南開大學
Nankai University

计算机学院

并行程序设计期末报告

期末项目开题报告

姓 名： 赵 元 鸣 2211757

苏 胤 华 2213893

专业 ：计算机科学与技术

2024 年 4 月 6 日

目录

1 研究背景

1.1 普通高斯消元法

高斯消元法是求解线性方阵组的一种算法， 它也可用来求矩阵的秩， 以及求可逆方阵的逆矩阵。它通过逐步消除未知数来将原始线性系统转化为另一个更简单的等价的系统。它的实质是通过初等行变化，将线性方程组的增广矩阵转化为行阶梯矩阵.

基础的高斯消元法可以分为以下几个步骤： 1. 根据线性方程组构造增广矩阵； 2. 通过初等变换逐步将系数矩阵变为上三角矩阵； 3. 得到简化的三角方阵组（对角线元素值为 1）；4. 使用后向替换算法求解， 即从尾向前逐步求解每个未知变量， 这种方法也称为顺序消去法。顺序消去法优点在于编程实现思路简单， 但是有两个缺点， 首先必须保证对角线元素不为 0，否则在回带过程中会导致分母出现 0；其次如果某个系数绝对值过小，会在回带过程中引起较大的误差，影响算法稳定性。

伪代码

Algorithm 1 初等行变换高斯消元法

Input: 待消元矩阵

Output: 行阶梯矩阵

```
1:  function Gaussian Elimination(MatrixA)
2:      for k = 1 to n do
3:          for i = k + 1 to n do
4:              A_ki / = A_kk
5:          end for
6:          A_kk = 1
7:          for i = k + 1 to n do
8:              for j = k + 1 to n do
9:                  A_ij = A_ij − A_ik * A_kj
10:             end for
11:             A_ik = 0
12:         end for
13:     end for
14:     return A
15: end function
```

在行主元基础上， 提出列主元消去的算法， 即每次选择第 k 列最大元素， 将对应行交换到第 k 行，用此方法尽可能减小误差， 而全主元小区的算法也相似， 每次从 k 行 k 列右下角矩阵中选择一个最大的元素将对应行列交换到第 k 行 k 列用作后续回带的除数。

在高斯消元法的基础之上还有高斯-若尔当消元法， 首先还是需要通过初等行变换得到上三角矩阵并对对角线归一，但是之后还需要对行主元上面的元素逐个消去， 最后得到一个元素全为 1、0 的单位阵，这种算法效率上没有基本的高斯消元法好，但是求解过程非常适合计算机编程。

1.2 特殊高斯消元法

特殊高斯消元法与普通高斯消去有许多区别，其运算均为有限域 GF（2）上的计算，即矩阵的元素只可能是 0 或者 1，其加法运算实际上为异或运算，异或运算的逆运算为自身。

输入的矩阵分为两类， 消元子和被消元行， 在输入时给定， 所有消元子的首个非零元素位置都不同， 但不会涵盖全部对角线元素， 而被消元行在消去过程中充当被减数， 但有可能恰好包含消元子并未覆盖位置的对角线 1 元素，此时该被消元行升格为消元子，补齐缺失的 1 元素。

由于实际问题中矩阵规模很大， 消元子和被消元行的数量很多， 大大超出内存， 因此会重复执行以下步骤， 抽取一个 batch，检查当前批次中每个被消元行的首项， 如果有对应消元子， 则将其异或对应消元子， 重复此过程指导变为空行或者升格为消元子， 并且如果某个被消元行变为空行则将其丢弃不参与后续计算， 重复上述过程指导所有批次处理完毕， 此时消元子和被消元行共同组成结果矩阵，可能存在很多空行。

Algorithm 2 特殊高斯消元法

Input: 消元子与消元行

Output: 行阶梯矩阵

```
1:  function Gaussian Elimination(MatrixA)
2:      for i = 1 to n do
3:          while E_i != 0 do
4:              if R_lp(E_i) != NULL then
5:                  E_i = E_i − R_lp(E_i)
6:              else
7:                  R_lp(E_i) = E_i
8:              endif
9:          endwhile
10:     endfor
11:     return E
12: end function
```

2 研究方案

2.1 程序并行分析

容易得出， 在进行基础高斯消元时嵌套了三次循环， 复杂度为 n^3，在串行计算过程中， 必须在一条指令执行之后才能继续执行后续指令， 同时对于大规模矩阵的求解， 串行算法需要大量的储存空间来储存中间结果， 需要反复访问寄存器， 从而导致效率低下， 通过观察算法， 基础高斯消元法主要由两个核心部分组成， 一是初等行变换消元， 也就是逐行做减法操作， 二是化系数为 1 时的除法， 针对算法的这两个核心步骤可以进行并行处理。而列主元消去的方式涉及到行列的交换， 交换的过程如何并行也是一个并行方向。

而对于特殊高斯消元， 如何选取批次进行消元是一个主要的问题， 由于每个批次消元后都有可能出现新的消元子， 因此在并行的过程中如何进行各个线程的信息传递以及负载均衡这类问题也显得十分重要，

2.2 实验设计

2.3 实验流程

本学期实验主要分为四个部分， SIMD、Pthread 和 OpenMP、MPI、GPU，针对这四个架构的特点（见 section3）总结为各自完成对高斯消元法批量处理、多线程、线程通信、 GPU 加速这四个环节，

针对这四个目标要对并行算法进行进一步的优化。

2.3.1 对比方式

首先是对不同算法方式的优化对比， 对基础高斯消元法、列主元高斯消元法和特殊高斯消元法的优化度进行对比。其次是代码运行环境的对比验证， 根据第一次作业的调研结果， 鲲鹏服务器的 arm 架构和使用 codeblock 下的 windows 架构可能由于优化力度不同等原因在相同程序的性能上产生较大差异， 因此选择分别在鲲鹏服务器与 codeblock 下各自进行并行优化。还有在不同问题规模下对程序进行运行验证， 在此基础上对并行算法优化的有效性进行分析。同时在具体任务当中， 根据任务的目标也需要各自设计对比实验， 在 pthread 优化中， 要选择不同进程数， 分配线程、同步的方式进行对比验证， 由于目前还没有展开对 pthread 的研究， 具体如何设计方案还需要后续设计。而在每一次的并行实验中， 结合之前实验中的并行方式， 对比使用多种并行方法和单一并行方法是否可以取得更好的并行效果也是一个可行的对比方式，例如结合 SIMD 和 Pthread 并行方法。

2.3.2 数据分析及指标

考虑到直接通过程序运行时间的对比， 对于加速效果的表现不一定足够直观， 考虑采取加速比的方式对运行结果进行呈现， 以基础的无优化时间作为基， 依次来计算并行对于程序时间的优化力度。并且考虑到相同的程序在不同规模或者对比条件下可能有不一样的优化效果， 可以根据实验结果对因素做权重划分处理，得出一个综合的加速比，用这个加速比来衡量优化的有效程度。

2.3.3 其他优化思路

为了取得更好的并行效果， 可以在实验基础上尝试一些其他的并行优化方法， 例如在 MPI 实验中， 由于内存不互通， 需要设计数据传递的过程， 是否可以通过流水线的思路对数据传递过程进行优化。对于数据储存方式， 由于数据矩阵可能非常稀疏， 使用倒排链表的方式存储消元子与消元行来减少储存空间，优化算法，但是要考虑如何进行并行的问题。对于数据访问方式，

3 四次编程作业基本方案

3.1 SIMD

SIMD（Single Instruction, Multiple Data）是一种并行计算的技术， 它在一条指令下同时处理多个数据元素。 SIMD 的基本思想是将多个相同类型的数据元素打包成一个向量， 然后通过一条指令对整个向量进行操作，以实现高效的并行计算。

根据 SIMD 的特点，我们可以选择在消元过程中对每一行的消元过程进行并行化处理， 每次从被消元行中取出多个行进行同时消元操作， 对于对角线元素的归一化处理过程， 也可以同时取出一行中的多个元素除以行主元（特殊高斯消元同理，相减类比异或，行类比消元子与被消元行）。

3.2 Pthread/OpenMP

POSIX Threads 简称 Pthreads，POSIX 是”Portable Operating System Interface”（可移植操作系统接口） 的缩写， 在 C++/C 程序中 pthread 能让进程在运行时可以分叉为多个线程执行， 并在程序的某个位置同步和合并，得到最终结果。

根据 Pthread 多线程的特点， 考虑到在高斯消元法进行行消去的过程当中， 使用一个行主元对其他每一行的消去并不冲突， 符合并行的条件， 因此可以利用多线程的思想， 将消元任务分解， 分配给多个线程共同完成， 可以采用动态分配、静态分配、信号量同步等作为进行线程的分配方式。同时， 在 Pthread 实验中可以集合 SIMD 进行进一步的优化，对比两种并行方法的优化效果。

OpenMP 是一套 C++ 并行编程框架, 能够使串行代码经过最小的改动自动转化成并行的， 具有广泛的适应性。相比于 pthread，OpenMP 改动很少， 且工作量也很少， 能方便快捷的实现并行加速。在 OpenMp 实验中也可以进行不同线程数的对比， 也可以选择不同的任务负责划分方式， 例如 static、dynamic 、 guided 等。

3.3 MPI

MPI 是一种用于编写并行程序的通信库接口， 全称为 Message Passing Interface（消息传递接口），可以理解为是一种独立于语言的信息传递标准， 有着多种具体实现。 MPI 库包含了一组函数和语法规则， 使得计算机集群上运行的多个进程可以相互通讯、协同工作。 MPI 优化相比其他的多线程优化会更加复杂，因为其各个进程之间的内存不共享，因此进程间要不停相互传递数据。

MPI 并行优化时， 会存在多个进程， 每个进程持有一定的任务量， 即行向量， 负责对这些行向量进行行消去和标准化（即对角线元素变为 1）。然而各个进程之间的数据互不连通， 因此通过消息传递方法来进行通信。在这个环节针对任务并行要解决的问题就是不同之间进程的通信协作， 例如在特殊高斯消元中如果产生了新的消元子，其他线程能否及时利用的问题。

首先可以对 MPI 是否阻塞进行对比实验， 其次任务划分的方式也有多种， 如平常的块划分， 或是循环划分等。每当一个进程将某行向量消元并且对角线元素标准化置 1 之后就会将其数据传递给其他进程，实现数据的同步，在最终消元完毕之后在 0 号进程中可以得到最终结果。

3.4 GPU

GPU 是计算机上为了渲染而生的硬件， 相比于 CPU 基于低延时设计， GPU 则是基于大吞吐量设计的， 拥有更多的计算单元用于数据处理， 适合对密集数据进行并行处理， 擅长大规模并发计算， 因此 GPU 也被应用于 AI 训练等需要大规模并发计算场景。并且 GPU 具有较大的内存带宽， 能够一次性从内存中读取更多的数据， 相比之下 CPU 的内存带宽较小， 数据读取能力差， 具有更大的内存读取开销。此外， GPU 对于并行计算任务有先天的架构优势， 天然支持 SIMD、SIMT 和并行计算， 尤其是面临巨大数据时（例如在本次学期算法分析的数据集规模可能非常大），GPU 吞吐量和计算速度都比较高。计划使用 OneAPI 进行并行的 GPU 加速实验。

4 小组分工

针对本学期实验主要分为四个部分， SIMD、Pthread 和 OpenMP、MPI、GPU，针对这四个架构的特 点（见 section3）总结为各自完成对高斯消元法批量处理、多线程、线程通信、 GPU 加速这四个环节，

针对这四个目标要对并行算法进行进一步的优化。由赵元鸣实现SIMD、OpenMP部分，由苏胤华实现Pthread和GPU部分，进行优化分别在鲲鹏服务器上和codeblock上。对比总结部分共同完成。