南開大學
Nankai University

计算机学院

并行程序设计第 4 次作业

高斯消去法的 Pthreads 并行化

姓名：丁屹

学号：2013280

专业：计算机科学与技术

2022 年 5 月 4 日

目录

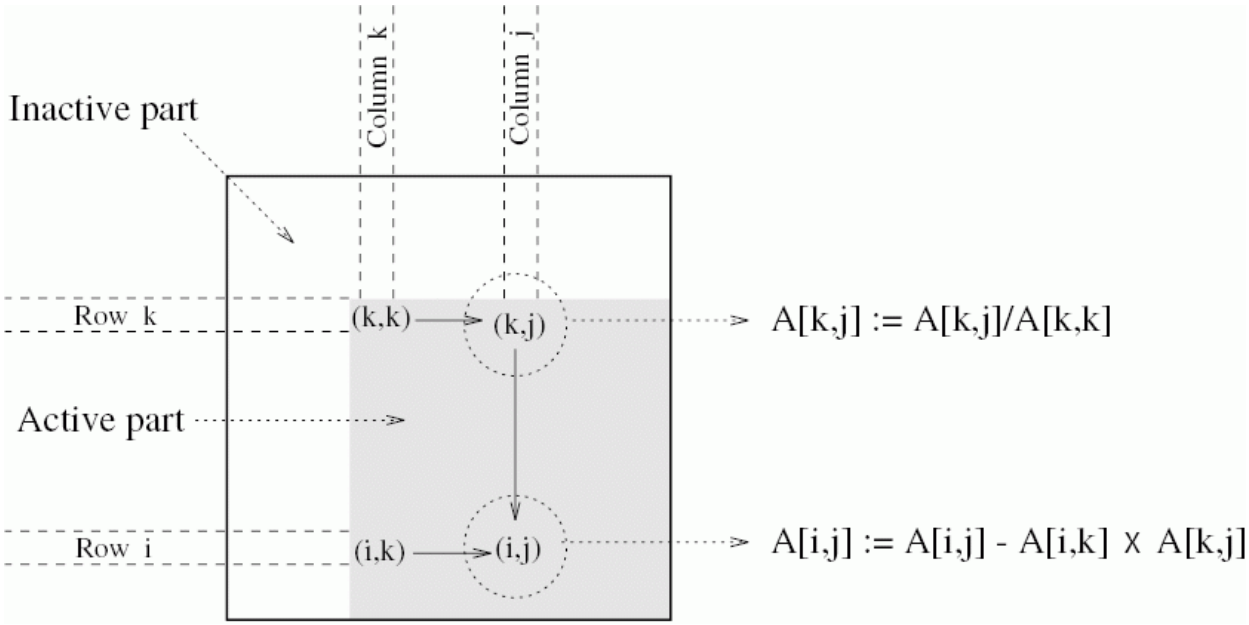

图 1.1: 高斯消去法示意图

1 问题描述

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，并且将后续的 $k+1$ 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
```

观察高斯消去算法，注意到伪代码第 4，5 行第一个内嵌循环中的 $A[k,j] := A[k,j]/A[k,k]$ 以及伪代码第 8 9 10 行双层 for 循环中的 $A[i,j] := A[i,j]-A[i,k]\times A[k,j]$ 都是可以进行向量化的循环。可以通过 SIMD 扩展指令对这两步进行并行优化。

2 Pthreads 算法设计

源码链接：https://github.com/ArcanusNEO/Parallel-Programming/tree/master/4

2.1 测试用例的确定

由于测试数据集较大，不便于各个平台同步，所以采用固定随机数种子为 12345687 的 mt19937 随机数生成器。经过实验发现不同规模下，所有元素独立生成，限制大小在 $[0, 100]$，能够生成可以被正确消元的矩阵。

代码如下：

测试数据集生成器

```
uniform_real_distribution<float> dist(0, 100);
mt19937 mt(12345687);
int n;
istringstream iss(argv[1]);
iss >> n;
cout << n << endl;
for (int i = 1; i <= n; ++i)
  for (int j = 1; j <= n; ++j) cout << dist(mt) << " \n"[j == n];
```

2.2 实验环境和相关配置

实验在华为鲲鹏 ARM 集群平台和本地 Arch Linux x86_64 平台完成；

华为鲲鹏 ARM 集群平台使用毕昇的 clang++ 编译器，本地 Arch Linux x86_64 平台使用 GNU GCC 编译器；

使用 cmake 构建项目，编译开关如下：

```
set(CMAKE_CXX_FLAGS_RELEASE "-O3")
set(THREADS_PREFER_PTHREAD_FLAG ON)
find_package(Threads REQUIRED)
```

2.3 算法设计

2.3.1 默认平凡算法

使用一维数组模拟矩阵，避免改变矩阵大小时第二维不方便调整、必须设成最大值的问题，可以减少 cache 失效；

使用 $\#define\ matrix(i,j)\ arr[(i)*n+(j)]$ 宏，增强可读性；

平凡算法

```
#define matrix(i, j) arr[(i) * n + (j)]
void func(int& ans, float arr[], int n) {
  for (int k = 0; k < n; ++k) {
    for (int j = k + 1; j < n; ++j) matrix(k, j) = matrix(k, j) / matrix(k, k);
```

```
    matrix(k, k) = 1.0;
    for (int i = k + 1; i < n; ++i) {
      for (int j = k + 1; j < n; ++j)
        matrix(i, j) = matrix(i, j) - matrix(i, k) * matrix(k, j);
      matrix(i, k) = 0;
    }
  }
#undef matrix
}
```

2.3.2 使用 Pthreads 动态创建线程并行化加速

动态创建线程 frame

```
#define matrix(i, j) arr[(i) *n + (j)]

#define MAX_SUB_THREAD 7

int     n;
float* arr;

struct thread_param_t {
  int k, t_id;
};

pthread_t       thread_handle[MAX_SUB_THREAD];
thread_param_t thread_param[MAX_SUB_THREAD];

void* thread_func(void* param) {
  auto p    = (thread_param_t*) param;
  auto k    = p->k;
  auto t_id = p->t_id;
  int  i    = k + t_id + 1;
  for (int j = k + 1; j < n; ++j)
    matrix(i, j) = matrix(i, j) - matrix(i, k) * matrix(k, j);
  matrix(i, k) = 0;
  pthread_exit(nullptr);
}

void func(int& ans, float arr[], int n) {
  ::n   = n;
  ::arr = arr;
  for (int k = 0; k < n; ++k) {
    for (int j = k + 1; j < n; ++j) matrix(k, j) = matrix(k, j) / matrix(k, k);
    matrix(k, k)     = 1.0;
    int worker_count = n - 1 - k;
    for (int offset = 0; offset < worker_count; offset += MAX_SUB_THREAD) {
      for (int t_id = 0, i = t_id + offset;
```

```
            i < worker_count && t_id < MAX_SUB_THREAD;
            ++t_id, i = t_id + offset) {
          thread_param[t_id] = {k, i};
          pthread_create(thread_handle + t_id, nullptr, thread_func,
                         thread_param + t_id);
        }
        for (int t_id = 0, i                                    = t_id + offset;
            i < worker_count && t_id < MAX_SUB_THREAD; ++t_id, i = t_id + offset)
          pthread_join(thread_handle[t_id], nullptr);
      }
    }
  }
  #undef matrix
}
```

2.3.3 使用 Pthreads 线程池和信号量同步并行化加速

线程池 + 信号量同步 + 主线程执行除法

```
#define matrix(i, j) arr[(i) *n + (j)]

#define MAX_SUB_THREAD 7

int    n;
float* arr;

struct thread_param_t {
  int t_id;
};

sem_t           sem_main;
sem_t           sem_workerstart[MAX_SUB_THREAD];
pthread_t       handle[MAX_SUB_THREAD];
thread_param_t  param[MAX_SUB_THREAD];

void* thread_func(void* param) {
  auto p    = (thread_param_t*) param;
  auto t_id = p->t_id;
  for (int k = 0; k < n; ++k) {
    sem_wait(sem_workerstart + t_id);
    for (int i = k + 1 + t_id; i < n; i += MAX_SUB_THREAD) {
      for (int j = k + 1; j < n; ++j)
        matrix(i, j) = matrix(i, j) - matrix(i, k) * matrix(k, j);
      matrix(i, k) = 0;
    }
    sem_post(&sem_main);
  }
  pthread_exit(nullptr);
```

```
  }

  void func(int& ans, float arr[], int n) {
    ::n   = n;
    ::arr = arr;
    sem_init(&sem_main, 0, 0);
    for (int i = 0; i < MAX_SUB_THREAD; ++i) sem_init(sem_workerstart + i, 0, 0);
    for (int t_id = 0; t_id < MAX_SUB_THREAD; ++t_id) {
      param[t_id].t_id = t_id;
      pthread_create(handle + t_id, nullptr, thread_func, param + t_id);
    }
    for (int k = 0; k < n; ++k) {
      for (int j = k + 1; j < n; ++j) matrix(k, j) = matrix(k, j) / matrix(k, k);
      matrix(k, k) = 1.0;
      for (int t_id = 0; t_id < MAX_SUB_THREAD; ++t_id)
        sem_post(sem_workerstart + t_id);
      for (int t_id = 0; t_id < MAX_SUB_THREAD; ++t_id) sem_wait(&sem_main);
    }
    for (int t_id = 0; t_id < MAX_SUB_THREAD; ++t_id)
      pthread_join(handle[t_id], nullptr);
    sem_destroy(&sem_main);
    for (int i = 0; i < MAX_SUB_THREAD; ++i) sem_destroy(sem_workerstart + i);
  }
  #undef matrix
}
```

线程池 + 信号量同步 + 工作线程执行除法

```
#define matrix(i, j) arr[(i) *n + (j)]

#define MAX_SUB_THREAD 7

int     n;
float* arr;

struct thread_param_t {
  int t_id;
};

sem_t           sem_leader;
sem_t           sem_div[MAX_SUB_THREAD - 1];
sem_t           sem_elim[MAX_SUB_THREAD - 1];
pthread_t       handle[MAX_SUB_THREAD];
thread_param_t  param[MAX_SUB_THREAD];

void* thread_func(void* param) {
  auto p    = (thread_param_t*) param;
  auto t_id = p->t_id;
  for (int k = 0; k < n; ++k) {
```

```
22        if (t_id == 0) {
23          for (int j = k + 1; j < n; ++j)
24            matrix(k, j) = matrix(k, j) / matrix(k, k);
25          matrix(k, k) = 1.0;
26        } else sem_wait(sem_div + t_id - 1);
27        if (t_id == 0)
28          for (int i = 0; i < MAX_SUB_THREAD - 1; ++i) sem_post(sem_div + i);
29        for (int i = k + 1 + t_id; i < n; i += MAX_SUB_THREAD) {
30          for (int j = k + 1; j < n; ++j)
31            matrix(i, j) = matrix(i, j) - matrix(i, k) * matrix(k, j);
32          matrix(i, k) = 0.0;
33        }
34        if (t_id == 0) {
35          for (int i = 0; i < MAX_SUB_THREAD - 1; ++i) sem_wait(&sem_leader);
36          for (int i = 0; i < MAX_SUB_THREAD - 1; ++i) sem_post(sem_elim + i);
37        } else {
38          sem_post(&sem_leader);
39          sem_wait(sem_elim + t_id - 1);
40        }
41      }
42      pthread_exit(nullptr);
43    }
44
45    void func(int& ans, float arr[], int n) {
46      ::n   = n;
47      ::arr = arr;
48      sem_init(&sem_leader, 0, 0);
49      for (int i = 0; i < MAX_SUB_THREAD - 1; ++i) {
50        sem_init(sem_div + i, 0, 0);
51        sem_init(sem_elim + i, 0, 0);
52      }
53      for (int t_id = 0; t_id < MAX_SUB_THREAD; ++t_id) {
54        param[t_id].t_id = t_id;
55        pthread_create(handle + t_id, nullptr, thread_func, param + t_id);
56      }
57      for (int t_id = 0; t_id < MAX_SUB_THREAD; ++t_id)
58        pthread_join(handle[t_id], nullptr);
59      sem_destroy(&sem_leader);
60      for (int i = 0; i < MAX_SUB_THREAD - 1; ++i) {
61        sem_destroy(sem_div + i);
62        sem_destroy(sem_elim + i);
63      }
64    }
65    #undef matrix
```

2.3.4 使用 Pthreads 线程池和 barrier 栅栏同步并行化加速

线程池 + 栅栏同步 + 工作线程执行除法

```
#define matrix(i, j) arr[(i) *n + (j)]

#define MAX_SUB_THREAD 7

int    n;
float* arr;

struct thread_param_t {
  int t_id;
};

pthread_barrier_t barrier_div;
pthread_barrier_t barrier_elim;
pthread_t         handle[MAX_SUB_THREAD];
thread_param_t    param[MAX_SUB_THREAD];

void* thread_func(void* param) {
  auto p    = (thread_param_t*) param;
  auto t_id = p->t_id;
  for (int k = 0; k < n; ++k) {
    if (t_id == 0) {
      for (int j = k + 1; j < n; ++j)
        matrix(k, j) = matrix(k, j) / matrix(k, k);
      matrix(k, k) = 1.0;
    }

    pthread_barrier_wait(&barrier_div);

    for (int i = k + 1 + t_id; i < n; i += MAX_SUB_THREAD) {
      for (int j = k + 1; j < n; ++j)
        matrix(i, j) = matrix(i, j) - matrix(i, k) * matrix(k, j);
      matrix(i, k) = 0.0;
    }

    pthread_barrier_wait(&barrier_elim);
  }
  pthread_exit(nullptr);
}

void func(int& ans, float arr[], int n) {
  ::n   = n;
  ::arr = arr;
  pthread_barrier_init(&barrier_div, nullptr, MAX_SUB_THREAD);
  pthread_barrier_init(&barrier_elim, nullptr, MAX_SUB_THREAD);

  for (int t_id = 0; t_id < MAX_SUB_THREAD; ++t_id) {
    param[t_id].t_id = t_id;
    pthread_create(handle + t_id, nullptr, thread_func, param + t_id);
```

```
49      }
50      for (int t_id = 0; t_id < MAX_SUB_THREAD; ++t_id)
51        pthread_join(handle[t_id], nullptr);
52
53      pthread_barrier_destroy(&barrier_div);
54      pthread_barrier_destroy(&barrier_elim);
55    }
56    #undef matrix
57  }
```

3 实验及结果分析