技术报告文档

一、基本算法介绍

图神经网络（GCN）的推理过程主要是通过归一化的邻接矩阵和节点特征矩阵的卷积操作，逐层更新节点的特征表示。其输入包括节点特征矩阵和图的结构。在训练开始前，需要对矩阵进行归一化处理。GCN 通过堆叠多个卷积层来进行推理。在每一层，节点的特征表示通过邻接节点的信息进行更新。本程序实现的 GCN 固定由两个图卷积层构成，第一层的激活函数使用 ReLU，第二层使用 LogSoftmax。

二、设计思路和方法

算法分为六部分，包括图数据预处理、内存初始化、读取输入特征和权重矩阵、GCN 第一层、GCN 第二层和结果验证等步骤。首先从文件中读取图数据，将其转换为适合矩阵计算的格式，并将边权重归一化，之后读取并初始化特征矩阵。随后进行 GCN 的两层卷积操作，首先将节点特征矩阵与权重矩阵相乘，再将所得矩阵与邻接矩阵相乘。对于第一层，对所得矩阵应用 ReLU 激活函数；对于第二层则应用 LogSoftmax 函数。最后进行结果验证，计算 GCN 第二层特征矩阵的最大行和。

三、算法优化

1．**采用并行计算**：使用 OPENMP 实现并行化计算，在执行循环重复计算时，通过#pragma omp parallel for 或#pragma omp parallel for schedule(dynamic)指令实现程序执行并行化，可大幅提升处理大规模数据时的性能。

2．**使用 CSR 格式存储稀疏矩阵**。CSR（Compressed Sparse Row，压缩稀疏行）是一种常用于存储稀疏矩阵的存储格式，主要用于节省空间和提高运算效率。该格式可通过仅存储非零元素及其位置来实现空间压缩，相比于邻接表存储法有效提升了计算效率。

3．**使用 AVX-512 指令集实现向量化**：AVX-512 是 Intel 开发的一组用于处理单指令多数据（SIMD）运算的扩展指令集。利用 SIMD 技术，向量化允许同一指令在多个数据元素上同时执行，从而减少了循环中的指令数量和内存访问次数，提高了数据吞吐量和并行度。此外，向量化通过减少循环迭代次数，降低了循环控制开销，显著提高了程序性能。

四、详细算法设计与实现

1．图数据读取与初始化：首先，通过 readGraph 函数从文件中读取图数据，以边列表形式存储在 raw_graph 向量中，并对变量进行初始化。

2．转换图存储格式：在 somePreprocessing()函数内调用 raw_graph_to_CSR 函数，将边列表转换为 CSR 格式存储，并通过 edgeNormalization 函数进行边权重归一化，加快后续的稀疏矩阵计算过程。

3．GCN 推理过程：进行两层 GCN 计算。对于第一层，首先通过 XW 函数进行节点特征矩阵 X0 与权重矩阵 W1 的乘法，得到中间结果 X1_inter，实现特征转换步骤。在 XW 函数中，使用 AVX-512 指令集实现了向量化。此外，由于在执行矩阵乘法时使用的_mm512_loadu_ps 函数对于矩阵列元素寻址有误，本程序通过 transpose_matrix 函数手动对矩阵取转置后再进行乘法运算。之后执行 AX_CSR 函数进行节点特征矩阵 X1_inter 与邻接矩阵的乘法，得到特征矩阵 X1。在 AX 函数中，参与计算的矩阵以 CSR 格式存储，使用#pragma omp parallel for 指令并行化最外层循环，并使用 AVX-512 指令集实现了向量化。随后对 X1 应用 ReLU 激活函数。对于 GCN 第二层，同样依次执行 XW 函数和 AX_CSR 函数，并对所得特征矩阵 X2 应用 LogSoftmax 函数。

4．结果验证：通过 MaxRowSum 函数计算特征矩阵 X2 的最大行和。并通过记录 GCN 计算开始和结束的时间点，输出 GCN 推理过程所耗费的时间。

五、实验结果与分析

经多次运行，程序实现的 GCN 优化计算过程计算结果与样例程序保持一致，输出的最大的顶点特征矩阵行和均为-16.68968964。

```
[guanzeyu@node11 example]$ ./your_team_name.exe 64 16 8 graph/1024_example_graph.txt embedding/1024.
bin weight/W_64_16.bin weight/W_16_8.bin
-16.68968964
24.00780600
[guanzeyu@node11 example]$ ./JLUzgz.exe 64 16 8 graph/1024_example_graph.txt embedding/1024.bin weig
ht/W_64_16.bin weight/W_16_8.bin
-16.68968964
7.45581000
```

实验结果表明，GCN 的优化计算过程相比原始程序的执行时间减少近 70%，计算效率得到显著提高。

六、程序代码模块说明

1．引入头文件和定义全局变量。

2．readGraph 函数：从文件中读取图的顶点数和边数，并将边的起点和终点存储在 raw_graph 中。

3．edgeNormalization 函数：遍历每一个顶点的所有出边，并根据该顶点和目标顶点的度数对边的权重进行归一化，将归一化后的边权重存储在 values 数组中。使用 OpenMP 并行化计算。

4．raw_graph_to_CSR 函数：将原始图的数据结构转换为 CSR 格式，以便于后续计算的高效进行。其中 col_indices 数组存储 CSR 格式中的列索引，每个元素表示一条边的目标顶点；row_ptr 数组存储 CSR 格式中的行指针，每个元素表示对应顶点的出边在 col_indices 和 values 数组中的起始位置；degree 数组则存储每个顶点的出度。

5．transpose_matrix 函数：对矩阵进行转置操作。

6．readFloat 函数：从文件中读取浮点数数据。

7．initFloat 函数：初始化浮点数数组，将其所有元素设为 0。

8．XW 函数：实现矩阵乘法运算，计算输入特征矩阵和转置后的权重矩阵的乘积，输出得到的特征矩阵，并使用 OpenMP 并行化和 AVX-512 指令进行向量化，以提升计算效率。

9．AX_CSR 函数：实现稀疏矩阵乘法，计算输入特征矩阵和归一化后的邻接矩阵的乘积，并使用使用 OpenMP 并行化和 AVX-512 指令进行向量化。

10．ReLU 函数：实现 ReLU 激活函数，并使用 AVX-512 指令进行向量化优化。

11．LogSoftmax 函数：实现 LogSoftmax 归一化。

12．MaxRowSum 函数：计算矩阵每行元素的和，并返回最大行和。

13．freeFloats 函数：释放动态分配的内存。

七、详细程序代码编译说明

在 JLUzgz 目录下，执行 makefile 文件，或执行 g++ -o ../JLUzgz.exe JLUzgz.cpp -mavx512f -mfma 指令均可完成程序代码编译。

八、详细代码运行使用说明

在 cgc_JLUzgz 目录下，执行

./JLUzgz.exe 64 16 8 graph/1024_example_graph.txt embedding/1024.bin weight/W_64_16.bin weight/W_16_8.bin

命令即可运行程序。