并行与分布式计算基础：作业 2 报告

吕洋

2022 年 12 月 31 日

目录

1 任务叙述及理论分析

我们考虑的任务是稀疏矩阵和向量之间的快速乘法（SpMV）：

$$y = Ab$$

其中 $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^{m}$, 这里 A 是一个稀疏矩阵。度量稀疏矩阵的一个指标即稀疏度 (sparsity)，为

$$k = \frac{\text{Nonzero entries number}}{n \times m}$$

利用稀疏矩阵存储格式，可以用较少内存存储规模很大的矩阵。显然，串行的 SpMV 的时间复杂度是 $\mathcal{O}(kmn)$。

2 并行划分方式

我们知道 `OpenMP` 是使用 `fork-join` 并行模型的。而对于 SpMV 这个问题来说, 最直观的一种并行方式就是让每个线程执行不同行的计算, 因为行与行之间的计算是独立的. 我们即采取这种并行划分方式, 示意图如下. 简单分析一下这种划分方式的优缺点:

- 优点: 程序实现简单, 并行效率较高.
- 缺点: 由于矩阵 A 每一行的非零元素个数可能分布地不均匀, 这会导致每个线程的任务分配不均, 延长等待和同步时间.

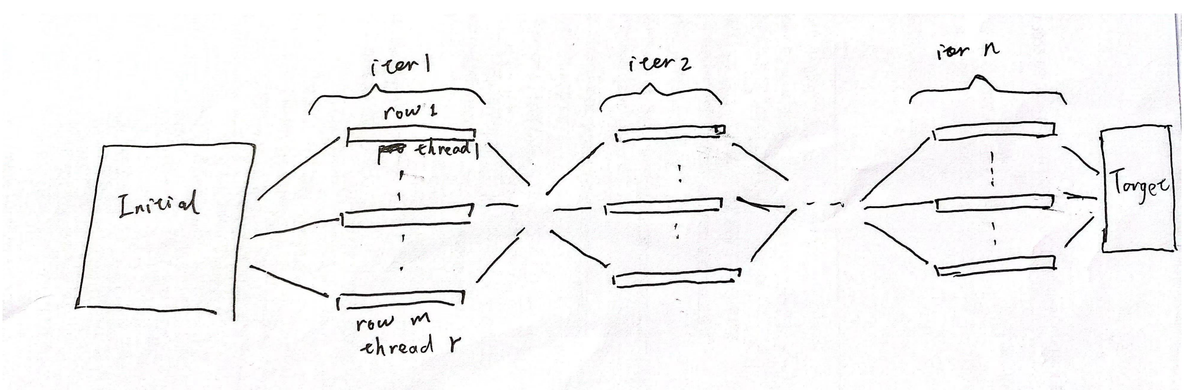

图 1: 一个简单的示意图

3 重要代码

```
void spmv_omp(const int *row_ptr, const int *col_ind, const double *values, const int num_rows, const double *x, double *y) {

#pragma omp parallel
{
    int i,r_start,r_end;
    double sum;
    #pragma omp for private(i,sum,r_start,r_end) nowait
    for ( i = 0; i < num_rows; i++)
    {
         sum = 0;
         r_start = row_ptr[i];
         r_end = row_ptr[i + 1];
        for (int j = r_start; j < r_end; j++) {
            sum += values[j] * x[col_ind[j]];
        }


        y[i] = sum;
    }

}
}
```

这是串行函数 `spmv_csr` 的并行版本. 这里我们使用经典的 `openmp` 的 for 循环, 显式声明了变量为 private 变量, 并使用了 `nowait`, 理由是经过实验, 这样的设定不会对实验的精确性造成影响, 并且能显著加快速度.

4 实验设定及理由

- 编译选项为`-fopenmp -lm -O3 -std=c99 -Ofast -march=native`. 这是为了尽量提高性能, 但经过实验,`-Ofast` 和 `-march=native` 没有显著的效果.
- 并行部分的计时采用的是 `omp_get_wtime()` 来进行计时, 这是应该使用的正确函数.
- 稀疏矩阵使用 CSR 格式, 这从并行方式来看是十分自然的.

5 实验结果及分析

最终的实验结果是:

Serial wall time(ms)	Parallel wall time (ms)	Max error	Max thread number
70.0000 ms	17.6671 ms	0.0000	12

对实验结果的解读, 我们需要注意以下几点:

- 实验结果与集群愿意分配给我们的核心数量密切相关, 因此相同的输入并不一定保持一致
- 设置的环境变量 `OMP_NUM_THREADS` 也并不一定就是实际的线程数量, 而应该理解为线程数量的上限.

6 通用性及限制

- 通用性: 我们没有针对 A 的具体形式进行调参, 因此这样的程序作为 API 时, 可以对不用的输入 A 生效.
- 限制: 没有进行向量化优化, 将来可参考开源库的相关函数进行优化.

7 总结

我们分析了问题，制定了不同的并行算法，并进行了较为仔细的分析，得出了令人满意的加速策略。关于源代码，请参考文件夹中的 readme 文件。