עיבוד מקבילי – עבודה 3

מגישים:

דור מולאי – 205870637
שניר מוסקוביץ' – 206293128

הקדמה:

הבעיה העומדת על הפרק היא חישוב הקבוע המתמטי 'e' בשיטת Shaker Salt, תוך שימוש במחשוב מקביל עם OpenMP.

המטרה היא לפתח אלגוריתם מקבילי שמעריך את 'e' על ידי ביצוע 128 ריצות עצמאיות של שיטת Shaker Salt ולאחר מכן מיצוע כל התוצאות. כל ריצה מורכבת מהדמיית 10^5 גרגרי חול העוברים 10^4 תנודות.

ההסתברות של גרגר ליפול בכל אחת מהתנודות היא 10^{-4}. התוכנית מודדת את זמן הריצה עבור 1, 2, 4 ו-8 תהליכונים (threads), ומבטיחה גודל מדגם עקבי לכל המקרים.

ההתמקדות העיקרית היא בפיתוח קוד מקבילי ב-OpenMP והערכת מדדי ביצועים: Run time, Speed up, Efficiency and Work (cost).

התוצאה הסופית מתקבלת על ידי מיצוע התוצאות של 128 הריצות כדי להעריך את הערך של 'e'.
לאחר מכן אנו נדרשים לצייר גרף עבור כל אחד ממדדי הביצועים כתלות במספר התהליכונים שעבדו במקביל.

מהלך הניסוי:

- אנו פותרים את הבעיה על ידי שימוש ב-openmp על מנת לחלק את העבודה בין מספר תהליכים שונים בכל פעם ומשווים את התוצאות המתקבלות.

- באלגוריתם אנו יוצרים בכל פעם מספר תהליכים (כמספר הנדרש) ומחלקים את העבודה באופן שווה בין כולם – כל תהליך אחראי על אותו מספר "נקודות".
על מנת לבצע זאת אנו משתמשים בפקודה:

```
#pragma omp parallel for reduction(+:total_num_of_fall)
```

  - שורה זאת מציינת שבלולאת for הבאה אחרי השורה התהליכים יעבדו בקביל, והיא גם אחראית על חלוקת העבודה באופן שווה.
  - 'reduction(+:total_num_of_fall)' מציין שבסוף הלולאה, שם נגמרת העבודה המקבילית הפעולה חיבור (+) מתבצעת על המשתנה total_num_of_fall כך שנקבל את הסכום שכל תהליך חישב בנפרד.
  - פקודה זו עוזרת למנוע טעויות משתמש שיכול לקרות על ידי שימוש בזיכרון משותף.

- אם ההסתברות של גרגר ליפול היא P, אז ההסתברות שלו לא ליפול היא 1-p, ומספר הגרגירים שיישארו לנו (שלא נפלו) אחר n נדנודים הוא: $N_n = (1-p)^n * N \leftarrow \frac{N_n}{N} =$ $(1-p)^n$.
כאשר N הוא מספר הגרגירים הכולל.
אם נשאיף את n לאינסוף $(n \to \infty)$ ונגדיר את $n = \frac{1}{p}$, נקבל כתוצאה את e^{-1}.
נחשב הופכי ונקבל את e.

- חישוב זה מתבצע 128 פעמים, לאחר מכן תהליך המאסטר מבצע מיצוע של כל התוצאות על מנת לקבל ערך מדויק כמה שיותר.

- תהליך המאסטר אחראי על חישוב הזמן הכולל של האלגוריתם – מודד זמן התחלה וזמן סיום לאחר שקיבל את המידע משאר התהליכים וסיים לחשב את הערך e.

- לבסוף תהליך המאסטר מדפיס את התוצאות: הערך של 'e' וזמן התהליך הכולל.

- חישוב כולל זה מתבצע עבור מספר שונה של תהליכונים בכל פעם – 1,2,4,8.

תוצאות:

טבלת התוצאות:

task	Run time	Speed-up	Efficiency	Work(cost)
1	1528.940523	1	1	1528.940523
2	905.289210	1.688	0.8444	1810.5784
4	1342.232601	1.139	0.2847	5368.9304
8	1567.356732	0.975	0.1218	12538.8538

כאשר חישבנו את Speed-Up לפי:

$$Speed - Up = \frac{t_s}{t_n}$$

כאשר t_s הוא זמן ההרצה אשר מדדנו עבור $n = 1$ (תהליך 1)

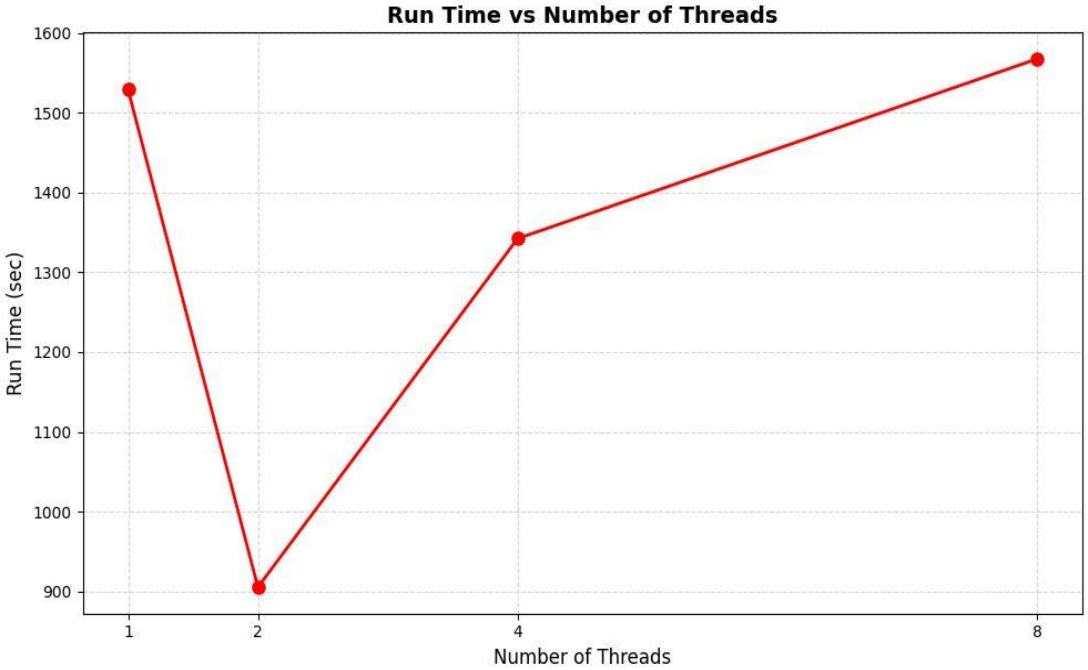
Efficiency:

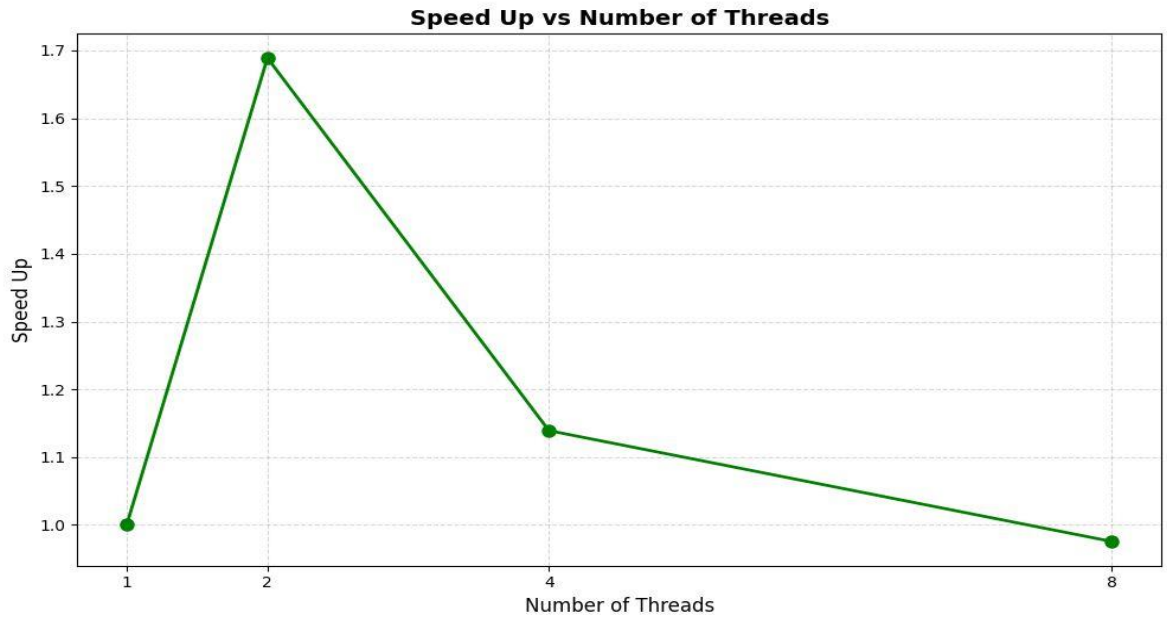
$$Efficiency = \frac{w_1}{w_n} = \frac{t_s}{n \cdot t_n}$$

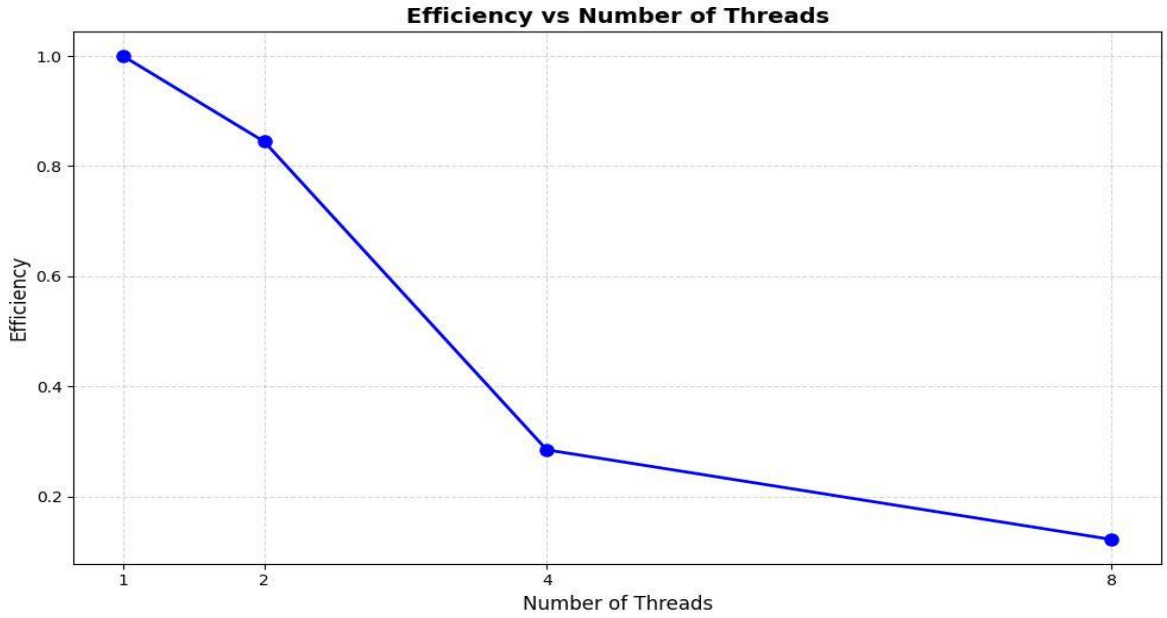
Work(cost):

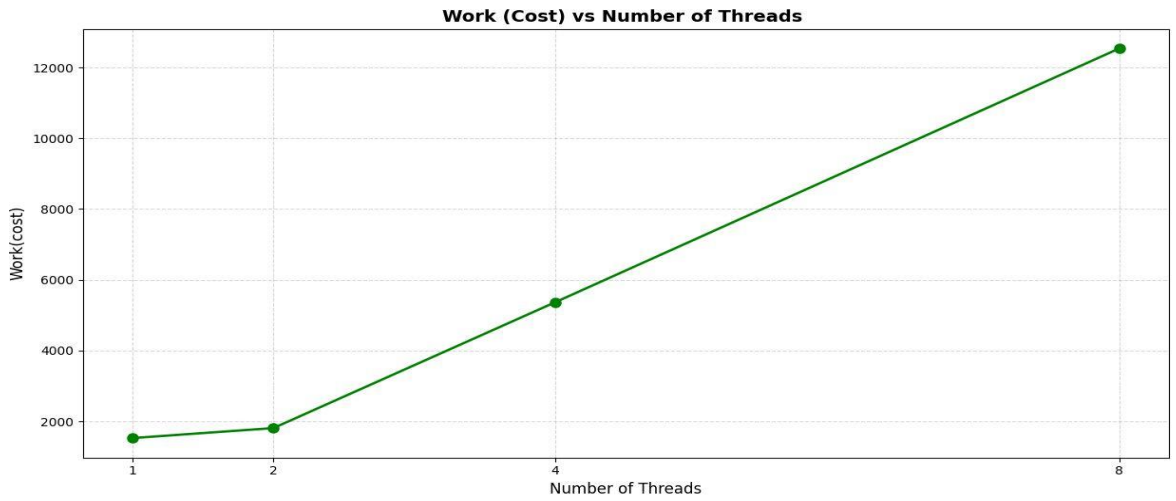
$$W(p) = p \cdot t_p$$

גרפים ומסקנות:




Speed Up vs Number of Threads
Speed Up
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
1
2
4
8
Number of Threads


Efficiency vs Number of Threads
Efficiency
1.0
0.8
0.6
0.4
0.2
1
2
4
8
Number of Threads


Work (Cost) vs Number of Threads
Work(cost)
12000
10000
8000
6000
4000
2000
1
2
4
8
Number of Threads

- ניתן לשים לב שכאשר אנו מבצעים את המשימה עם שני תהליכונים, ישנה ירידה בזמן מאשר אותה משימה עם תהליכון אחד, וכאשר אנו ממשיכים לבצע את המשימה עם 4 ו-8 תהליכונים אנו מקבלים זמן עבודה ארוך יותר (הגרף בעליה).
לאחר בדיקה אנו משערים שזה קורה כיוון שאנו עובדים ב-VM על המחשב שלנו, והמחשב מקצה לו 2 ליבות (core).

לכן במעבר מתהליכון אחד לשניים אנו רואים שיפור, כי ה-VM מנצל את כל הליבות שלרשותו, ואילו עם יותר תהליכונים אנו עליה כיוון שהמחשב מנצל את הליבות אך הרבה זמן מהעיסוק שלו מתבזבז על context-switch, שזה מעבר פעילות בין התהליכונים – פעולה המבזבזת זמן.

- אנו רואים שיפור של כמעט פי 2 בזמן בין תהליך אחד לשני תהליכים, באידיאל השיפור היה פי 2 בדיוק אך במציאות זה לא קורה כיוון שלא כל הקוד הוא מקבילי, בנוסף יש חלקים של סנכרון בהם התהליכים נדרשים להחליף את המידע (reduction) ולכן ישנה ירידה קלה בביצועים.
- גרף ה-Speed-Up הוא סוג של מראה לגרף ה-Run-Time, בדיוק לפי חישוב המשוואה.
- יש לשים לב ליחסי הגודל הגרפים (הסקאלה על מערכת הצירים קובעת את הפרופורציה).
- הסברים אלה משפיעים גם על שאר הגרפים, למשל על גרף ה- Work, באידיאל היינו מצפים שחישוב העבודה עבור תהליך 1 ושני תהליכים יהיה זהה, אך בפועל העבודה המחושבת עבור שני תהליכים גדולה יותר, ואילו העבודה עבור 4,8 תהליכים גדלה באופן ליניארי לפי אותו הסבר מלמעלה (context-switch).
- בגרף ה- Efficiency אנו רואים אותו דבר, היינו מצפים באידיאל לאותו ערך בשני תהליכים, אף בפועל מקבלים יעילות נמוכה יותר, ואילו ל4,8 תהליכים אנו מקבלים יעילות נמוכה בהרבה.
- קיבלנו קירוב די טוב (3 ספרות אחרי הנקודה) של e אך לא מושלם, אם היינו מבצעים יותר הרצות מ-128, ואולי גם יותר גרגירים היינו מקבלים קירוב טוב יותר.
- יש אפשרות למקבל בין התהליכים השונים את ההרצות (כל אחד מקבל מספר הרצות מתוך 128), או את הגרגירים בכל הרצה, או את הנדנודים של כל גרעין. יש להתאים את הקוד לכל אפשרות בהתאם (המשתנה אותו סוכמים וכו').

שאלות אמריקאיות:

1. באיזו פקודה משתמשים ב-OpenMP ליצירת אזור עבודה מקבילי?
   א. #pragma omp parallel
   ב. #pragma omp for
   ג. #pragma omp sections
   ד. #pragma omp critical

2. באיזו פקודה משתמשים ב-OpenMP כדי לחלק בין תהליכונים עבודה של איטרציות של לולאה?
   א. #pragma omp parallel
   ב. #pragma omp for
   ג. #pragma omp sections
   ד. #pragma omp critical

פתרונות: שאלה 1 - תשובה א'.
שאלה 2 - תשובה ב'.