）的使用，可能需要一定的同步机制。

不可并行化的部分：

- 内层的 l 循环是一个串行的过程，因为每次 l 循环的结果会影响下一次 l 循环的起始点，这会导致竞争条件，所以不能直接并行化。
- 对于计算 acc 的部分也是串行的，因为每个 acc 的计算依赖于上一个 acc 的值。

可能产生空等的地方：

- 在内层的 l 循环中，如果稀疏矩阵 B 中的非零元素分布不均匀，可能会导致某些线程的 l 循环执行时间较长，其他线程可能会空等。
- 在内层循环中，由于访问 C[i][j] 是串行的，可能会导致某些线程在等待其他线程完成后才能进行下一步的计算。

负载是否均衡划分：

- 外层循环的并行化是相对容易实现均衡的，因为每个线程可以处理不同的行。
- 内层循环的负载均衡则取决于稀疏矩阵 B 中非零元素的分布情况，如果分布均匀，则负载较为均衡，否则可能会出现负载不均衡的情况。

额外的开销：

- 并行化可能需要额外的同步机制来保证数据的一致性，特别是在内层循环中访问稀疏矩阵 B 的数据时，需要避免数据竞争。
- 另外，如果使用多线程并行化，还需要考虑线程创建、销毁以及线程间通信的开销。

1.2 算法描述

PCA框架

Problem (问题):

- 问题是矩阵乘法中的性能瓶颈，特别是当稀疏矩阵的稀疏性很高时，传统的稠密矩阵乘法算法效率较低。

Countermeasure (对策):

- 对策是设计一种针对稀疏矩阵乘法的高效算法，以提高性能。

Analysis (分析):

- 分析了稀疏矩阵乘法的特点，发现稀疏矩阵的大部分元素为零，因此可以利用稀疏矩阵的结构来减少乘法运算次数，从而提高性能。

Mechanism (机制):

- 算法采用了基于稀疏矩阵结构的优化策略：
  - 利用稀疏矩阵的列偏移量和行索引，遍历稀疏矩阵的非零元素；
  - 对于每个稀疏矩阵的列，通过行索引找到稠密矩阵对应的行，并计算点积的一部分；
  - 最终将点积结果存储到结果矩阵中。

伪代码

```
对于全局函数 denseSparseMatrixMultiplyKernel:

输入参数：
    d_D：指向稠密矩阵的指针
    d_line_ptr：指向稀疏矩阵列偏移量的指针
    d_index：指向稀疏矩阵行索引的指针
    d_val：指向稀疏矩阵非零元素值的指针
    d_Result：指向结果矩阵的指针
    M：稠密矩阵的行数
    N：稠密矩阵的列数/稀疏矩阵的行数
    P：稀疏矩阵的列数/结果矩阵的列数

计算当前线程的行索引 i 和列索引 j：
    i = blockIdx.y * blockDim.y + threadIdx.y;
    j = blockIdx.x * blockDim.x + threadIdx.x;

如果 i 小于 M 且 j 小于 P：
    初始化累积变量 acc 为 0；
    对于稀疏矩阵列 j 中的每个非零元素：
        取出当前非零元素的行索引 idx；
        取出当前非零元素的值 val；
        使用稠密矩阵中第 i 行和稀疏矩阵中第 j 列对应的元素相乘，然后加到 acc 中；
    将 acc 存储到结果矩阵 d_Result 中的第 i 行第 j 列处；
```

2.实验结果

强可扩放性分析，即在固定问题规模的情况下（OJ 平台），分析程序在不同核数下的性能表现。分析未能达到线性加速的可能因素。如果出现超线性加速，分析可能原因。

BLOCK_SIZE 此处设置为36

核数	时间/ms
(P + BLOCK_SIZE - 1)^2	4565ms
2*(P + BLOCK_SIZE - 1)^2	4023ms
4*(P + BLOCK_SIZE - 1)^2	3904ms
6*(P + BLOCK_SIZE - 1)^2	4234ms
8*(P + BLOCK_SIZE - 1)^2	4565ms

核数增加时，并没有观察到线性加速的情况。在一定程度上，随着核数的增加，性能反而略微下降了。这可能有几个原因：

1. **通信开销**：随着核数的增加，通信开销可能会增加。特别是在并行计算中，核之间需要进行通信以协调任务和共享数据。如果通信开销增加超过了计算开销的减少，那么性能就不会线性增长。
2. **缓存争用**：多个核同时访问共享的内存区域时可能会导致缓存争用。当核数增加时，这种情况可能会更加明显，因为更多的核试图同时访问相同的数据，从而导致性能下降。
3. **任务划分不均匀**：如果任务被划分为多个子任务，并且这些子任务在各个核上执行的时间不均匀，那么一些核可能会比其他核花费更多的时间在等待其他核完成它们的任务上，从而降低整体性能。

3.总结

选好合适的算法进行并行化，不要只是并行已有的串行代码，稀疏矩阵相乘不同于稠密矩阵相乘