中国图计算挑战赛提交报告

参赛队名：xzx

指导老师：巩树凤

队长：谢山

队员：张祥志、许愿

一、 基本算法介绍

在本次图卷积神经网络推理问题的计算优化过程中，我们采用了CSR(Compressed Spare Row)的数据结构来储存图结构；在计算 XW 和 AX 的过程中，涉及到矩阵乘积的计算，我们使用了 SIMD 指令集和 openMP 进行并行化高性能计算。实现并行化的过程中，主要运用到了 Work Sharing 和 Work Stealing 的思想来加速程序运行。

二、 设计思路和方法

在本次代码的优化过程中，我们将预处理、XW、AX、RELU、LogsoftMax、MaxRowSum 所占用的时间各项输出。

首先优化时间占比较大的预处理，在预处理的过程中，我们意识到使用邻接表来存储稀疏图结构以及后续访问表内元素的时间成本较高，所以考虑将原始图结构转化成 CSR 结构储存，以加快后续访问图结点的速度；

其次是优化 XW 这步计算矩阵乘积同样时间占比较高的部分，在高性能计算矩阵乘积的领域内，已经有比较成熟的思路，比如交换循环次序以充分利用 Cache、开启线程并行充分调度多核计算、将 1*1 的单步计算转化成 1*4 甚至是 4*4 的多步同时计算、利用 SIMD 指令集进行多步计算等等；

然后是对 AX 部分的优化，同样作为矩阵乘积的运算，AX 部分不同于 XW，无法使用同样的思路进行优化。我们采取了线程并行以及应用指令集优化的方式来加速这部分。

最后是其他一些时间占比较低的模块，对这部分进行优化的效果微乎其微，甚至在优化的过程中加入的操作时间成本高于优化效果，所以只进行了简单的并行优化。

三、 详细算法设计与实现

- somePreprocessing

1、算法目标

将原始图数据存储在 raw_graph 数组中，其中 raw_graph 是一维数组，每两个元素表示一条边的源节点和目标节点。算法将这些边转换为压缩稀疏行（CSR）格式的图表示，其中包括三个主要部分：csrgraph.rowPointers 用于存储每个节点的邻接边的起始位置索引，csrgraph.mergedArray 用于存储所有邻接边的目标节点和相应的权重，degree 用于存储每个节点的度。

2、算法步骤

a. 初始化数据结构：

counts：用于临时存储每个节点的边计数，即度，初始化为 0。

tmp_degree：用于存储每个节点的倒数度值，初始化为 0。

csrgraph.mergedArray：用于存储压缩后的边数据。

csrgraph.rowPointers：用于存储每个节点的邻接边的起始位置索引，初始化为 0。

degree：用于存储每个节点的度，初始化为 0。

b. 计算每个节点的度：

并行遍历原始图的所有边，通过统计每个节点的出度（边的源节点）来计算节点的度。

c. 计算每个节点的倒数度：

并行遍历所有节点，计算每个节点的倒数度值并存储在 tmp_degree 数组中。

d. 计算 CSR 格式的 rowPointers 数组：

并行计算每个节点的邻接边的起始位置索引。rowPointers[i] 存储的是节点 i 的邻接边在 mergedArray 中的起始位置索引。

e. 将边数据转换为 CSR 格式：

并行遍历原始图的所有边，为每个边找到其源节点，并找到相应的位置存储在 mergedArray 中。

使用 counts 数组来跟踪每个节点已经插入的邻接边数量，并确保边在 mergedArray 中的正确位置。

- XW

1、算法输入和输出：

in_dim：输入矩阵 in_X 的维度（输入特征的数量）。

out_dim：输出矩阵 out_X 的维度（输出特征的数量）。

in_X：输入矩阵，其大小为 v_num x in_dim，其中 v_num 是输入数据的样本数。

out_X：输出矩阵，其大小为 v_num x out_dim。

W：权重矩阵，其大小为 in_dim x out_dim。

2、算法步骤：

a. 使用指针类型转换：

将输入矩阵 in_X、输出矩阵 out_X 和权重矩阵 W 转换为二维数组指针类型，以便在后续代码中使用二维数组的方式访问数据。

b. 并行计算矩阵乘法：

使用 OpenMP 并行处理指令#pragma omp parallel for，将计算过程分配给多个线程并行执行。

c. 矩阵乘法展开：

将矩阵乘法的计算过程展开以利用 SIMD 指令的并行计算能力。

循环中的变量 i 表示输入数据的样本索引，j 表示输出特征索引，k 表示输入特征索引。

针对输出特征索引 j，使用步长为 4 的展开方式（SIMD 寄存器可以同时处理 4 个单精度浮点数），在每个循环中计算 4 个输出特征的乘积和累加。

_mm_setzero_ps() 用于初始化累加寄存器，_mm_set1_ps()用于加载输入值，_mm_load_ps()用于加载权重，_mm_add_ps()和_mm_mul_ps()分别用于执行累加和乘法操作，_mm_store_ps()用于将累加结果存储到输出数组中。

d. 处理剩余的输出特征：

对于无法用步长为 4 的方式展开的剩余部分（输出特征索引 j 为 tmp1 到 out_dim-1），使用常规的循环方式计算矩阵乘法。

- AX

1、算法输入和输出：

dim：输入向量和输出向量的维度大小。

in_X：输入向量，包含 dim 个单精度浮点数。

out_X：输出向量，包含 dim 个单精度浮点数。

2、算法步骤：

a. 使用指针类型转换：

将输入向量 in_X 和输出向量 out_X 转换为二维数组指针类型，以便在后续代码中使用二维数组的方式访问数据。

b. 并行计算稀疏矩阵-向量乘法：

使用 OpenMP 并行处理指令#pragma omp parallel for，将计算过程分配给多个线程并行执行。

c. 稀疏矩阵-向量乘法：

- 对于每个节点（样本）i，遍历其邻接边，并执行稀疏矩阵的行与输入向量的乘法。

- 使用 csrgraph.rowPointers 数组存储了稀疏矩阵 csrgraph.mergedArray 中每个节点（行）的邻接边在 mergedArray 中的起始位置索引，csrgraph.rowPointers[i]表示节点 i 的邻接边起始位置，

csrgraph.rowPointers[i + 1]表示节点 i 的邻接边结束位置。

- 对于每个邻接边，找到目标节点 nbr.columnIndices（列索引）和对应的权重 nbr.weight。
- 使用步长为 4 的 SIMD 指令展开循环计算稀疏矩阵-向量乘法，以利用并行计算能力。
- 在每个循环中，从输入向量和输出向量中加载数据，并使用 SIMD 指令执行加法和乘法操作，然后将结果存储回输出向量。

d. 处理剩余部分：

- 对于无法用步长为 4 的方式展开的剩余部分，使用普通的循环方式计算稀疏矩阵-向量乘法。

四、 实验结果与分析

程序 / Time/ms	源程序	优化后
preprocessing	2028.54	42.8914
Xw1	2728.64	17.4134
Ax1	893.912	28.7254
Relu	86.9769	3.86788
Xw2	330.754	4.90384
Ax2	656.861	17.4044
Logsoftmax	85.8867	3.92187
Maxrowsum	14.7592	2.69954

通过对比可以发现，优化后的代码在各个阶段的计算时间上都明显减少，特别是 xw1、ax1 和 ax2 等矩阵计算阶段的耗时大幅下降。优化后的代码在利用 SIMD 指令进行向量化计算、使用 CSR 格式存储

图数据等方面进行了改进，从而提高了计算效率。相比之下，原始代码在各个阶段的计算时间上都较长，运行时间较长。

因此，通过对原始代码进行优化，可以显著提升图神经网络的计算性能，减少计算时间。

五、 程序代码模块说明

该程序代码为神经网络图算法的 C++实现，采用了 OpenMP 进行并行处理，并使用了 SIMD 指令（SSE 和 AVX）进行优化计算。

1. 数据结构：

- `struct Edge`： 代表图中具有权重和列索引的边。

- `struct CSRGraph`： 表示图的压缩稀疏行（CSR）表示法，通常用于稀疏矩阵存储。

2. 函数

- `readGraph(const char *fname)`： 从文件中读取图数据并将其存储在`raw_graph`向量中。

- `readFloat(char *fname, float *&dst, int num)`：从二进制文件中读取浮点数据并存储到`dst`数组中。

- `initFloat(float *&dst, int num)`： 用零初始化`dst`数组。

- Raw_Graph_To_CSR()`： 将图形数据从原始格式转换为 CSR 格式，填充`csrgraph`和`degree`向量。

- `XW(int in_dim, int out_dim, float *in_X, float *out_X, float *W)`： 在输入矩阵`in_X`和权重矩阵`W`之间执

行矩阵乘法(XW)，并将结果存储在`out_X`中。

- `AX(int dim, float *in_X, float *out_X)`： 在输入矩阵`in_X`和图的邻接矩阵之间执行矩阵向量乘法(AX),并将结果存储在`out_X`中.

- `ReLU(int dim, float *X)`： 对输入数组`X`的元素应用 ReLU 激活函数。

- `LogSoftmax(int dim, float *X)`： 对输入数组`X`的元素应用 LogSoftmax 激活函数。

- `MaxRowSum(float *X, int dim)`： 计算矩阵`X`中行的最大和。

3. 主函数 (`main`)：

- 使用`readGraph`和`readFloat`从文件中读取输入参数和数据。

- 使用`initFloat`初始化各种数组的内存。

- 使用`somePreprocessing`进行一些预处理，将原始图形数据转换为 CSR 格式。

- 作为神经网络计算的一部分，执行一系列操作（矩阵乘法、激活等）。

4. 并行化：

- 代码使用 OpenMP(`#pragma omp`)对部分计算循环进行并行化，加快在多核处理器上的执行速度。

- SIMD（单指令、多数据）指令（SSE 和 AVX）用于优化兼

容硬件上的计算，这些硬件可以同时处理多个数据元素。

六、 详细程序代码编译说明

1. `g++`: 用于编译 C++代码的 GNU Compiler Collection (GCC)编译器命令。

2. `-O3`: 该选项启用第 3 级积极优化。优化级别控制编译器为优化代码的速度和空间所做的努力。O3 ″代表高优化级别，可以显著提高生成代码的性能。

3. fopenmp ″选项： 该选项使代码支持 OpenMP 指令。OpenMP 是一个允许开发人员编写并行程序和利用多核处理器的 API。通过启用该选项，代码可以从使用 OpenMP 实用程序的并行化中获益。

七、 详细代码运行使用说明

在工作目录下执行 source.exe 文件，并且依次传入输入顶点特征长度、第一层顶点特征长度、第二层顶点特征长度、图结构文件名、输入顶点特征矩阵文件名、第一层权重矩阵文件名、第二层权重矩阵文件名。