# **מכללת אורט בראודה המחלקה להנדסת תוכנה** **16.2.2017**

## מבחן סופי בקורס תכנות מתקדם 61617

**מועד א'**

שלוש שאלות 100% ועוד שאלת בונוס 10%.

שלוש שאלות תכנות ב-C, שאלה רביעית תאורטית.

## שאלה מספר 1 (33 נקודות)

בעזרת **סגמנטי זיכרון משותפים וסמפורים**, עליך לממש מעין pipeline.

החישוב שנעשה הוא מכפלה פנימית של וקטורים ∑aibi: הרעיון הוא שזה יעשה ע"י 2 תהליכים: תהליך אחד יבצע את הכפלים, ישמור את תוצאות הכפלים בשטח משותף, והתהליך השני יסכם את תוצאות הכפל וישמור את הסכום בשטח המשותף.

ההנחה היא שהשטח המשותף אינו מספיק גדול להכיל את כל הכפלים. לכן יש צורך בסנכרון ע"י סמפורים על מנת להביא בחשבון 2 תנאי תחרות:

* התהליך המסכם עשוי להקדים את המכפיל.
* התהליך המכפיל עשוי לגלות שאין מקום פנוי בשטח המשותף ולכן הוא ממלא את השטח המשותף באופן מעגלי, וכאשר אין מקום הוא ממתין על סמפור עד שיתפנה מקום בשטח המשותף. אקבל גם פתרונות של מילוי ושחרור בבת אחת.
* יש לשחרר את המשאבים המשותפים כאשר אין בהם צורך יותר.
* תהליך האב הוא זה שמשתמש בתוצאה. תהליך האב יכול להיות אחד משני התהליכים החישוביים.

עליך לממש רוטינת C שהכרזתה:

double pipeline\_inner(double darr1[], double darr2[], int n)

המקבל 2 מערכים ממשיים darr1 ו-darr2 וגודלם n ומחזירה את סכומם לפי האלגוריתם הנ"ל. הנח כי בשטח המשותף לי ניתן לאכסן יותר מ-8 מספרים ממשיים (בנוסף לעוד 3-4 שדות) וכי n עשוי להיות גדול יותר.

לדוגמא, הפלט של התוכנית הבאה:

void main()

{

double darr1[N] = {1.0, 2.0, 3.0, 4.0,

5.0, 6.0, 7.0, 8.0,

9.0, 10.0, 11.0, 12.0,

13.0, 14.0, 15.0, 16.0,

17.0, 18.0, 19.0, 20.0};

double darr2[N] = {11.0, 22.0, 33.0, 44.0,

55.0, 66.0, 77.0, 88.0,

99.0, 110.0, 111.0, 122.0,

133.0, 144.0, 155.0, 166.0,

177.0, 188.0, 199.0, 202.0};

double inner\_prod;

int i, j;

inner\_prod = pipeline\_inner(darr1, darr2, N);

printf("darr1:\n");

for(i=0; i<N; i++)

printf(" %lf ", darr1[i]);

printf("\n");

printf("darr2:\n");

for(i=0; i<N; i++)

printf(" %lf ", darr2[i]);

printf("\n");

printf("inner\_prod: %lf\n", inner\_prod);

} /\* main \*/

יהיה:

darr1:

1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000 10.000000 11.000000 12.000000 13.000000 14.000000 15.000000 16.000000 17.000000 18.000000 19.000000 20.000000

darr2:

11.000000 22.000000 33.000000 44.000000 55.000000 66.000000 77.000000 88.000000 99.000000 110.000000 111.000000 122.000000 133.000000 144.000000 155.000000 166.000000 177.000000 188.000000 199.000000 202.000000

**inner\_prod: 29860.000000**

המהלך העניינים עשוי להיות כזה:

התהליך המסכם ייעצר על שטח משותף ריק,

התהליך הראשון יכניס לשטח המשותף את הערכים:

11.0, 44.0, 99.0, 176.0, 275, 396.0, 538.0, 704.0

התהליך המכפיל ישחרר את התהליך המסכם וייעצר כי אין לו איפה לשמור מכפלות,

התהליך המסכם יסכם את המכפלות וישחרר את המכפיל וחוזר חלילה.

**שאלה מספר 2** (34 נקודות)

ב-unix ישנו תוכנית בשם diff אשר בהינתן 2 קבצים כמעט זהים, המסוגל להדפיס רק את ההבדלים ביניהם.

לדוגמא, בהינתן 2 הקבצים הבאים:

// print\_numbs1.c

#include <stdio.h>

int main()

{

int i;

for(i=0; i < 10; i++)

printf("%d\n", i);

}// main

// print\_numbs2.c

#include <stdio.h>

int main()

{

long int i;

for(i=0; i < 10; i++)

printf("%ld\n", i);

}// main

הפעלת הפקודה

diff print\_numbs1.c print\_numbs2.c

ייתן למסך פלט:

1c1

< // print\_numbs1.c

---

> // print\_numbs2.c

7c7

< int i;

---

> long int i;

10c10

< printf("%d\n", i);

---

> printf("%ld\n", i);

הסימן 7c7 מציין ששורה 7 בקובץ הראשון כמעט זהה לשורה 7 בקובץ השני ולאחר מכן מודפסים 2 השורות הכמעט זהות עם "<" לפני השורה של הקובץ הראשון ו-"<" לפני השורה בקובץ השני, עם שורה של "---" ביניהם.

בשאלה זו עליך לכתוב תוכנית אשר (כמו diff) מקבלת שמות של 2 קבצים, מריצה את diff עליהם, מנתבת את הפלט של diff ל-pipe, ובעזרתו מדפיסה למסך את תוכן הקובץ הראשון עם השורות הכמעט זהות של הקובץ השני כהערות.

לדוגמא הפלט של הפקודה:

./a.out print\_numbs1.c print\_numbs2.c

יהיה:

// print\_numbs1.c

//// print\_numbs2.c

#include <stdio.h>

int main()

{

int i;

// long int i;

for(i=0; i < 10; i++)

printf("%d\n", i);

// printf("%ld\n", i);

}// main

**שאלה מספר 3 (33 נקודות)**

ב-unix קיים סיגנל בשם SIGALRM שאותו תהליך יכול להזמין שישלך לעצמו באינטרוולים קבועים ביו השאר ע"י קריאת המערכת

int ualarm(int milisecs1, int milisecs2);

הדבר גורם לכך שמערכת ההפעלה תשלח SIGALRM כל milisecs2 אלפיות שניה כאשר הראשונה תתרחש בעוד milisecs1 אלפיות שניה.

הקריאה

ualarm(0,0);

מבטלת את ההזמנה.

השתמש ב-SIGALRM ובעוד סיגנל על מנת לממש מעין מנגנון דיבוג:

התוכניתן יוכל להוסיף משתנים למאגר watch שיוגבל נניח ל-20 משתנים פעילים.

התוכנית תבדוק כל מאית שניה אם עורכם של המשתנים ב-watch השתנה ותדפיס הודעה מתאימה למסך. כמו כן לחיצת Ctrl-C תגרום להדפסת הערכים הנוכחיים של כל משתני ה-watch. למען הפשטות, המשתנים שלהם ימומש watch יהיו כולם int.

המנגנון יתמוך ברוטינות הבאות:

void watch\_init()

המאתחל את המנגנון.

void watch\_end()

המבטל את המנגנון.

void var\_watch(int \*ptr, char var\_name[]);

המוסיפה משתנה שלם המוצבע ע"י המשתנה ptr למאגר. לצורך זיהוי, ערכו יודפס עם המחרוזת var\_name.

לדוגמא,

פלט אפשרי של התוכנית הבאה:

int main()

{

int i, j, k, m;

watch\_init();

var\_watch(&i, "i");

var\_watch(&j, "j");

i = j = 0;

for(m=0; m < 1024; m++)

{

for(; (i % 4) != 0; i++)

for(k=0; k < 1000\*1024\*1024; k++)

;

i++;

for(; (j % 7) != 0; j++)

for(k=0; k < 1000\*1024\*1024; k++)

;

j++;

} // for

watch\_end();

} // main

יהיה:

Varible i changed from 32767 to 1

Varible j changed from 613591200 to 1

Varible i changed from 1 to 2

Varible i changed from 2 to 3

Varible i changed from 3 to 5

Varible j changed from 1 to 2

Varible j changed from 2 to 3

Varible j changed from 3 to 4

Varible j changed from 4 to 5

Varible j changed from 5 to 6

Varible j changed from 6 to 8

Varible i changed from 5 to 6

Varible i changed from 6 to 7

Varible i changed from 7 to 9

Varible j changed from 8 to 9

Varible j changed from 9 to 10

Varible j changed from 10 to 11

Varible j changed from 11 to 12

Varible j changed from 12 to 13

Varible j changed from 13 to 15

Varible i changed from 9 to 10

^CValue of variable i == 10

Value of variable j == 15

Varible i changed from 10 to 11

Varible i changed from 11 to 13

^CValue of variable i == 13

Value of variable j == 15

Varible j changed from 15 to 16

Varible j changed from 16 to 17

Varible j changed from 17 to 18

^\Quit

**שאלה 4 (בונוס 10 נק')**

עליך לממש רוטינת אסמבלי של הלינוקס הניתנת לקריאה מתוך תוכנית C המבצעת כפל פנימי של מספרים ממשיים float ע"י SIMD בשיטה הבאה:

* ההנחה היא שמספר האיברים מתחלקת ב-8.
* הסכום של כל שמינייה במערך תעובד כך:

1. 4 האיברים הראשונים בכל מערך יוכפלו בבת אחת.
2. 4 האיברים השניים בכל מערך יוכפלו בבת אחת.
3. 8 המספרים בשני המכפלות יסוכמו בבת אחת.
4. 4 המספרים האחרונים יסוכמו סדרתית.

לדוגמא,

אם

Arr1[] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};

Arr2[]= {11.0, 22.0, 33.0, 44.0, 55.0, 66.0, 77.0, 88.0};

אזי בשלב ראשון יחושבו:

{1.0\*11.0, 2.0\*22.0, 3.0\*33.0, 4\*44.0}

{5.0\*55.0, 6.0\*66.0, 7.0\*77.0, 8\*88.0}

לאחר מכן

{1.0\*11.0+5.0\*55.0, 2.0\*22.0+6.0\*66.0, 3.0\*33.0+7.0\*77.0, 4\*44.0+8.0\*88.0}

ולבסוף

1.0\*11.0+5.0\*55.0+ 2.0\*22.0+6.0\*66.0+ 3.0\*33.0+7.0\*77.0+ 4\*44.0+8.0\*88.0

ההכרזה ב-C על הרוטינה שלך תהיה:

extern float inner\_product(float arr1[], float arr2[], int n);

לדוגמא, פלט של התוכנית הבאה:

int main()

{

int i;

float sum;

float arr1[8] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};

float arr2[8] = {80.0, 70.0, 60.0, 50.0, 40.0, 30.0, 20.0, 10.0};

int n = 8;

sum = inner\_product(arr1, arr2,n);

printf("arr1:\n");

for(i=0; i < n; i++)

printf("%10.2f", arr1[i]);

printf("arr2:\n");

for(i=0; i < n; i++)

printf("%10.2f", arr2[i]);

printf("sum = %f\n", sum);

} // main()

יהיה:

arr1:

1.00 2.00 3.00 4.00 5.00 6.00 7.00 8.00arr2:

80.00 70.00 60.00 50.00 40.00 30.00 20.00 10.00

sum = 1200.000000