



南開大學
Nankai University

计算机学院
并行程序设计

特殊高斯消去法的并行优化

并行程序设计期末研究报告

姓名：丁屹、卢麒萱

学号：2013280、2010519

专业：计算机科学与技术

2022 年 7 月 10 日

目录

1 问题描述	2
2 研究设计	2
2.1 测试用例	3
2.2 实验环境和相关配置	3
2.3 串行稀疏矩阵算法	3
2.4 串行位元矩阵算法	3
3 算法分析	4
3.1 正确性分析	4
3.2 正确性验证	4
3.3 复杂度分析	4
3.4 运行时间分析	4
3.4.1 计时方式	4

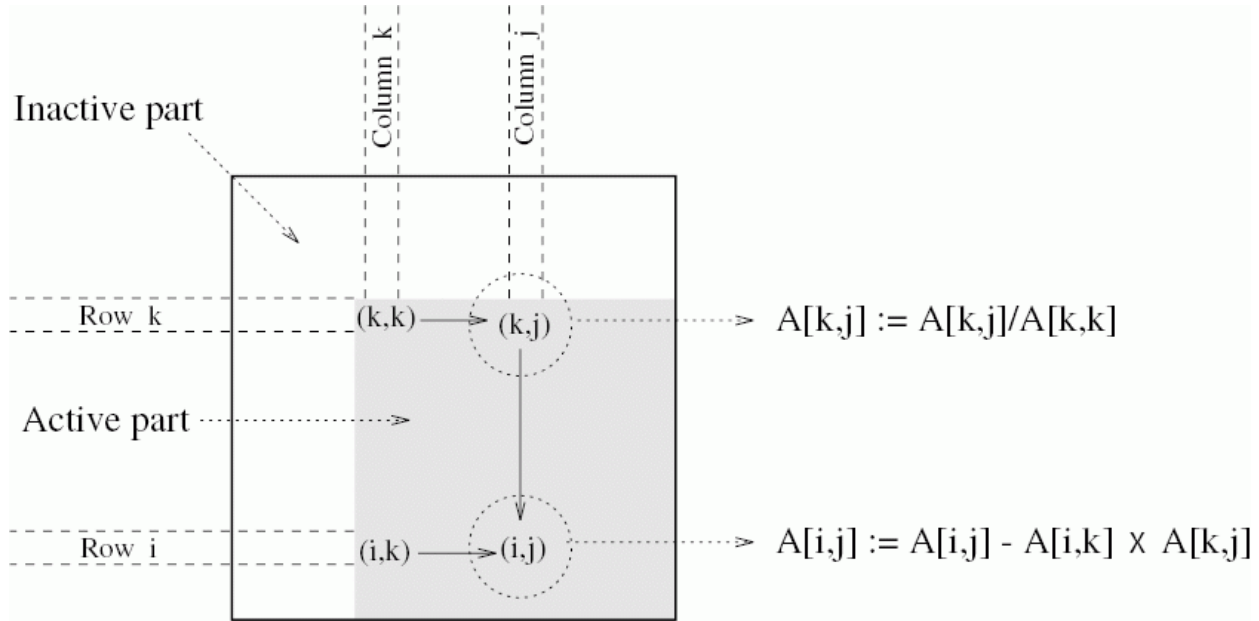


图 1.1: 高斯消去法示意图

1 问题描述

普通高斯消去的计算模式如图 1.1 所示，在第 k 步时，对第 k 行从 (k,k) 开始进行除法操作，并将后续的 $k+1$ 至 N 行进行减去第 k 行的操作，串行算法如下面伪代码所示。

Algorithm 1 普通高斯消元算法伪代码

```

1: function LU
2:   for  $k := 0$  to  $n$  do
3:     for  $j := k+1$  to  $n$  do
4:        $A[k,j] := A[k,j]/A[k,k]$ 
5:     end for
6:      $A[k,k] := 1.0$ 
7:     for  $i := k+1$  to  $n$  do
8:       for  $j := k+1$  to  $n$  do
9:          $A[i,j] := A[i,j] - A[i,k] * A[k,j]$ 
10:      end for
11:       $A[i,k] := 0$ 
12:    end for
13:  end for
14: end function

```

特殊高斯消去法

2 研究设计

项目链接: <https://github.com/NeoWans/Parallel-Programming-Final>

2.1 测试用例

测试用例由老师提供的 Groebner.7z 压缩包解压后获得, 总共 11 组数据, 软链接至 res/目录下, 命名规则为% 组号%.0 (非零消元子)、% 组号%.1 (被消元行)、% 组号%.2 (消元结果)

2.2 实验环境和相关配置

实验在本地 x86 Arch Linux 环境下完成, 使用 Makefile 构建项目, 开启 Ofast 加速; 使用的 CPU 为 AMD Ryzen 4800HS (8C16T), 系统 RAM 大小为 38.6G, 显卡为 nVIDIA RTX 2060 6G。

2.3 串行稀疏矩阵算法

使用 STL list 存储矩阵中每行的非零位置, 逐行放入嵌套的外层 STL list; 使用 STL map 存储消元行首项与消元行的映射。

Listing 1: 稀疏矩阵消元部分

```

1 void gauss(list_matrix_t& m) {
2     for (auto& eliminatee : m.op) {
3         while (!eliminatee.empty()) {
4             auto key = *(eliminatee.cbegin());
5             auto& eliminator = m.pool[key];
6             if (eliminator.empty()) {
7                 eliminator = eliminatee;
8                 break;
9             } else {
10                auto jt = eliminatee.begin();
11                auto it = eliminator.cbegin();
12                while (it != eliminator.cend() && jt != eliminatee.end())
13                    if (*it > *jt) eliminatee.insert(jt, *it++);
14                    else if (*it == *jt) jt = eliminatee.erase(jt), ++it;
15                    else ++jt;
16                for (; it != eliminator.cend(); ++it) eliminatee.push_back(*it);
17            }
18        }
19    }
20 }

```

2.4 串行位元矩阵算法

使用 STL bitset 倒序存储矩阵每行, 逐行放入外层 STL vector; 使用 STL map 存储消元行首项与消元行的映射。

其中值得注意的是, STL bitset 提供了快速查询最低真值位索引的内建成员函数 `__Find_first()`, 与之对应的是算法需要 $lp(E[i])$ 操作, 即获得被消元行第 i 行的首项, 然而正序存储时 `__Find_first` 函数得到的是被消元行第 i 行的末项, 因此需要倒序存储。具体实现使用了“bsmap”宏²处理映射关系。由于 STL bitset 需要使用常量模板参数声明, 因此使用了“matrix_max_sz”常量, 大小为 85401, 即测试样例中的最大矩阵大小。bsmap 可以保证定义域和陪域在 $[0, matrix_max_sz)$ 内且为双射, 同时满足 $\forall x \in [0, matrix_max_sz) \text{ bsmap}(\text{bsmap}(x)) = x$ 。

Listing 2: bsmmap 宏

```
1 #define bsmmap(i) (matrix_max_sz - 1 - (i))
```

Listing 3: 位元矩阵消元部分

```
1 void gauss(bitset_matrix_t& m) {
2     for (auto& eliminatee : m.op) {
3         while (eliminatee.any()) {
4             auto key = bsmmap(eliminatee._Find_first());
5             auto& eliminator = m.pool[key];
6             if (eliminator.none()) {
7                 eliminator = eliminatee;
8                 break;
9             } else eliminatee ^= eliminator;
10        }
11    }
12 }
```

3 算法分析

3.1 正确性分析

3.2 正确性验证

由于% 组号%.2 (样例正确消元结果) 被链接到 res/目录下, 而% 组号%.out (程序计算结果) 被输出到 misc/ 目录下, 使用 diff -wB misc/% 组号%.out res/% 组号%.2 即可在忽略输出格式差异的前提下判断消元是否正确。每次运行完成只需运行单行脚本 4 即可判断正确性。经过验证, 所有实现均保证了正确性。

Listing 4: 单行 Bash 脚本

```
1 for i in {1..11}; do diff -wB "misc/$i.out" "res/$i.2"; done
```

3.3 复杂度分析

3.4 运行时间分析

3.4.1 计时方式

表 1: 不同方法运行情况

Matrix rank	serial list (ms)	serial bitset (ms)	16 threads bitset (ms)
130	0.03233	0.099817	2.402953
254	1.867949	3.173488	2.272141
562	4.735001	3.08886	1.877817
1011	78.952469	98.256365	12.519157
2362	827.574329	426.11729	58.782182
3799	15708.6549	5026.35341	649.369009
8399	273776.479	30331.4359	5585.23381
23045	>1200000	220484.326	49605.6164
37960	>1200000	322201.459	77384.6553
43577	>1200000	1016832.99	252625.626
85401	96746.3559	440.782952	85.025176