Introduction to Parallel Processing

Home assignment #3: Exponent estimation using Salt Shaker by Monte Carlo.

Itay Eitani 315871764

Ron Milutin 316389584

הקדמה

בעבודה זו נתבקשנו לבצע חישוב מקורב לe. רעיון האלגוריתם הוא לקחת דף עם חור באמצע ברדיוס כלשהו, לשפוך מלח מעל הדף ולאסוף אחרי כל שלב את המלח שנשאר על הדף. אם נמשיך בדרך זו מספר רב של פעמים, נוכל לחלק את מספר החלקיקים הראשוני של המלח במספר החלקיקים שנותרו על הדף לאחר n איטרציות ונקבל את הקירוב לe. העבודה המקבילית באה לידי ביטוי בכך שכדי להגיע לחישוב המקורב של e הרצנו 128 הערכות של e כאשר כל הרצות אלה מתחלקות בין מספר הטרדים שמוגדרים בטרמינל. לאחר מכן סכמנו את כל התוצאות מכל המעבדים והצגנו את החישוב המקורב לe.

בעבודה זו נשתמש בנוסחאות של Speedup, Efficiency וWork אשר למדנו בהרצאה הראשונה:

- **Speedup**: t_s / t_n , $0 \le \text{speedup} \le n$
- **Work (cost)**: $n \cdot t_n$, $t_s \le W(n) \le \infty$
  (number of numerical operations)
- **Efficiency**: $t_s / (n \cdot t_n)$, $0 \le \varepsilon \le 1$
  (w_1/w_n)

תמונה 1. נוסחאות של Speedup, יעילות ועבודה.

כאשר הזמנים בתמונה 1 הם "הזמן הטורי", כלומר זמן הריצה הטוב ביותר שתרוץ התוכנית על ידי מעבד אחד, ו"זמן מקבילי" כלומר זמן הריצה שתלוי בn, יהיה תלוי במספר המעבדים שירוצו במקביל בביצוע התוכנית.

מהלך ניסוי

כפי שנאמר בניסוי הנ"ל התבקשנו לבצע חישוב מקורב של e בעזרת חישובים מקביליים.

ראשית נסביר את האלגוריתם של התוכנית שכתבנו.

רעיון האלגוריתם הוא לקחת את המשימה הגדולה, של 128 חישובים בלתי תלויים של e, ולחלק אותה באופן שווה בין הטרדים, ואז הרצת במקביל ולסוף סכימה של כלל התוצאות מהם.

אם נכנס לפרטי האלגוריתם, סך הכל יש 128 הרצות בלתי תלויות, כאשר כל הרצה מסמלצת את נפילת המלח על הניירת 10,000 פעמים כאשר סך החלקיקים הוא 100,000. בכל שלב נספור כמה חלקיקים נפלו ונוריד את הכמות הזו מסך חלקיקי המלח שנזרוק על הנייר בפעם הבאה. נדמה את חלקיקי המלח ע"י הגרלת מספר רנדומלי בין 0 ל1. כאשר המספר יהיה קטן מההסתברות של המלח ליפול, שמדמה את יחס השטחים של החור חלקי יחס השטח של כל הדף, אז נחשיב את חלקיק המלח כאילו נפל דרך החור. כדי לקבל קירוב טוב של e נרצה שהחור יהיה יחסית מאוד קטן לשטח הדף ולכן לקחנו את ההסתברות 10^-4 (נתון בתרגיל).

להלן התוצאות עבור הרצות של מעבד 1,2,4, ו8:

```
[hpc-user@hpc ~]$ nano hw3.c
[hpc-user@hpc ~]$ export OMP_NUM_THREADS=1
[hpc-user@hpc ~]$ gcc -fopenmp -o hw3 ./hw3.c
[hpc-user@hpc ~]$ ./hw3
Final estimate for 'e' is: 2.719024569563
RunTime: 413.938499 seconds
[hpc-user@hpc ~]$ export OMP_NUM_THREADS=1
[hpc-user@hpc ~]$ gcc -fopenmp -o hw3 ./hw3.c
[hpc-user@hpc ~]$ export OMP_NUM_THREADS=2
[hpc-user@hpc ~]$ gcc -fopenmp -o hw3 ./hw3.c
[hpc-user@hpc ~]$ ./hw3
Final estimate for 'e' is: 2.718300513615
RunTime: 226.175180 seconds
[hpc-user@hpc ~]$ export OMP_NUM_THREADS=4
[hpc-user@hpc ~]$ gcc -fopenmp -o hw3 ./hw3.c
[hpc-user@hpc ~]$ ./hw3
Final estimate for 'e' is: 2.718556550176
RunTime: 263.956598 seconds
[hpc-user@hpc ~]$ export OMP_NUM_THREADS=8
[hpc-user@hpc ~]$ gcc -fopenmp -o hw3 ./hw3.c
[hpc-user@hpc ~]$ ./hw3
Final estimate for 'e' is: 2.717960519813
RunTime: 237.575801 seconds
[hpc-user@hpc ~]$
```

תמונה 2. דוגמת הרצה במעבד 1, 2, 4 מעבדים ו8 מעבדים.

ניתן לראות בתמונה 2 שככל שנגדיל את מספר הטרדים, ונראה זאת עוד בהמשך בניתוח של יעילות וspeedup, נקבל זמן ריצה קטן יותר. הדבר שוב מסתדר עם מה שלמדנו בכיתה, כי הדבר קורה עקב חלוקה של משימה מסובכת למספר טרדים, אך בפועל נראה בהמשך כי לא תהיה ירידה בזמן הריצה.

השוואת זמני ריצה בשימוש מספר טרדים

כפי שנראה בהמשך על ידי השוואה בין זמני ה.s.u והיעילות, נצפה שככל שנשתמש ביותר טרדים, כל טרד יקבל משימה קטנה יותר לביצוע, וכך כשכמה טרדים יעבדו יחד נקבל אותה תוצאה בזמן ריצה קצר יותר (עם זאת נגלה בהמשך שישנם גורמים נוספים שמשפיעים על החישובים).

מהרצת המשימה, בשינוי מספר הטרדים התקבלו התוצאות הבאות (הוספנו לטבלה גם את חישובי הSU והיעילות):

Number of tasks	1st run	2nd run	3rd run	Average run	Speedup	Efficiency	Work (cost)
1	330	308	305	314	1	1	314
2	173	170	176	173	1.815029	0.907514	346
4	169	170	169	169	1.857988	0.464497	676
8	171	172	174	172	1.825581	0.304262	1376

טבלה 3. השוואת של זמני הריצה בין מספר המעבדים וכן חישובי הפרמטרים SU ויעילות ועבודה.

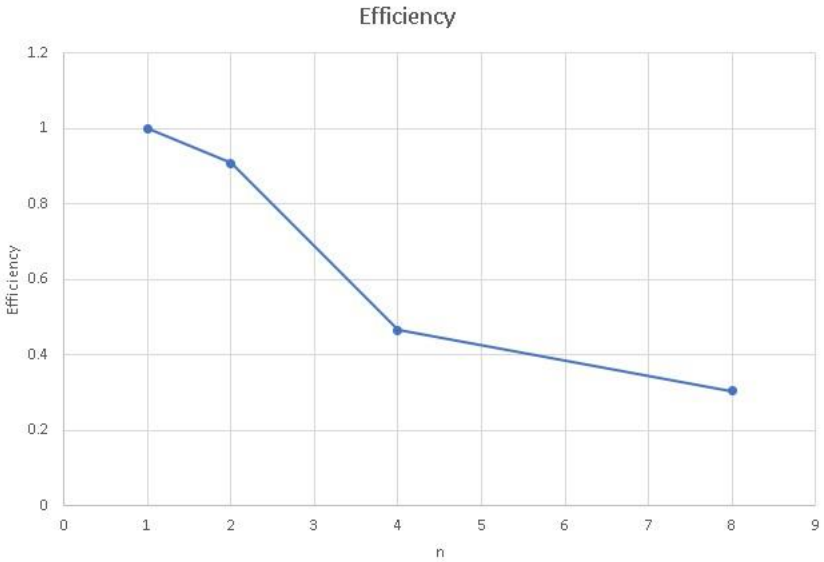
(נתייחס לעמודה של זמן הריצה המחושב הממוצע)

מטבלה זו ניתן לראות שכאשר פירקנו את המשימה לכמה טרדים קיבלנו ירידה בזמן הריצה, כאשר העבודה נעשית באופן מקבילי. עם זאת, ונסביר בהמשך, קיבלנו רוויה של זמן הריצה, כלומר הוא לא יוכל לרדת באופן משמעותי, וזאת עקב כך שאנחנו מריצים דרך המכונה הווירטואלית (VM) ומכיוון שהוקצו לה 2 ליבות, כאשר נבחר 2 תהליכונים ומעלה נקבל זמן ריצה זהה כלומר העבודה לא יכולה להתבצע ע"י מספר גדול יותר של ליבות.

Speedup and efficiency and work vs n

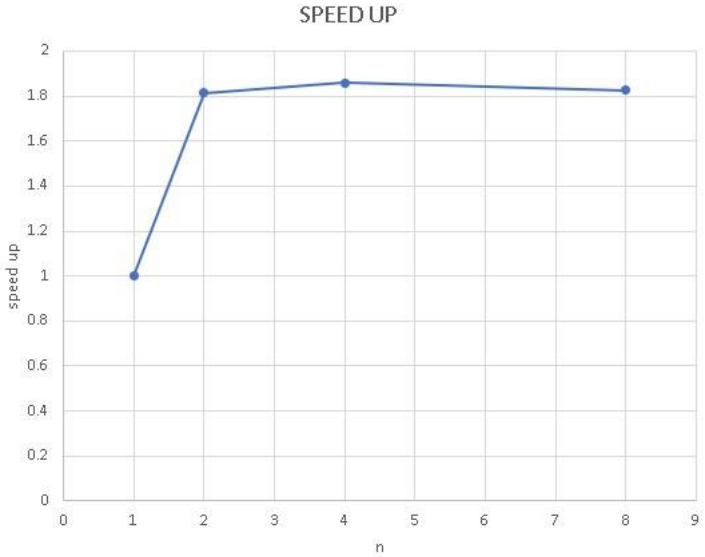
כפי שנאמר בפרק ההקדמה, נחשב בעזרת הנוסחאות את הפרמטרים .S.U, יעילות ועבודה כתלות בn. מהתוכנית עלו התוצאות הבאות:

(חשוב לציין שהצירים בגרפים הבאים חסרי יחידות ולכן לא ראינו צורך להוסיף אותם)



גרף 4. יעילות כתלות במספר התהליכים

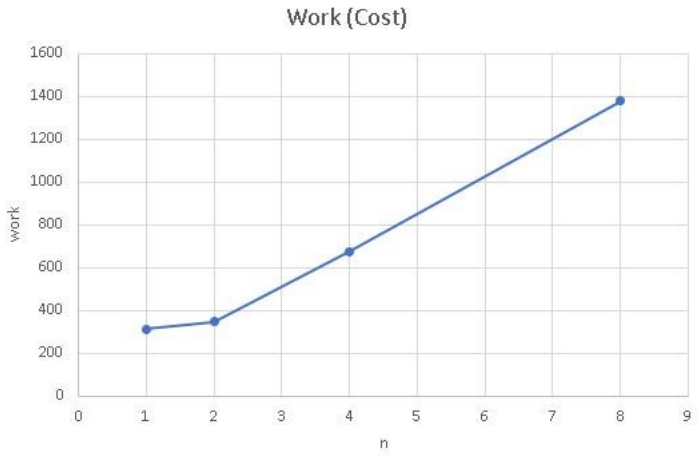
מגרף היעילות, ניתן לראות שככל שנעלה את מספר התהליכונים (n), נקבל ירידה ביעילות. הדבר מסתדר הגיונית וזאת מכיוון שיעילות מוגדרת על ידי הזמן הטורי, חלקי המכפלה בין מספר התהליכונים לזמן המקבילי. הזמן הטורי קבוע , ולמרות שהזמן המקבילי יורד כאשר מספר התהליכים עולה, הוא לא יורד באופן משמעותי, ולכן סך הכל המכנה יגדל, ולכן נקבל כי היעילות תקטן.



גרף 5. SU כתלות במספר התהליכים

מגרף הSpeedup ניתן לראות שכאשר נעלה במספר התהליכונים, נקבל עלייה בSU עד לנקודה של עבודה עם 2 תהליכונים, ואז נקבל su קבוע. הדבר נובע מכך שכאשר נעבוד עם מספר תהליכונים

גדול ממספר הליבות במערכת שלנו, נקבל כי זמן הריצה יישאר קבוע כי סך העבודה תתחלק בין 2 הליבות. (כזכור מהנוסחאות, הSU מחושב על ידי הזמן הטורי חלקי הזמן המקבילי)



גרף 6. עבודה כתלות במספר התהליכים

כזכור עבודה מחושבת על ידי מכפה של מספר התהליכונים בזמן הריצה עם מספר תהליכונים אלה. מגרף קודם ראינו כי זמן הריצה נשאר קבוע ולכן העבודה תגדל כתלות במספר התהליכונים. כלומר משמעות גרף זה שאין ניצול נכון של משאבי המחשב מאחר והעבודה גדלה משמעותית ככל שמספר התהליכונים יגדל מעל 2.

סיכום ומסקנות

מביצוע הניסוי קיבלנו תוצאות כפי שציפינו – זמן הריצה קטן ככל שמספר התהליכונים יגדל, ואז מגיע שלב, שבו מספר התהליכים גדול ממספר הליבות במחשב הוירטואלי (2), ואז נקבל זמן ריצה קבוע, היעילות תרד הSU ידיע לרוויה וכמות העבודה תעלה. כלומר החישוב המקבילי בשלב זה הופך ללא יעיל.

מחישוב הSU ראינו מהנוסחה כי הקשר בינו לבין הזמן המקבילי הוא הפוך, ואכן ניתן לראות זאת בתוצאות הניסוי(עד למצב כאמור שמספר התהליכים עולה על מספר המעבדים, ואז נקבל רוויה של הSU).

מחישוב היעילות ראינו התנהגות הפוכה מהתנהגות הSU, זאת מאחר שכאשר כמות המעבדים גדלה וזמן הריצה נשאר קבוע, נקבל סך הכל שהיעילות תרד כאשר מספר המעבדים יעלה. גם כאן ניתן לראות שאין צורך בעבודה עם מעל 2 תהליכונים כי יש שם ירידה משמעותית ביעילות.

דרך לשיפור האלגוריתם:

-ייתכן שישנם אלגוריתמים יעלים יותר לחישוב e (במאמר שניתן לנו ישנם מספר אפשרויות תחת מונטה קרלו).

הסקת מסקנות כללית מעבודה זו היא כי תכנות בעל אופי מקבילי מייעל את זמני הריצה, אך הדבר תלוי במספר הליבות שמוקצות למערכת שלנו. במידה ונחלק את העבודה למספר תהליכונים הגדול ממספר הליבות, נקבל עבודה גדולה יותר ולכן בזבוז של המשאבים שלנו.

חיבור 2 שאלות אמריקאיות

1) מה הפקודה של openMP שנועדה לסמן אזור מקבילי?
- #pragma omp barrier
- #pragma omp parallel
- #pragma omp task
- #pragma omp master

2) מה תפקיד המשתנה num_threads?
- מציין את מספר האיטרציות בלולאה
- קובע את מספר הטרדים לשימוש באזור המקבילי
- קובע את הסנכרון בין הטרדים באזור קריטי
- קובע את תזמון הטרדים באזור המקבילי