Московский авиационный институт
Национальный исследовательский университет

Операционные системы
Кафедра 806

Лабораторная работа №3

Студент: Минибаев Айдар
Группа: М8О-301Б-18
Преподаватель: Миронов Е.С.
Дата: .
Подпись: .

Москва 2020

1. Постановка задачи

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы. При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемых программой. При возможности необходимо использовать максимальное количество возможных потоков.

Рассчитать детерминант матрицы

2. Метод решения

Проект состоит из 2-х исполняемых файлов: генератор квадратных матриц и сам рассчитыватель детерминанта.
Запуск генератора матриц matrix-generator происходит со следующими ключами: размерность матрицы и коэффициент (вещественное число), который будет влиять на числа в получившийся матрице.
Числа в матрице рассчитываются по следующему алгоритму:

```
double float_num = rand()%(int)(1000*avg);
float_num /= 1000;
float_num = round(float_num);
```

где avg - коэффициент, вводимый при запуске программы.
Запуск программы по расчету определителя матрицы происходит с ключами: название файла с матрицей и количество потоков.

1. Считывание матрицы из файла в структуру

```
struct nMatrix {
   int **matrix;
   int n;
};
```

2. Если размерность матрицы 1 или 2, то ответ можно вернуть сразу.
3. Разбиваем матрицу на части по строкам: если всего потоков t, а размерность матрицы n, то размер окна получается floor(n / t).
4. На каждом таком сегменте запускаем поточную функцию, которая в цикле обходит строки полученного сегмента.
5. Вычисление определителя происходит по первому столбцу.

3. Программная реализация

matrix-generator.c

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

int main(int argc, char** argv) {
   if(argc < 3) {
       fprintf(stderr, "Usage: %s <size of NxN matrix> <matrix range modifier> [srand value]\n",
```

```
argv[0]);
        exit(0);
    }

    int n = atoi(argv[1]);
    int i, j;
    double avg = atof(argv[2]);

    if(argc == 4) {
        srand(atoi(argv[3]));
    } else {
        srand(time(NULL));
    }

    fprintf(stdout, "%d\n", n);
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            double float_num = rand()%(int)(1000*avg);
            float_num /= 1000;
            float_num = round(float_num);
            fprintf(stdout, "%d ", (int)float_num);
        }
        fprintf(stdout, "\n");
    }
}
```

main.c

```
#include <stdlib.h>
#include <stdio.h>
#include "determinants.h"

struct nMatrix* loadmatrix(char *filename);
int num_of_threads = 1;

int main(int argc, char **argv) {
    if(argc != 3) {
        fprintf(stderr, "Usage: %s <matrix file> <num of threads>\n", argv[0]);
        exit(0);
    }
    struct nMatrix* m = loadmatrix(argv[1]);
    num_of_threads = atoi(argv[2]);
    printmatrix(m);
    printf("%ld\n", determinant_threaded_setup(m));
    freematrix(m);
}

struct nMatrix* loadmatrix(char *filename) {

    FILE *matrixfile = fopen(filename, "r");

    if(matrixfile == NULL) {
        perror("Couldn't open file\n");
        exit(-1);
    }
```

```
    int i, j, k, n;

    if(fscanf(matrixfile, " %d", &(n)) == 0) {
        fprintf(stderr, "Empty file :/\n");
        exit(-1);
    }

    struct nMatrix *m = newmatrix(n);
    m->n = n;

    fprintf(stdout, "Preparing to load a %dx%d matrix...\n", n, n);

    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            if(fscanf(matrixfile, " %d", &(m->matrix[i][j])) == 0) {
                fprintf(stderr, "Not enough lines. Should be %dx%d", n, n);
                freematrix(m);
                exit(-1);
            }
        }
    }

    fclose(matrixfile);

    return m;

}
```

determinants.h

```
#ifndef THREADS_SPECIFIED
#define THREADS 1
#endif

struct nMatrix {
    int **matrix;
    int n;
};

long determinant_threaded_setup(struct nMatrix* m);
long determinant_nonthreaded(struct nMatrix *m);
struct nMatrix* submatrix_copy(struct nMatrix *m, int i, int j);
struct nMatrix* newmatrix(int n);

void freematrix(struct nMatrix *m);

void printmatrix(struct nMatrix* m);
```

determinants.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <math.h>
```

```c
#include "determinants.h"

extern int num_of_threads;

struct matrixthread {
   struct nMatrix *matrix;
   int start;
   int end;
};

long getcofactor(int i, int j, int a, struct nMatrix *m);
void *determinant_threaded(void *arg);


long determinant_threaded_setup(struct nMatrix* m) {
   int i;
   int start, end = m->n;
   int partition = (int)floor(m->n/(double)num_of_threads);
   pthread_t *pth = malloc(num_of_threads * sizeof(pthread_t));
   long joined_determinant = 0;

   fprintf(stderr, "partition size: %d\n", partition);

   if(m->n == 1) { return m->matrix[0][0]; }

   if(m->n == 2) {
       return m->matrix[0][0] * m->matrix[1][1] - m->matrix[0][1] * m->matrix[1][0];
   }

   for(i = num_of_threads-1; i >= 0; i--) {
       start = i*partition;

       struct matrixthread *t = (struct matrixthread*)malloc(sizeof(struct matrixthread));
       t->matrix = m;
       t->start = start;
       t->end = end;

       fprintf(stderr, "thread #%d calculating from %d -> %d\n", i, start, end);
       pthread_create(&pth[i], NULL, determinant_threaded, (void*)t);

       end = start;

   }



   for(i = 0; i < num_of_threads; i++) {
       void *returnValue;
       pthread_join(pth[i], &returnValue);
       fprintf(stderr, "Thread %d returned determinant %ld\n", i, *(long*)returnValue);
       joined_determinant += *(long*)returnValue;
       free(returnValue);
   }

   free(pth);
```

```
    return joined_determinant;
}

void *determinant_threaded(void *arg) {
    struct matrixthread *t = (struct matrixthread*)arg;
    long *det = malloc(sizeof(long));
    *det = 0;
    int i;

    for(i = t->start; i < t->end; i++) {
        long temp = getcofactor(i, 0, t->matrix->matrix[i][0], t->matrix);
        *det += temp;
        fprintf(stderr, "Sub determinant[%d][0] (%d) = %ld\n", i, t->matrix->matrix[i][0], temp);
    }
    free(t);
    pthread_exit((void*)det);


}

long determinant_nonthreaded(struct nMatrix *m) {
    int i;
    int d = 0;
    int c = -1;

    if(m->n == 2) {
        return m->matrix[0][0]*m->matrix[1][1] -m->matrix[0][1]*m->matrix[1][0];
    }

    for(i = 0; i < m->n; i++) {
        d += getcofactor(i, 0, m->matrix[i][0], m);
    }

    return d;
}

long getcofactor(int i, int j, int a, struct nMatrix *m) {
    if(a == 0) return 0;
    struct nMatrix *u = submatrix_copy(m, i, 0);
    int c;
    if((i+j)%2 == 0) {
        c = 1;
    } else {
        c = -1;
    }
    long det = c * a * determinant_nonthreaded(u);
    freematrix(u);
    return det;
}

struct nMatrix* newmatrix(int n) {
    int i;

    struct nMatrix* m = (struct nMatrix*)malloc(sizeof(struct nMatrix));
    m->matrix = (int**)malloc(n * sizeof(int*));
    for(i = 0; i < n; i++) {
```

```
        m->matrix[i] = (int*)malloc(n * sizeof(int));
    }
    m->n = n;

    return m;
}


void freematrix(struct nMatrix* m) {
    int i;

    for(i = 0; i < m->n; i++) {
        free(m->matrix[i]);
    }

    free(m->matrix);
    free(m);
}

struct nMatrix* submatrix_copy(struct nMatrix *m, int i, int j) {
    struct nMatrix *u = newmatrix(m->n-1);
    int a, b, x, y;

    for(a = 0, x = 0; a < u->n; a++, x++) {
        for(b = 0, y = 0; b < u->n; b++, y++) {
            if(x == i) x++;
            if(y == j) y++;
            u->matrix[a][b] = m->matrix[x][y];
        }
    }

    return u;
}


void printmatrix(struct nMatrix* m) {
    int i, j;

    for(i = 0; i < m->n; i++) {
        for(j = 0; j < m->n; j++) {
            printf("%4d ", m->matrix[i][j]);
        }
        printf("\n");
    }
}
```

4. Тестирование

```
$ make
gcc main.c determinants.c -o determinant -pthread -lm -D DEBUG -D ZERO
gcc matrix-generator.c -o matrix-generator -lm -D DEBUG -D ZERO
$ ./matrix-generator 8 42 > matrix.txt
$ cat matrix.txt
8
5 7 25 19 13 30 14 1
```

```
4 29 14 36 32 9 38 3
21 28 16 16 19 21 17 41
7 1 15 17 38 12 17 1
18 18 20 8 6 11 28 10
40 18 22 30 27 36 9 7
22 24 22 18 22 16 17 5
35 31 22 32 19 15 9 38
$ ./determinant matrix.txt 4
Preparing to load a 8x8 matrix...
  5    7   25   19   13   30   14    1
  4   29   14   36   32    9   38    3
 21   28   16   16   19   21   17   41
  7    1   15   17   38   12   17    1
 18   18   20    8    6   11   28   10
 40   18   22   30   27   36    9    7
 22   24   22   18   22   16   17    5
 35   31   22   32   19   15    9   38
partition size: 2
thread #3 calculating from 6 -> 8
thread #2 calculating from 4 -> 6
thread #1 calculating from 2 -> 4
thread #0 calculating from 0 -> 2
Sub determinant[2][0] (21) = -22813946148
Sub determinant[4][0] (18) = 32468269320
Sub determinant[6][0] (22) = -12377547424
Sub determinant[0][0] (5) = -9650195960
Sub determinant[3][0] (7) = -8876922936
Sub determinant[5][0] (40) = 82608993920
Sub determinant[7][0] (35) = -43864431940
Sub determinant[1][0] (4) = 7403947760
Thread 0 returned determinant -2246248200
Thread 1 returned determinant -31690869084
Thread 2 returned determinant 115077263240
Thread 3 returned determinant -56241979364
24898166592



$ ./matrix-generator 2 42 > matrix.txt
$ ./determinant matrix.txt 4
Preparing to load a 2x2 matrix...
  5   10
 33   36
partition size: 0
-150



$ ./matrix-generator 1 42 > matrix.txt
$ ./determinant matrix.txt 1
Preparing to load a 1x1 matrix...
 42
partition size: 1
42
```

```
$ ./matrix-generator 3 42 > matrix.txt
$ ./determinant matrix.txt 8
Preparing to load a 3x3 matrix...
  4   17    8
 26    8   14
 37   34    8
partition size: 0
thread #7 calculating from 0 -> 3
thread #6 calculating from 0 -> 0
thread #5 calculating from 0 -> 0
thread #4 calculating from 0 -> 0
thread #3 calculating from 0 -> 0
thread #2 calculating from 0 -> 0
thread #1 calculating from 0 -> 0
thread #0 calculating from 0 -> 0
Sub determinant[0][0] (4) = -1648
Sub determinant[1][0] (26) = 3536
Sub determinant[2][0] (37) = 6438
Thread 0 returned determinant 0
Thread 1 returned determinant 0
Thread 2 returned determinant 0
Thread 3 returned determinant 0
Thread 4 returned determinant 0
Thread 5 returned determinant 0
Thread 6 returned determinant 0
Thread 7 returned determinant 8326
8326
```

Перейдя по ссылке: https://matrixcalc.org/#determinant%28%7B%7B4,17,8%7D,%7B26,8,14%7D,%7B37,34,8%7D%7D%29 , можно убедиться, что определитель действительно равен 8326.

5. Выводы

Программные потоки очень удобно использовать для многозадачности и для большей скорости работы некоторых алгоритмов. В отличии от процессов они быстрее создаются и контекст выполнения у потоков переключается быстрее чем у процессов. Большое отличие потоков от процессов состоит в том, что потоки делят между собой одно адресное пространство. В данной лабораторной работе была продемонстрирована обработка матрицы по определенному алгоритму в многопоточном режиме. Обычно параллельные алгоритмы хорошо обрабатывать в многопоточном режиме на графических процессорах, так как на них очень много ядер.