Εργαστήριο Παράλληλων Συστημάτων - Εργασία 1

Χρήστος Μαργιώλης

Δεκέμβριος 2022

Περιεχόμενα

1 Κώδικας

Ο κώδικας έχει σχόλια μόνο στα σημεία που θεώρησα ότι μπορεί να προκύψει κάποιο «μπέρδεμα».

```
#include <err.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

#include <omp.h>

#define abs(x)  ((x) < 0 ? -(x) : (x))

static void     *emalloc(size_t);
static int      safe_input(const char *, ...);
static void     pretty_print(int **, int, const char *);
static int      strictly_diagonal_dominant(int **, int);
static int      diagonal_max(int **, int);
static int      **new_array(int **, int, int);
static int      calc_min(int **, int);

/*
 * Fail-safe malloc(3).
 */
static void *
emalloc(size_t nb)
{
        void *p;

        if ((p = malloc(nb)) == NULL)
                err(1, "malloc");

        return (p);
}

static int
safe_input(const char *fmt, ...)
{
        va_list args;
        char buf[48];
        int n, rc;

        /* Collect the arguments into a buffer. */
        va_start(args, fmt);
        vsnprintf(buf, sizeof(buf), fmt, args);
        va_end(args);
```

```
        /*
         * The following loop keeps asking for input as long as the current
         * input wasn't correct. In this case "incorrect" input means anything
         * other than digits.
         */
        do {
                printf("\r%s", buf);
                rc = scanf("%d", &n);
                (void)getchar();
        } while (rc != 1);

        return (n);
}

/*
 * Print the contents of a 2D array like:
 *
 * array = [
 *      [x, y, z]
 *      [x, y, z]
 * ]
 */
static void
pretty_print(int **arr, int n, const char *name)
{
        int i, j;

        printf("\n%s = [\n", name);
        for (i = 0; i < n; i++) {
                printf("\t[");
                for (j = 0; j < n; j++) {
                        printf("%d%s", arr[i][j],
                           (j == n - 1) ? "]\n" : ", ");
                }
        }
        printf("]\n");
}

static int
strictly_diagonal_dominant(int **a, int n)
{
        int i, j, sum, flag = 1;

#pragma omp parallel for
        for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++) {
```

```
                        if (i == j)
                                continue;
                        sum += abs(a[i][j]);
                }
                if (abs(a[i][i]) <= sum)
                        flag = 0;
        }

        return (flag);
}

static int
diagonal_max(int **a, int n)
{
        int i, max;

        max = a[0][0];
#pragma omp parallel for reduction(max : max)
        for (i = 0; i < n; i++) {
                if (abs(a[i][i]) > max)
                        max = a[i][i];
        }

        return (max);
}

static int **
new_array(int **a, int m, int n)
{
        int **b, i, j;

        b = emalloc(n * sizeof(int *));
#pragma omp parallel for
        for (i = 0; i < n; i++) {
                b[i] = emalloc(n * sizeof(int));
                for (j = 0; j < n; j++)
                        b[i][j] = m - a[i][j];
        }

        return (b);
}

static int
calc_min(int **b, int n)
{
        int i, j, min;
```

```
        /* with reduction */
        min = b[0][0];
#pragma omp parallel for reduction(min : min)
        for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++) {
                        if (b[i][j] < min)
                                min = b[i][j];
                }
        }

        /* without reduction, with critical region protection */
        min = b[0][0];
#pragma omp parallel for private(i, j) shared(min)
        for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++) {
#pragma omp critical
                {
                        if (b[i][j] < min)
                                min = b[i][j];
                }
                }
        }

        /* XXX: didn't implement binary tree method */

        return (min);
}

int
main(int argc, char *argv[])
{
        int **a, **b;
        int i, j, m, min, n, ntd;
        double start, end;

        ntd = safe_input("threads: ", 0);
        omp_set_num_threads(ntd);

        n = safe_input("n: ", 0);
        a = emalloc(n * sizeof(int *));
        for (i = 0; i < n; i++) {
                a[i] = emalloc(n * sizeof(int));
                for (j = 0; j < n; j++)
                        a[i][j] = safe_input("a[%d][%d]: ", i, j);
        }
```

```
        start = omp_get_wtime();

        if (strictly_diagonal_dominant(a, n)) {
                m = diagonal_max(a, n);
                b = new_array(a, m, n);
                min = calc_min(b, n);

                pretty_print(a, n, "A");
                pretty_print(b, n, "B");
                printf("Diagonal max: %d\n", m);
                printf("Min: %d\n", min);

                free(b);
        } else
                printf("not strictly diagonal dominant\n");

        end = omp_get_wtime();
        printf("Total time: %f seconds\n", end - start);

        free(a);

        return (0);
}
```

2 Προβλήματα

Δεν κατάφερα να υλοποιήσω τον υπολογισμό του ελάχιστο με χρήση αλγορίθμου δυαδικού δέντρου (ερώτημα d2.2).

3 Βοηθητικό script

```
#!/bin/sh

usage()
{
        echo "usage: ${0##*/} nthreads N" 1>&2
        exit 1
}

test ${#} -ne 2 && usage
echo ${1}
echo ${2}
strings -n 1 < /dev/random | grep -o '[[:digit:]]' | head -n$((${2} * ${2}))
```

Το script `randinput` (δεν τρέχει σε Windows) δέχεται ως είσοδο τον αριθμό των νημάτων, καθώς και το N. Στην συνέχεια τυπώνει (με την συγκεκριμένη σειρά) τον αριθμό των νημάτων, το N, και N τυχαίους αριθμούς. Αυτό είναι χρήσιμο στο να μπορούμε να κάνουμε δοκιμές με διαφορετικά δεδομένα (ειδικά για μεγάλες τιμές N) χωρίς να πρέπει να δώσουμε τα δεδομένα χειροκίνητα.

Η χρήση του script έχει ως εξής:

```
usage: randinput nthreads N
```

Παρακάτω φαίνεται μία ενδεικτική χρήση:

```
$ ./randinput 2 3
2
3
8
0
1
5
1
2
0
4
8
```

'Αρα έχουμε 2 νήματα, έναν πίνακα $3x3$, τα στοιχεία του οποίου είναι:

$$[[8, 0, 1], [5, 1, 2], [0, 4, 8]]$$

Τώρα, μπορούμε να διοχετεύσουμε την έξοδο του script σε ένα αρχείο και να το δώσουμε ως είσοδο στο κύριο πρόγραμμα:

```
./randinput 2 3 > input.txt
```

4 Ενδεικτικά τρεξίματα

Σημειώνεται ότι το πρόγραμμα τρέχει κανονικά και χωρίς αρχείο για είσοδο.

Για 2 threads και $A = 4x4$:

```
christos@pleb$ make
cc ex1.c -fopenmp -lomp -o ex1
christos@pleb$ ./randinput 2 4 > input.txt
christos@pleb$ ./ex1 < input.txt
a[3][3]:
A = [
        [4, 6, 4, 2]
        [4, 1, 4, 7]
        [6, 2, 8, 6]
        [9, 0, 8, 1]
]

B = [
        [4, 2, 4, 6]
        [4, 7, 4, 1]
        [2, 6, 0, 2]
        [-1, 8, 0, 7]
]
Diagonal max: 8
Min: -1
Total time: 0.000249 seconds
```

Για 4 threads και $A = 10x10$:

```
christos@pleb$ make
cc ex1.c -fopenmp -lomp -o ex1
christos@pleb$ ./randinput 4 10 > input.txt
christos@pleb$ ./ex1 < input.txt
a[9][9]:
A = [
        [0, 7, 5, 1, 0, 2, 5, 8, 6, 6]
        [1, 7, 9, 1, 3, 5, 8, 4, 1, 3]
        [6, 4, 5, 3, 4, 1, 9, 8, 0, 8]
        [3, 6, 6, 9, 5, 1, 7, 6, 0, 6]
        [1, 3, 8, 8, 9, 6, 4, 3, 0, 0]
        [1, 4, 1, 4, 7, 5, 5, 5, 4, 3]
        [7, 6, 5, 5, 5, 8, 5, 2, 1, 4]
        [8, 5, 3, 9, 5, 3, 0, 5, 7, 2]
        [2, 9, 3, 6, 4, 5, 6, 0, 2, 9]
        [4, 7, 2, 4, 5, 3, 8, 4, 2, 1]
]

B = [
        [9, 2, 4, 8, 9, 7, 4, 1, 3, 3]
        [8, 2, 0, 8, 6, 4, 1, 5, 8, 6]
        [3, 5, 4, 6, 5, 8, 0, 1, 9, 1]
        [6, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        [8, 6, 1, 1, 0, 3, 5, 6, 9, 9]
        [8, 5, 8, 5, 2, 4, 4, 4, 5, 6]
        [2, 3, 4, 4, 4, 1, 4, 7, 8, 5]
        [1, 4, 6, 0, 4, 6, 9, 4, 2, 7]
        [7, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        [5, 2, 7, 5, 4, 6, 1, 5, 7, 8]
]
Diagonal max: 9
Min: 0
Total time: 0.000746 seconds
```

Για 3 threads και $A = 100x100$. Παρατηρούμε ότι το συγκεκριμένο τρέξιμο είχε δεδομένα τέτοια ώστε να μην πληρείται η προϋπόθεση του να είναι ο πίνακας αυστηρά διαγώνια δεσπόζων:

```
christos@pleb$ make
cc ex1.c -fopenmp -lomp -o ex1
christos@pleb$ ./randinput 3 100 > input.txt
christos@pleb$ ./ex1 < input.txt
a[99][99]: not strictly diagonal dominant
Total time: 0.000313 seconds
```

Στα παρακάτω τρεξίματα ο πίνακας θα είναι $1000x1000$, θα έχει τα ίδια στοιχεία, αλλά κάθε τρέξιμο θα έχει διαφορετικό αριθμό threads, ωστέ να παρατηρήσουμε τις διαφορές στην ταχύτητα εκτέλεσης του προγράμματος ανάλογα με τον αριθμό των threads.

Παρατηρούμε ότι όταν τα threads είναι περισσότερα από 1, η ταχύτητα εκτέλεσης σχεδόν υποδιπλασιάζεται.

Για 1 thread:

```
Total time: 0.385317 seconds
```

Για 2 threads:

```
Total time: 0.634895 seconds
```

Για 3 threads:

```
Total time: 0.666589 seconds
```

Για 4 threads:

```
Total time: 0.696766 seconds
```