南开大学

Nankai University

南　　开　　大　　学

计　算　机　学　院

并行程序设计期末研究开题报告

Gröbner 基计算中的高斯消去

姓名：姚知言

年级：2022 级

专业：计算机科学与技术

指导教师：王刚

2024 年 4 月 6 日

摘要

高斯消元法是线性代数中线性方程组求解的一个重要算法。但传统高速消元法的时空复杂度较大。本次实验主要针对传统高斯消去和 Gröbner 基计算中的高斯消去展开研究，对其算法进行优化以及并行化改进。在本次报告中，将对该研究的研究历史及现状进行概括，并对后续研究计划进行规划。

关键字：高斯消去，Gröbner 基计算，并行，综述，计划

目录

一、 过往研究综述

(一) 普通高斯消去

1. 历史发展

高斯消元法（Gauss–Jordan elimination）是求解线性方程组的经典算法，它在当代数学中有着重要的地位和价值，是线性代数课程教学的重要组成部分。高斯消元法除了用于线性方程组求解外，还可以用于行列式计算、求矩阵的逆，以及其他计算机和工程方面。

高斯消元法以著名德国数学家 Carl Friedrich Gauss(1777-1855) 命名。Gauss 被认为是历史上最重要的数学家之一，他在数学的众多分支，如数论、代数、分析、微分几何等以及统计学、物理学、天文学、大地测量学、地理学、电磁学、光学等领域都有重要的贡献。Gauss 还享有“数学王子”的美誉. 值得一提的是，这种解线性方程组的消元法最早出现在中国古代数学著作《九章算术》中，相关内容在大约公元前 150 年前就出现了。

2. 运算流程与改进

普通高斯消去的流程如图1所示:

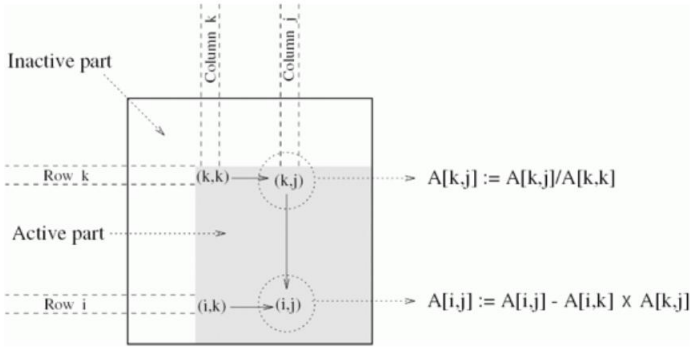

图 1: 普通高斯消去流程

该算法的主要流程是在第 k 步的时候，对第 k 行进行除法运算使得 (k,k) 位置元素变为 1，然后对后续行减去第 k 行的相应倍数，使得后续行第 k 列的元素均为 0，然后进行下一步循环。

在一般情况下，消元算法的伪代码如下:

Input: 原始矩阵 A,A 的阶数 n
Output: 消元结果矩阵 A

```
1: function LU(A, n)
2:     for k = 1, 2, ··· , n do
3:         for j = k + 1, k + 2, ··· , n do
4:             A[k][j] ← A[k][j]/A[k][k]
5:         end for
6:         A[k][k] ← 1.0
7:         for i = k + 1, k + 2, ··· , n do
8:             for j = k + 1, k + 2, ··· , n do
9:                 A[i][j] ← A[i][j] − A[i][k] ∗ A[k][j]
10:            end for
```

```
            A[i][k] ← 0.0
        end for
    end for
end function
```

将消元后的结果进行回代，求出 x 的数值，伪代码如下：

```
Input: 消元结果矩阵 A，A 的阶数 n
Output: 结果数组 X
function GEN(A, n, X)
    for i = n, n − 1, ··· , 1 do
        X[i] ← A[i][n + 1]/A[i][i]
        for j = i − 1, i − 2, ··· , 1 do
            A[j][n + 1] ← A[j][n + 1] − X[i] ∗ A[j][i]
        end for
    end for
end function
```

该方法的优势是思路清晰，对于学习过线性代数的研究者易于理解。然而，该方法在准确性上存在以下问题：

1. 每次运算时，必须保证对角线上的元素不为 0(即运算中的分母不为 0)，否则将无法产生合法的除法运算，算法无法继续进行。
2. 即使对角线上的元素不为 0，但如果该元素的绝对值很小，由于其在第 k 次运算中做除数，可能会引起较大误差。

为解决以上问题，可以通过列主元消元法或全主元消元法进行改进。

- 列主元消元法

  在第 k 步消元前，先找出 k 行下所有第 k 列元素最大的非零元素 A[i,k]，将第 i 行与第 k 行进行整行交换，这样既不影响原方程的解，也可以将绝对值最大的 A[i,k] 作为主元，放在除数的位置上，尽可能减小引入误差。

- 全主元消元法

  与列主元消元法类似，只不过全主元消元法是对待更新的右下角矩阵中所有元素中选取一个绝对值最大的 A[i,j] 作为主元，对行和列同时进行交换。

此外，该算法的时间复杂度较高，高达 $O(n^3)$，这从运行时间上来说还有较大的优化空间。

3. 并行化探索

在高斯消元法化上三角矩阵的过程中，每一次清除一列元素时，每一行的元素需要进行一次数字运算，行与行之间没有数据依赖，可以实现并行化；同理，在回溯过程中，行与行之间同样没有数据依赖，可以进行并行化。

将数据根据处理器的数量进行划分，分行处理，以实现负载均衡，提升并行效率。

(二) Gröbner 基计算中的高斯消去

1. Gröbner 基概述

Gröbner 基的概念由 B. Buchberger 于 1965 年在他的博士论文《Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal》(计算零维多项式理想的剩余类环的基的算法) 中引入, 论文的导师是著名的代数几何学家 W. Gröbner。Buchberger 在论文中设计了计算多元多项式理想的 Gröbner 基算法（称为 Buchberger 算法），并提出了优化该算法的若干准则（称为 Buchberger 第一、第二准则)。

Gröbner 基是由多元多项式理想的特殊生成元构成的集合，它具有非常良好的性质。通过计算 Gröbner 基很多有关多项式理想的基本问题都可以算法化求解。前面提到的理想成员判定问题也可以通过使用 Gröbner 基的性质获得解决：一个多元多项式属于给定生成元的多项式理想当且仅当它对该理想的 Gröbner 基的范式为 0。又譬如，字典序 Gröbner 基关于其中出现的变元具有一定的层级结构，即所谓的消元性质。基于这种性质，Gröbner 基方法又可以用来研究各种消元问题，如多项式方程组求解、参数曲线与曲面的隐式化等。

2. Gröbner 基在高斯消去中的应用

Gröbner 基计算方法相比于普通高斯消去方法进行了以下几点改进：

1. Gröbner 基中的运算均为有限域 GF(2) 上的运算，也就是矩阵元素的值只可能是 0 或 1。在该条件下，加法运算的实际执行方式为异或运算，即：0+0=0，0+1=1，1+0=1，1+1=0。同理由于异或运算的逆运算为自身，因此减法运算也是异或运算，即在高斯消去中实际上仅仅存在异或运算，将从一行中对另一行消去的运算退化为减法。
2. 矩阵行分为两类，“消元子”和“被消元行”。“消元子”在消去过程中充当减数，不会充当被减数。所有“消元子”的首个非零元素的位置都不同。“被消元行”在消元过程中充当被减数，但若其包含“消元子”中缺少的对角线 1 元素，也可升格成“消元子”。

该方法能够较好的降低算法的时间复杂度，在实际问题操作中的计算过程如下：

1. 分批次将” 消元子” 和” 被消元行” 读入内存。
2. 对每个读入的” 被消元行” 检查首项，如有对应” 消元子”，则将其与” 消元子” 进行异或操作，重复此操作直到无法进行，即发生 3/4 所述情况。
3. 若“被消元行”变成空行，将其丢弃。
4. 若“被消元行”首项无对应“消元子”，则将其升格为“消元子”。
5. 重复，直至所有批次处理完毕。

在实际执行该问题时，需要对数据存储和访问，计算进行较为合理的规划，同时也要结合输入数据格式进行优化调整。此外，需要以合理的方式对程序进行并行化优化。以上问题的研究思路将在下一部分详细展开。

二、 研究方案

(一) 优化方向

1. 数据存储

在该部分中主要考虑以下两种实现方式：

- 位向量方式

  通过位向量方式存储每个消元子和被消元行，可以将消元操作变为位向量异或操作，该方法实现较为简单，且较为适合并行化，容易达到更高的并行效率。然而，在 Gröbner 基计算中，消元子和被消元行非常稀疏，非零元素在 5% 以下，在该情况下，或许有更优的存储方式。

- 类倒排链表方式

  通过稀疏 0/1 矩阵的紧凑存储方式，或将更适合 Gröbner 基计算较为稀疏的应用场景。该方法将每个消元子和被消元行仅仅保存 1 元素的位置，按升序排列。该方法将占用更少的存储空间，然而算法涉及较为复杂，且有着较高的并行化难度。

2. 数据访问顺序与消元顺序

- 数据访问顺序

  在实际工作中，使用到的矩阵规模可能非常庞大，达到百万千万级的行列，如此多的数据难以全部放入内存。在这种情况下，将不得不考虑设计外存算法——即将数据分批次读入内存进行处理与计算。在这个过程中，需要合理的考量批次划分方法，在保证算法正确性的前提下，获取更改性能。当然，再次过程中，对于 I/O 时间的考虑也是必不可少的。

- 消元顺序

  在读取数据之后，也需要对消元顺序进行设计，以便获得更好的性能。在实验中，将先对简单的按行号顺序进行消元的方法进行测试，并在实践中逐渐发掘其他的消元顺序与之对比。

3. 并行化设计

- 数据存储的并行化设计

  首先将针对位向量存储方式实现并行化优化。当数据规模足够大的时候，将向量拆分，子向量的异或自然形成任务，并分配给不同的计算单元，可实现较好的并行化优化。当数据规模较小的时候，需要考虑消元操作间的并行，结合消元顺序设计适合的并行任务划分，在提高并发度的同时保证正确性。

  在实现以上存储方式并行化优化的同时，也将尝试对类倒排链表的方式展开并行化优化，在不断探索中确定优化方式，并将各优化方式优化比进行对比分析。

- 外存算法的并行化设计

  在对外存算法进行并行化优化的时候，需要同时考虑计算和访存的问题。在多线程并行化设计的时候，可以考虑计算和访存的异步模式，在前台进程进行消元计算时，后台线程将会读取下一步要处理的数据。这种模式可以较好的并行化实现计算和 I/O 操作的并行，从而降低运行时间。当然，在设计时也需要对计算和 I/O 步骤进行仔细分析，降低依赖关系，从而保证程序正确性的前提下提升算法效率，实现异步模式。

(二) 详细计划

实验将在 x86 和 arm 平台中分别展开，根据不同的算法设计和并行优化形式分别展开性能测试，同时借助 Vtune 和 perf 等对程序进行 profiling 分析，比较不同算法的性能和优缺点，进行总结，得出结论。

实验将在总体分为五个部分逐步展开，即 SIMD 向量优化、Pthread 多线程优化、OpenMP 多线程优化、MPI 并行优化以及 GPU 优化。

- SIMD 向量优化

  结合 SIMD 相关内容，对普通高斯消去和 Gröbner 基计算中的高斯消去进行单指令流多数据流优化设计，将平凡算法与优化算法进行比较，通过 profiling 工具结合实际运行结果进行分析。

- Pthread 多线程优化

  结合 Pthread 相关内容，对普通高斯消去和 Gröbner 基计算中的高斯消去进行多线程优化设计，将平凡算法与优化算法进行比较，通过 profiling 工具结合实际运行结果进行分析。

- OpenMP 多线程优化

  结合 OpenMP 相关内容，对普通高斯消去和 Gröbner 基计算中的高斯消去进行多线程优化设计，将平凡算法与优化算法进行比较，通过 profiling 工具结合实际运行结果进行分析。

- MPI 并行优化

  结合 MPI 相关内容，对普通高斯消去和 Gröbner 基计算中的高斯消去进行并行优化设计，将平凡算法与优化算法进行比较，通过 profiling 工具结合实际运行结果进行分析。

- GPU 优化

  结合 MPI 相关内容，对普通高斯消去和 Gröbner 基计算中的高斯消去进行 GPU 优化设计，将平凡算法与优化算法进行比较，通过 profiling 工具结合实际运行结果进行分析。