技术报告文档

矩阵装置

在进行矩阵乘法 C = A * B 时，对于每个元素 C(i, j)，需要访问矩阵 A 第 i 行和矩阵 B 第 j 列的元素，并进行乘法累加。矩阵 A 和矩阵 B 这样的二维数据结构，在内存上，是以行优先按照一维存储的。因此，在访问矩阵 A 的第 i 行时，对于内存的访问是连续的，而在访问矩阵 B 的第 j 列时，对于内存的访问是不连续的。因此在将矩阵 B 转置之后，对于矩阵 B 的第 j 列的访问就变成了对 B 转置后矩阵的第 j 行的访问，对于内存的访问是连续的，可以较快的提高访问速度，并且有利于缓存的命中，提高缓存的命中率，提高内存访问效率。

```
float tmp_W_trans[out_dim][in_dim];

#pragma omp for collapse(2)
for (int i = 0; i < in_dim; i++) {
  for (int j = 0; j <out_dim; j++) {
      tmp_W_trans[j][i] = tmp_W[i][j];
  }
}
```

OpenMP 并行

OpenMP (Open Multi-Processing) 是一种用于并行计算的编程接口，可用于共享内存多核系统的并行化。它通过使用指令和编译器指示，允许程序员在现有的顺序代码中标记需要并行执行的部分。我们对代码中的 for 循环进行了 OpenMP 处理。

```
136 void AX(int dim, float *in_X, float *out_X) {
137   float(*tmp_in_X)[dim] = (float(*)[dim])in_X;
138   float(*tmp_out_X)[dim] = (float(*)[dim])out_X;
139   #pragma omp parallel for
140   for (int i = 0; i < v_num; i++) {
141     vector<int> &nlist = edge_index[i];
142     for (int j = 0; j < nlist.size(); j++) {
143       int nbr = nlist[j];
144       for (int k = 0; k < dim; k++) {
145         tmp_out_X[i][k] += tmp_in_X[nbr][k] * edge_val[i][j];
146       }
147     }
148   }
149 }
```