Exercise 5

תכנות פונקציונלי במערכות מבוזרות ומקבילות - 381.1.0112

מגיש: עומר לוכסמבורג

פונקציות ring:

את פונקציית ring_parallel בנינו באופן סידרתי. כל תהליך נוצר ולמעשה היווה את החוליה הקיימת במעגל לפי מספרו. לאחר מכן, כל תהליך יצר את התהליך הבא אחריו על התהליך ה-N.

רק לאחר כל הבניה, שלחנו M הודעות מ-P1, כך שההודעה הבאה תשלח רק לאחר שקיבלנו את ההודעה הקודמת.

מאחר וכל התהליך הוא סידרתי, נצפה לקבל תוצאה טובה יותר עבור ring_serial, שבנוי בצורה דומה, (אך מאחר ויש רק תהליך אחד אז שלב הבניה אינו נחוץ). הסיבה לכך היא שתהליכים מקביליים שמבצעים פעולה סידרתית יחד, "יבזבזו" זמן על הקצאות משאבים ובקשת זמן מעבד.

להלך תוצאות לדוגמה (הזמנים במילי-שניות – מאחר ונדרשת תוצאה בשלמים ייתכן ויהיה 0):

	N=10, M=100	N=100, M=100	N=1000, M=100	N=10, M=1000	N=100, M=1000	N=1000, M=1000
ring_parallel	2	14	144	12	125	1320
ring_serial	0	6	28	3	28	293

ניתן לראות שהתוצאות אכן כמו שציפינו לתוכנית סידרתית.

פונקצית mesh:

את שתי הפונקציות (גם parallel וגם serial) רשמתי בעזרת אותן פונקציות, כך שיש התחשבות בכמה תהליכים או תהליך אחד.

ראשית בניתי תהליך monitor שמפקח על העבודה והוא למעשה יסיים את התהליך כאשר C הודיע לו $(N^2-1)\cdot M$ פעמים כי התקבלה הודעה חדשה שמיועדת אליו (אל C).

נוסח ההודעות שנשלחות הוא $\{From, To, Target, Messege\ index\}$:

- $From$: מתאר ממי נשלחה ההודעה במקור, כלומר מי הוא זה שיצר את ההודעה.
- To: מתאר אל מי **ברגע** הגיע ההודעה. כך נוכל לעקוב אחר התהליכים והמיקומים של ההודעה בלוח.
- $Target$: אל מי מיועדת ההודעה. הודעה שהפיץ *C* לכולם תהיה בנוסח כללי שכולם יזהו: $\{to, all\}$, ואילו כל הודעה אחרת שיוזמת משבצת אחרת בלוח תכיל בחלק זה את *C* עצמו שידעו לכולם.
- $Messege\ index$: מספר בין 1 ל-*M* המעיד איזה מספר הודעה זו.

לכל משבצת מפה משלה שבה יאוחסנו הודעות שהתקבלו, על מנת שלא נתייחס לאותה הודעה שוב.

למשבצת C יש פעולות משלה, שהן שליחת הודעות לכל השכנים כדי שיעבירו לכל הלוח – הודעות אלה מודיעות לכל משבצת שתתחיל לשלוח הודעות חזרה ליעד C. בנוסף עוד פעולה שקיימת למשבצת C היא עדכון כמות ההודעות שהתקבלו עבור אינדקס הודעה ספציפי. עדכון זה נעשה ע"י שליחת הודעה עם האינדקס הרלוונטי אל ה-monitor.

כאשר אנו מדברים על הפונקציה הסיריאלית, mesh_serial, נעשים אותם הדברים בדיוק, רק במקום לשלוח ל-PID-ים שונים, אנו שולחים ל-PID יחיד בשם "$worker$".

להלך תוצאות לדוגמה עבור C=1, כלומר בפינת הלוח (הזמנים במילי-שניות – מאחר ונדרשת תוצאה בשלמים ייתכן ויהיה 0):

		M=10	M=100	M=1000
mesh_parallel	***N=2***	3	14	140
mesh_serial		4	38	398
mesh_parallel	***N=3***	7	79	1,002
mesh_serial		24	254	3,338
mesh_parallel	***N=4***	28	266	3,635
mesh_serial		81	999	14,298
mesh_parallel	***N=5***	76	766	10,084
mesh_serial		223	2,479	32,939

ניתן לראות כי מאחר וזו פעולה מקבילית, הפונקציה המשתמשת בN^2+1 תהליכים מקביליים על פני $2+1$ יעילה הרבה יותר מבחינת זמן חישוב.

להלן התוצאות עבור *N=5* כאשר *C=13* (באמצע הלוח):

		M=10	*M=100*	*M=1000*
באמצע הלוח				
mesh_parallel	***N=5***	*51*	*670*	*8,748*
mesh_serial	***C=13***	*165*	*2,196*	*29,143*
בפינת הלוח				
mesh_parallel	***N=5***	76	766	10,084
mesh_serial	***C=1***	223	2,479	32,939

ניתן לראות כי העובדה ש-*C* במרכז הלוח, אכן משפרת במקצת את התוצאות של שתי הפונקציות, וזאת מכיוון שבמרכז הלוח ההודעות המרחק הכי גדול למשבצת נחתך פי 2 (למשל מרחק מ-[1,1] לפינה [*N,N*] קטן למרחק מ-[$\frac{N}{2}$,$\frac{N}{2}$] לפינה [*N,N*]).

* יתכן והתוצאות יצאו זהות למקרה ש-*C=1*, אך לרוב התוצאות הן כפי שפירטתי כאן.