稀疏矩阵乘法的高性能实现

摘要：一共实现了 CPU 多进程、CPU OpenMP、KNL 三个版本。其中 AT+A 的性能在 CPU OpenMP 中是最好的，ATA 的则是 KNL 上最快。报告中重点介绍 CPU 多进程的算法，KNL 上的算法基本思想是类似的。需要特别注意 KNL 上处理对象和指针数组的时候特别慢，因此 CPU 上的实现不能直接照搬，大量的 array of structure 需要转换成 structure of array。这里介绍的 CPU 算法的一大缺点是算法开始收集矩阵 A 时通信量大。但是 KNL 上由于是单机的，不存在通信问题，因此在 KNL 上能有不错的表现。

AT+A

AT+A 的各行在进程间的分布与 A 的各行在进程间的分布一置，每个只进程负责计算 AT+A 分布在本进程的那一部分，因此计算结束之后不需要进行通信。所有的通信代价都发生在计算之前。具体步骤如下：

1. 首先从其他进程收集矩阵的其余部分，存储在本进程的内存空间中。由 0 号进程负责收集完整的矩阵再发送给其他进程。把进程本地存放矩阵 A 所有元素 value 的数组叫做 A_all_value，把本地存放矩阵 A 完整 col_idx 的数组叫做 A_all_col_idx。
2. 对本进程的每一行 i 维持一个集合 K[i]。遍历 A_all_col_idx, 对于其中的每个元素 j，设 j 所在的行为 row_j，对应的元素值为 value_j, value_j 在 A_all_value 中的地址是 addr_j。如果 j 在 A 在本进程拥有的行编号范围之内，则把(row_j, value_j, addr_j)这一三元组放入 K[j]中。这一部分操作可以由多个线程同时处理，最后把每个线程的结果合并起来构成最后的集合 K。这样，对集合 K[j]中的每个元素(a, b, c)，a 表示 AT 中第 j 行的第 a 个元素非零，对第 j 行的最终结果有贡献。b 是 AT 中第 j 行第 a 个元素的值，c 是这一值在 A_all_value 中的地址。
3. 对本进程 A 中的每一行 i，对于本进程 row_col 在第 i 行的每一个元素 j，把(j, value_j, addr_j)放进 A[i]中。其中 value_j 是 A[i][j]的值，A[i][j]在 A 本地 value 中的地址。如果 A[i]中已经有一个元素(k, value_k, addr_k)使得 k 等于 j，则把这两个元素合并为(j, value_k + value_j, {addr_j, addr_k})。
4. 对本进程 A 中的每一行 i，把 K[i]按照三元组的第一个元素进行排序，这样 K[i]中每个三元组的第一个元素就依次表示 AT+A 中第 i 行所有非零元的列号，而每个三元组的第二个元素就依次表示 AT+A 中第 i 行所有非零元的值。即可完成计算。
5. 把 K[i]中每个三元组的最后一个元素抽取出来放在一个数组 A_addr 中，就记下了计算 AT+A 中的每个元所需要的操作数在 A_all_value 中的地址。下次扰动矩阵 A 的时候，只要通信得到 A 的所有值，并存放在 A_all_value 中，然后遍历 A_addr，就可以直接取得计算 AT+A 中每个非零元的操作数，无需再进行步骤 1 到 4。

综上 1 到 4 都属于预处理的范畴。只有步骤 5 是每次扰动之后需要的，这一算法在单进程情况下可以把加法做到很快。这是因为 A 的所有非零元都存在本进程中，所以不需要通信就可以直接遍历 A_addr 进行计算。这也是 CPU OpenMP 版本中实现的算法。

ATA

如果把 A 的各行表示为$a_1^T, \dots, a_n^T$，则$\mathrm{A}^T A = a_1 a_1^T + \cdots + a_n a_n^T$。在 CSR 中 A 是按行存储的，依次遍历 CSR 中每一行a_i^T，就可以知道$a_i a_i^T$的非零元位置，进而整理得到$\mathrm{A}^T A$的非零元位置。在这一过程中，同时记录下对$\mathrm{A}^T A$每个非零 A(row, col)有贡献的所有a_i^T的行号 i，以及元素 A(i, row)在a_i^T当中是第几个非零元。有了这些信息，扰动之后时候只需遍历$\mathrm{A}^T A$中每个非零元的位置，然后在 CSR 中找到对每个非零元有贡献的 A 的行a_i^T，就容易计算出$\mathrm{A}^T A$中每个非零元的值。具体步骤如下：

1. 首先从其他进程收集矩阵的其余部分，存储在本进程的内存空间中。由 0 号进程负责收集完整的矩阵再发送给其他进程。把进程本地存放矩阵 A 所有元素 value 的数组叫做 A_all_value，把本地存放矩阵 A 完整 col_idx 的数组叫做 A_all_col_idx。
2. 对 A 在本进程的每一行 i，维护一个集合 K[i]。用 A_all_col_idx 遍历 A 的每一行a_j^T。A_all_col_idx 第 j 行的每个非零元 k，如果 k 在本进程存放的 A 的行号范围之内，则把(j, s)放入 K[k]中，其中 s 是 k 在 A_all_col_idx 第 j 行中的位置。这样 K[i]中就存放了一系列二元组，每个二元组的第一个元素表示$a_j a_j^T$在第 i 行有非零元的行号 j,并且这蕴涵着$\mathrm{A}^T A$的第 i 行的第 j 个元素是非零元。第二个元素表示的是 i 在 A_all_col_idx 的第 j 行中的位置。
3. 对 A 在本进程的每一行 i，把 K[i]按二元组第一个元素排序。这样就得到了$\mathrm{A}^T A$每一行的所有非零元列号。对行 i 中每一个非零元列号 j，利用 A_all_col_idx 和 A_all_value 取出行a_j^T，计算$a_j[i]$与a_j^T的乘积，并加到$\mathrm{A}^T A$第 i 行对应的位置上。

综上，只要记录下 K[i],就可以很容易地在每次扰动之后计算第$\mathrm{A}^T A$ 第 i 行的各元素。因此 1、2 两步均为预处理，只有第 3 步需要反复执行。