Introduction to Parallel Processing

Home assignment #1: Estimating Π

Itay Eitani 315871764

Ron Milutin 316389584

הקדמה

בעבודה זו נתבקשנו לבצע חישוב מקורב לפאי, ע"י בניית ספירה וקובייה החוסמת אותה, הגרלת מספר רב של נקודות בתוך הקובייה, ואז חלוקה לנקודות השייכות לספירה מתוכן. העבודה המקבילית באה לידי ביטוי בכך שחילקנו את הקובייה לn פרוסות (בציר x), ועבור מחשב הוביט יחיד, כל מעבד ביצע את הגרלת הנקודות, ובדיקת המרחק של הנקודה מהראשית – עבור פרוסה בודדת. לאחר מכן סכמנו את כל התוצאות מכל המעבדים והצגנו את החישוב המקורב לפאי.

בעבודה זו נשתמש בנוסחאות של Speedup ושל Efficiency 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 ויעילות

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.

כמו כן, בעבודה זו נשתמש בכלי "פרופיילינג" - Jumpshot and Scalasca. כלים אלו הם כלי ניתוח ביצועים שמשמשים לזיהוי ואבחון בעיות בביצועים של תוכנית, אשר רצה באופן מקבילי.

ג'אמפשוט הינו קוד פתוח, מבוסס ג'אבה והוא כלי הדמיה לביצוע ניתוח ביצועים שמספקת מבנה היררכיה לאחסון מספר רב של אובייקטים הניתנים לציור בצורה מדרגית ויעילה לצורך הדמיה. בדרך כלל ישמש להמחיש ולנתח תוכניות המשתמשות בMPI. השימוש בו בעבודה זו יעשה על מנת לקבל ציר זמן של תהליך החישובים וכדי לראות את מהלך התקשורת בין המעבדים.

סקאלסקה נועד לניתוח ביצועים של תוכניות בעל אופי מקבילי. בעבודה זו השימוש שלו יהיה עבור השוואת הנתונים בצורה עמוקה יותר. בעזרת כלים אלו נוכל להביא לשיפור בביצועים של התוכנית שלנו.

מהלך ניסוי

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

ראשית נסביר את האלגוריתם של התוכנית שכתבנו.

רעיון האלגוריתם הוא לקחת את המשימה הגדולה, של הגרלת מספר גדול של נקודות בתוך קובייה, ולחלק את המשימה לתת משימות קטנות (יחסית) שיהיו בלתי תלויות אחת בשנייה, ואז הרצת המעבדים במקביל, ולבסוף סכימה של כלל התוצאות מהמעבדים. על כל פעולות אלה מנצח דרגה 0 של המחשב.

אם נכנס לפרטי האלגוריתם, ראשית דרגה 0 מגרילה n "זרעים" רנדומליים המתפלגים באופן אחיד, כאשר כל זרע יישלח לכל דרגה אחרת בפעולה, ובעזרת הזרע יבוא לידי ביטוי האי תלות בין המעבדים – בעזרתו הגרלת הנקודות בכל חלק תעשה אכן באופן רנדומלי ושונה מאשר כל מעבד אחר.

לאחר מכן האלגוריתם מבצע את סך כל ההגרלות של הנקודות הנדרשות לכל מעבד, ושולח את מספר הנקודות שנמצאו בתוך הספירה לדרגה 0. לבסוף דרגה 0 מרכזת את כל המידע מהמעבדים, ומדפיסה את התוצאות.

להלן התוצאות עבור הרצות של מעבד 1, 2 ו4:

```
hobbit1.ee.bgu.ac.il> ./hw1
Number of processes: 1
Number of points: 100000000
Number of points per process: 100000000
Pi = 3.141962
Absolute difference from reference value = 3.689064e-04
Execution time = 4.649796 seconds
hobbit1.ee.bgu.ac.il> mpirun -np 2 ./hw1
Number of processes: 2
Number of points: 100000000
Number of points per process: 50000000
Pi = 3.141444
Absolute difference from reference value = 1.484736e-04
Execution time = 2.236731 seconds
hobbit1.ee.bgu.ac.il> mpirun -np 4 ./hw1
Number of processes: 4
Number of points: 100000000
Number of points per process: 25000000
Pi = 3.141022
Absolute difference from reference value = 5.706336e-04
Execution time = 1.226706 seconds
hobbit1.ee.bgu.ac.il>
```

תמונה 2. דוגמת הרצה במעבד 1, 2 מעבדים ו4 מעבדים.

ניתן לראות בתמונה 2 שככל שנגדיל את מספר המעבדים, ונראה זאת עוד בהמשך בניתוח של יעילות וspeedup, נקבל זמן ריצה קטן יותר. הדבר שוב מסתדר עם מה שלמדנו בכיתה, כי ע"י חלוקה של משימה מסובכת למספר מעבדים, בהנחה שזמני התקשורת ביניהם לא יעלו את זמן הריצה, נקבל זמן ריצה טוב יותר, בפעולה של כמה מעבדים במקביל.

השוואת זמני ריצה בשימוש מספר מעבדים

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

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

Number of processes	1st run	2nd run	3rd run	Average run	Speedup	Efficiency
1	4.71	4.65	4.72	4.69	1	1
2	2.22	2.19	2.21	2.205	2.1269841	1.06349206
4	1.22	1.22	1.22	1.22	3.8442623	0.96106557
6	0.87	0.87	0.83	0.86	5.4534884	0.90891473
8	0.62	0.68	0.69	0.66	7.1060606	0.88825758
10	0.88	0.82	0.85	0.85	5.5176471	0.55176471
12	0.75	0.8	0.79	0.78	6.0128205	0.50106838
14	0.73	0.72	0.76	0.74	6.3378378	0.4527027
16	0.66	0.7	0.69	0.68	6.8970588	0.43106618

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

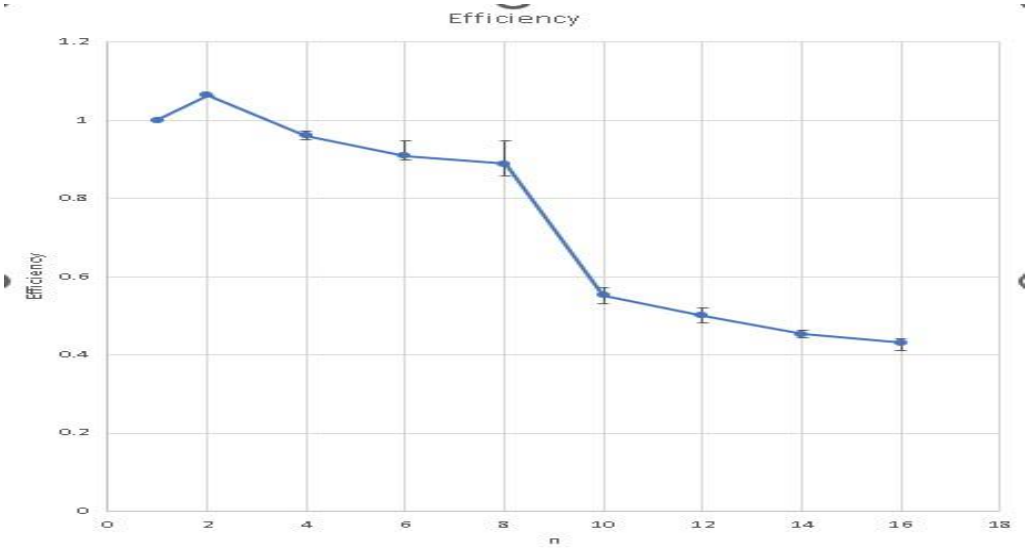
(נתייחס לעמודה של זמן הריצה המחושב הממוצע)

מטבלה זו ניתן לראות שכאשר פירקנו את המשימה לכמה מעבדים קיבלנו ירידה בזמן הריצה, כאשר העבודה נעשית באופן מקבילי. אם זאת, ונסביר בהמשך, קיבלנו מעין רוויה של זמן הריצה, כלומר הוא לא יוכל לרדת באופן משמעותי, וזאת עקב התקשורת הרבה בין המעבדים, שמעלה את זמן הריצה, למרות שנצפה שזמן הריצה של כל מעבד בממוצע יהיה נמוך יותר (עקב הגרלת מספר נמוך יותר של נקודות).

Speedup and efficiency vs n

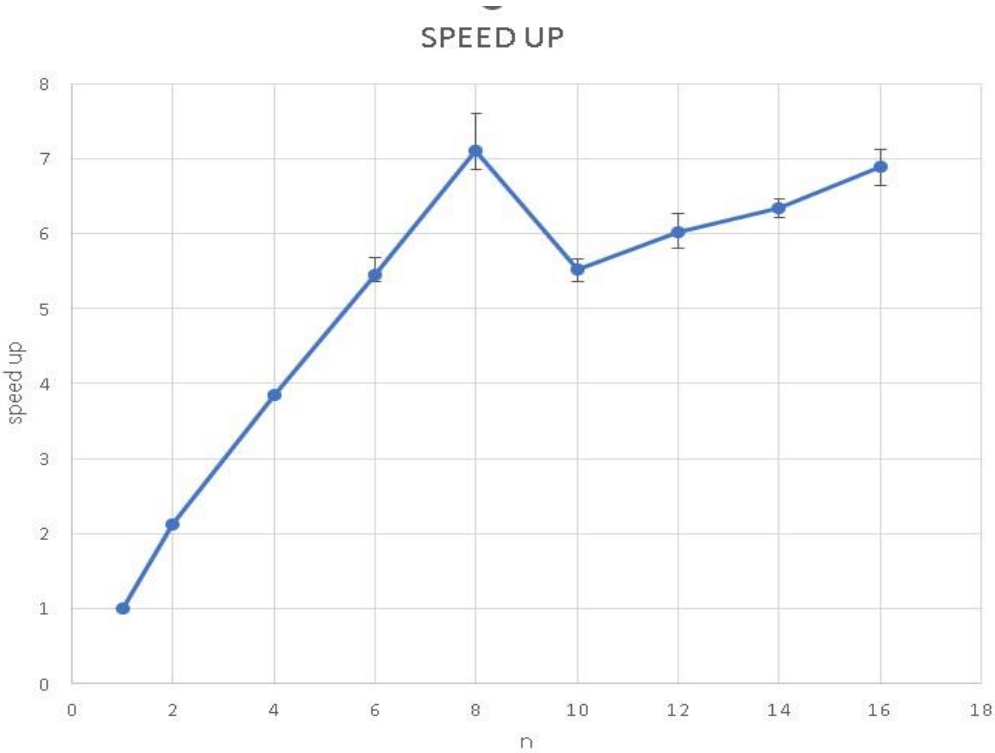
כפי שנאמר בפרק ההקדמה, נחשב בעזרת הנוסחאות את הפרמטרים .S.U ויעילות כתלות בn. מהתוכנית עלו התוצאות הבאות:

(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 התהליכים (n), נקבל ירידה ביעילות. הדבר מסתדר הגיונית וזאת מכיוון שיעילות מוגדרת על ידי הזמן הטורי, חלקי המכפלה בין מספר התהליכים 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.

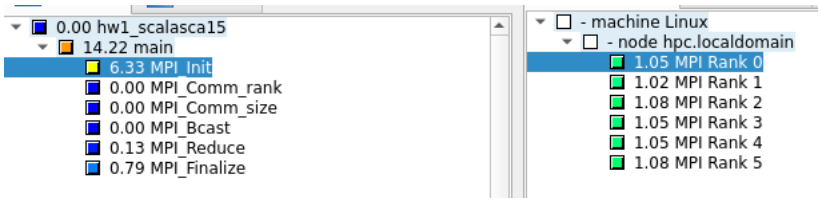


גרף 5. SU כתלות במספר התהליכים

מגרף הSpeedup ניתן לראות שכאשר נעלה במספר התהליכים, נקבל עלייה בSU עד לנקודה של עבודה עם 8 תהליכים, ואז נקבל ירידה וכן עלייה מתונה. הדבר נובע מכך שכאשר נעבוד עם מספר תהליכים גדול ממספר הליבות במערכת שלנו, נקבל כי רוב זמן הריצה של התוכנית יושקע בתקשורת בין המעבדים, ולכן זמן הריצה לא ירד כפי שהיינו מצפים לפי הגרף במספר מעבדים נמוך. (כזכור מהנוסחאות, הSU מחושב על ידי הזמן הטורי חלקי הזמן המקבילי)

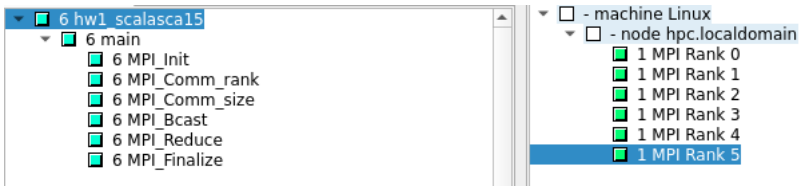
Jumpshot and Scalasca

מניתוח של הכלי סקלסקה קיבלנו את התוצאות הבאות:



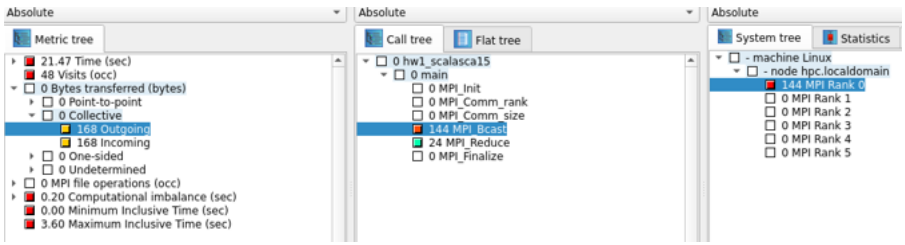
תמונה 6. ניתוח סקלסקה – זמן ריצה.

מניתוח זמן של התוכנית, ניתן לראות כי רוב זמן הריצה הושקע בMPI_init והזמן התחלק יחסית באופן שווה בין המעבדים, כלומר מבחינת זמן הריצה של פעולות mpi היא הכי כבדה מביניהן.



תמונה 7. ניתוח סקלסקה – ביקורים.

מבחינת ביקורים של התוכנית בפונקציות השונות, מכיוון שהרצנו את סקלסקה על ידי 6 מעבדים, כל מעבד ייכנס לפונקציה פעם אחת.

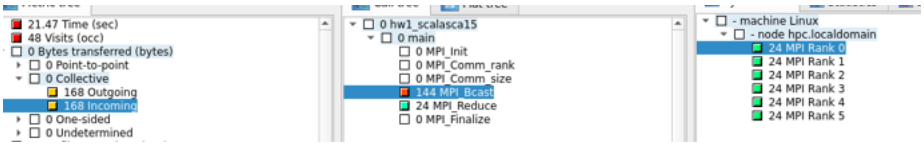


תמונה 8. ניתוח סקלסקה – bcast.

מתמונה 8 ניתן לראות את החלוקה לפקודות תקשורת יוצאות ונכנסות. כאן ניתן לראות את הפקודה היוצאת bcast, שהיא שידור משתנה לכלל המעבדים, והיא מבוצעת על ידי דרגה 0 בלבד, ואכן ניתן לראות בצד ימין.

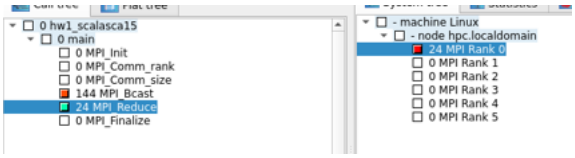
תמונה 9. ניתוח סקלסקה - reduce.

מתמונה 9 ניתן לראות כי פונקציית רדוקציה, שבמקרה זה הכוונה לתקשורת היוצאת, מכילה את כל שליחת התקשורת של הנתונים מהדרגות שמבצעות את החישובים לדרגה 0.



תמונה 10. ניתוח סקלסקה – bcast_incoming.

מתמונה 10 ניתן לראות את תעבורת התקשורת הנכנסת בתוכנית. אכן בbcast רק דרגה 0 שולחת את המידע לכל שאר הדרגות, אך המידע נקלט בכל המעבדים וניתן לראות זאת בצד ימין.

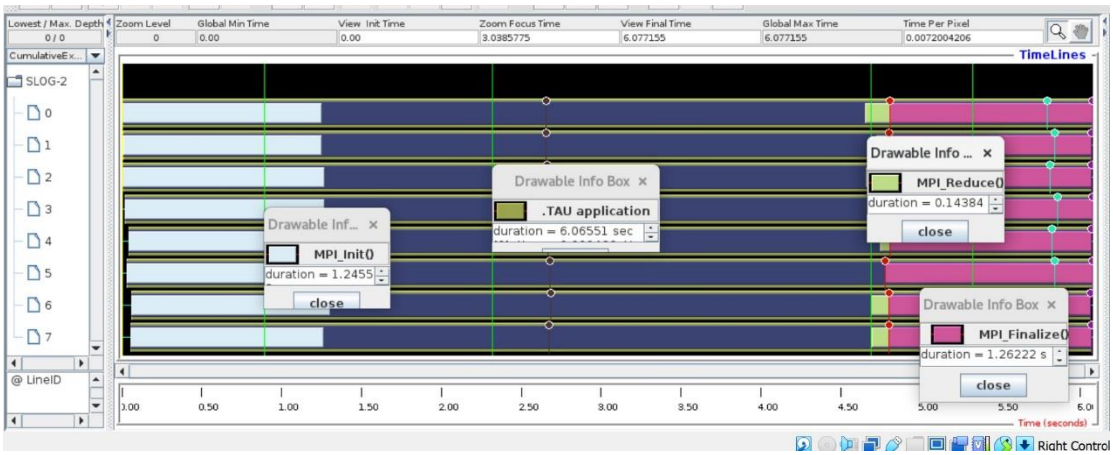


תמונה 11. ניתוח סקלסקה – reduce _incoming.

שוב בתמונה 11 ניתן לראות את פעולת איסוף המידע של פונקציה הרדוקציה, והדבר נעשה על ידי דרגה 0 בלבד.

מסקלסקה ניתן לראות באופן ברור את אופן העברת המידע בין המעבדים, ואת זמני הריצה של הפונקציות השונות.

ביצענו ניתוח עם כלי Jumpshot על 8 מעבדים. נקבל תמונה ביחס לציר הזמן של כל פקודה שמתבצעת בכל מעבד:



תמונה 12. ניתוח ג'אמפשוט.

בגרף הנ"ל ציר ה X הוא ציר הזמן וציר ה Y מתייחס למספר המעבד. כל צבע מצויין לידו בריבוע לאיזו פקודה הוא מתייחס.

נבחין מהתמונה שזמני האתחול והסיום (MPI_Finalize ,MPI_Init, הצבעים והתכלת והורוד) לוקחים זמן משמעותי מאד מריצת התוכנית. הכחול זהו זמן ריצת התוכנית.

סיכום ומסקנות

מביצוע הניסוי קיבלנו תוצאות כפי שציפינו – זמן הריצה קטן ככל שמספר המעבדים יגדל, ואז מגיע שלב, שבו מספר התהליכים גדול ממספר המעבדים של השרת, ואז זמן התקשורת ייקח חלק גדול מזמן הריצה, ונקבל זמן ריצה שאינו משתפר בעליית התהליכים ואולי אף גדל. 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 ואז ירידה).

מחישוב היעילות ראינו התנהגות הפוכה מהתנהגות הSU, זאת מאחר שכאשר כמות המעבדים גדלה וזמן הריצה קטן במקצת, נקבל סך הכל שהיעילות תרד כאשר מספר המעבדים יעלה. כמובן שגם כאן יש קשר ישיר לתקשורת שכן התקשורת בין מספר מעבדים מורידה את היעילות.

ישנן כמה דרכים לשיפור האלגוריתם:

-ישנם אלגוריתמים יעילים יותר לחישוב פאי כמו למשל בשיטת גאוס לז'נדר.

-הפקודות bcast וכן reduce הן פקודות שעלולות להוות חסימת תקשורת בין המעבדים. ישנן פקודות מקבילות אליהן שלא למדנו, ובעזרת ניתן לאפשר לתהליכים לעבוד בזמן המתנה לקבלת מידע ממעבדים אחרים.

-במקום שדרגה 0 תייצר n זרעים רנדומליים ותשלח לכל המעבדים, כל מעבד יכול בעצמו לייצר זרע משלו (בעזרת זמן והמזהה הייחודי שלו) ואז התוכנית תחסוך בזכרון, וכמובן בזמן תקשורת עקב פעולת bcast.

הסקת מסקנות כללית מעבודה זו היא כי תכנות בעל אופי מקבילי מייעל את זמני הריצה, אך הדבר תלוי במספר המעבדים שקיימים לנו במערכת. במידה ונחלק את העבודה למספר תהליכים הגדול ממספר המעבדים, נקבל זמן תקשורת גבוהה יותר, ונוכל לקבל הרעה בביצועים.

חיבור 2 שאלות אמריקאיות

1) מה תפקיד mpi_reduce()?
- להפיץ מידע למספר מעבדים
- לאסוף מידע ממספר מעבדים, ולבצע פעולת רדוקציה
- לסנכרן בין זמני הריצה של המעבדים
- לעצור את הריצה של כל המעבדים

2) מה מהבאים לא נחשב למחשוב מקבילי?
- SIMD
- MIMD
- SISD
- MISD