Introduction to Parallel Processing

Home assignment #2: N-body problem using MPI

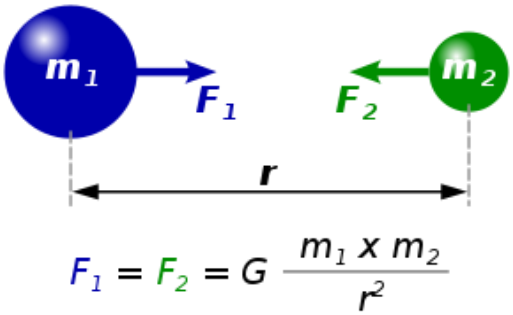
Itay Eitani 315871764

Ron Milutin 316389584

הקדמה

בעבודה זו נתבקשנו לבצע סימולציה של בעיית N הגופים ב2 מימדים, ע"י שימוש בספריית MPI.

בעיה זו היא בעיה שמוכרת בפיסיקה, ותפקידה לחשב מהם המיקומים של N גופים במרחב כתוצאה מכוח הכבידה הפועל בין כל שני גופים. כוח הכבידה בין שני גופים מחושב ע"י הנוסחה של ניוטון:



תמונה 1. חישוב הכוחות בין שני גופים במרחב דו ממדי.

לפי נוסחה זו, הכוח בין כל שני גופים מחושב כמכפלת קבוע הגרביטציוני עם המסה של שני הגופים, חלקי המרחק ביניהם בריבוע. בעבודה זו נניח כי המסה של כל הגופים שווה למסת השמש, ומהירות הגופים שווה.

העבודה המקבילית באה לידי ביטוי בכך שכל מעבד יקבל מספר שווה של גופים, ויבצע את החישובים של התנועה עבור כל גוף ושאר הגופים במרחב, ואז בסוף כל מקטע זמן, לאחר שכל המעבדים ביצעו את החישובים, המעבדים יעבירו את המידע בין אחד לשני כדי לבצע את החישובים במקטע הזמן הבא וכך הלאה. החישובים כמובן מבוצעים עבור מחשב הוביט יחיד.

בעבודה זו נשתמש בנוסחה של Speedup אשר למדנו בהרצאה הראשונה:

- **Speedup:** t_s / t_n , $0 \le speedup \le n$
- **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 2. נוסחה של Speedup

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 סימולציה של בעיית N הגופים בעזרת ספריית MPI.

ראשית נסביר את האלגוריתם של התוכנית שכתבנו.

רעיון האלגוריתם הוא לקחת את המשימה הגדולה, של חישוב הכוחות הפועלים בין כל שני גופים, ועדכון המיקום שלהם בהתאם לכוחות אלו, ולחלק אותה בין כל המעבדים בכדי לחסוך בזמן ריצה.

הרעיון הוא שבכל צעד זמן, מיקום ומהירות כל גוף צריכים להתעדכן לפי הכוחות הפועלים בין הגוף לכל שאר הגופים. כדי לחשב זאת כל מעבד יקבל חלק שווה של חישובים לבצע בתחילת כל צעד זמן, ואז בסוף צעד הזמן כל המידע יופץ לשאר המעבדים, כדי שהחישוב בזמן הבא יוכל להיות מעודכן לפי התנועה המעודכנת של הגופים.

לאחר שהזמן הכולל הגיע לפרק זמן שהגדרנו מראש, וזאת כדי לא לעלות על פרק הזמן שהוגדר במשימה (לפחות 2 דקות), המעבדים יפסיקו לבצע את החישובים, התוכנית תדפיס את מיקום הגופים ותסיים את ריצתה.

להלן התוצאות עבור הרצות של מעבדים 2, 8, 16:

```
.
Star 988: pos x=0.202419, pos y=0.751446, vel x=-1965.970457, vel y=563.186355.
Star 989: pos x=0.555176, pos y=0.740106, vel x=4221.837646, vel y=4280.184221.
Star 990: pos x=0.962502, pos y=0.795804, vel x=138323.533930, vel y=-228663.691
265.
Star 991: pos x=0.307858, pos y=0.762070, vel x=-1120.362540, vel y=5402.901245.
```

תמונה 3. מיקום של הגופים בהרצה של 2 מעבדים

```
Star 988: pos x=0.910091, pos y=0.471675, vel x=915.686923, vel y=-946.735673.
Star 989: pos x=0.357426, pos y=0.270678, vel x=358.174120, vel y=-816.034917.
Star 990: pos x=0.093148, pos y=0.447482, vel x=-3222.976542, vel y=11193.451169
.
Star 991: pos x=0.176096, pos y=0.854515, vel x=9934.080529, vel y=30.735094.
```

תמונה 4. מיקום של הגופים בהרצה של 8 מעבדים

```
Star 988: pos x=0.809056, pos y=0.342503, vel x=12868.181442, vel y=6280.961601.
Star 989: pos x=0.376557, pos y=0.358186, vel x=9739.874242, vel y=-2088.679839.
Star 990: pos x=0.766918, pos y=0.541386, vel x=-984.549528, vel y=272.794759.
Star 991: pos x=0.963624, pos y=0.854731, vel x=-21135.649481, vel y=-2414.05053
4.
```

תמונה 5. מיקום של הגופים בהרצה של 16 מעבדים

תמונות אלו באות להמחיש שהקוד רץ תקין עבור מספר מעבדים שונה. את זמני הריצה וה speedup ננתח בהמשך.

<u>השוואת זמני ריצה בשימוש מספר מעבדים</u>

כפי שנראה בהמשך על ידי השוואה בין זמני ה.s.u, נצפה שככל 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:

Number of processes	1st run	2nd run	3rd run	Average run	Speedup
2	127.233	130.12	124.12	127.233	1
4	66.7	65.359	64.018	65.359	1.946679
6	42.9	47.63	45.02	45	2.8274
8	36.29	36.1	36.5	36.29	3.506007
16	65.3	68.07	66.3	66.7	1.907541

טבלה 6. השוואת של זמני הריצה בין מספר המעבדים וכן חישוב הפרמטר SU .

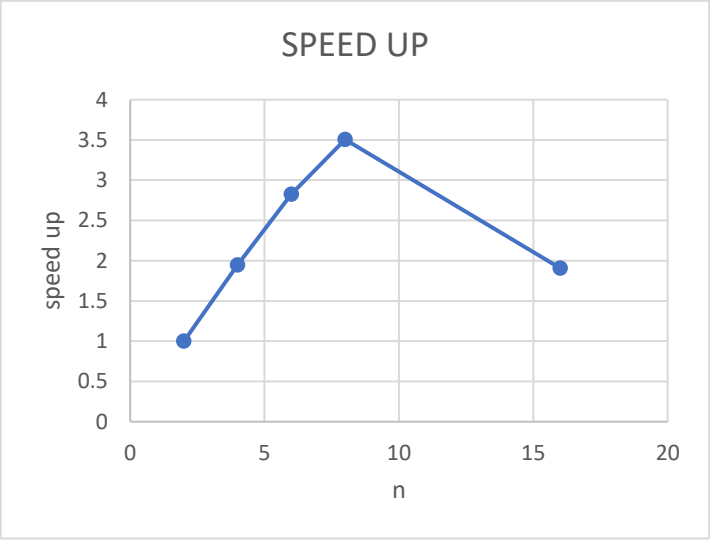
(נתייחס לעמודה של זמן הריצה המחושב הממוצע)

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 חלוקה של עומס המשימות בין המעבדים).

Speedup vs n

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

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

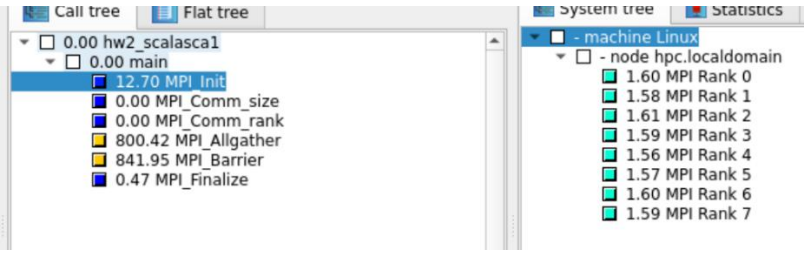


גרף 7. SU כתלות במספר התהליכים

מגרף הSpeedup ניתן לראות שכאשר נעלה במספר התהליכים, נקבל עלייה בSU עד לנקודה של עבודה עם 8 תהליכים, ואז נקבל ירידה. 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 (בזכור מהנוסחאות, הSU מחושב על ידי הזמן הטורי חלקי הזמן המקבילי). סיבה נוספת אפשרית לכך היא שעקב המחשב שלנו מורכב ממספר ליבות מוגבל, כאשר נבקש יותר תהליכים ממספר הליבות, חלק מהליבות למעשה יבצעו עבודה כפולה ולכן נקבל עלייה בזמן הריצה ולכן ירידה בspeedup.

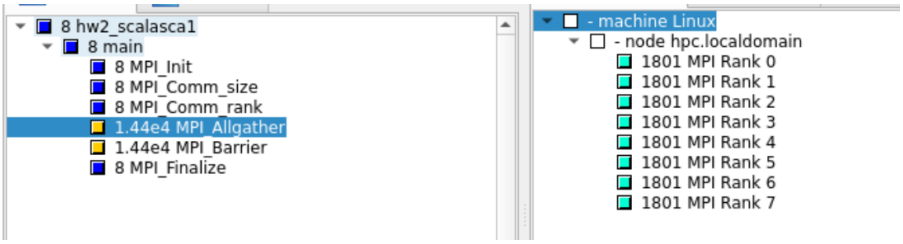
Jumpshot and Scalasca

מניתוח של הכלי סקלסקה קיבלנו את התוצאות הבאות:



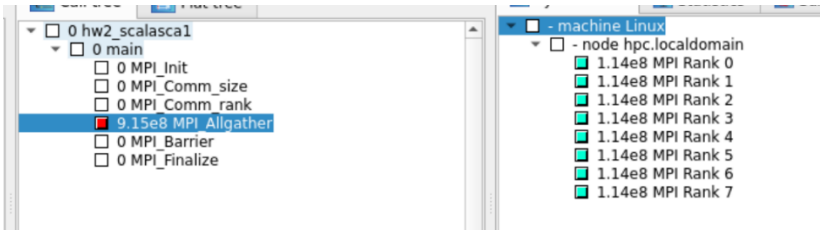
תמונה 8. ניתוח סקלסקה – זמן ריצה.

מניתוח זמן של התוכנית, ניתן לראות כי רוב זמן הריצה הושקע בgather and barrier והזמן התחלק יחסית באופן שווה בין המעבדים, כלומר מבחינת זמן הריצה של פעולות mpi הן הכי כבדות(זאת מאחר שהפעולה התבצעה מספר רב של פעמים).



תמונה 9. ניתוח סקלסקה – ביקורים.

מבחינת ביקורים של התוכנית בפונקציות השונות, מכיוון שהרצנו את סקלסקה על ידי 8 מעבדים, כל מעבד ייכנס לפונקציה פעם אחת, אך קיימת לולאה שכל מעבד נכנס אליה 1800 פעמים כי זה צעד הזמן שבחרנו.

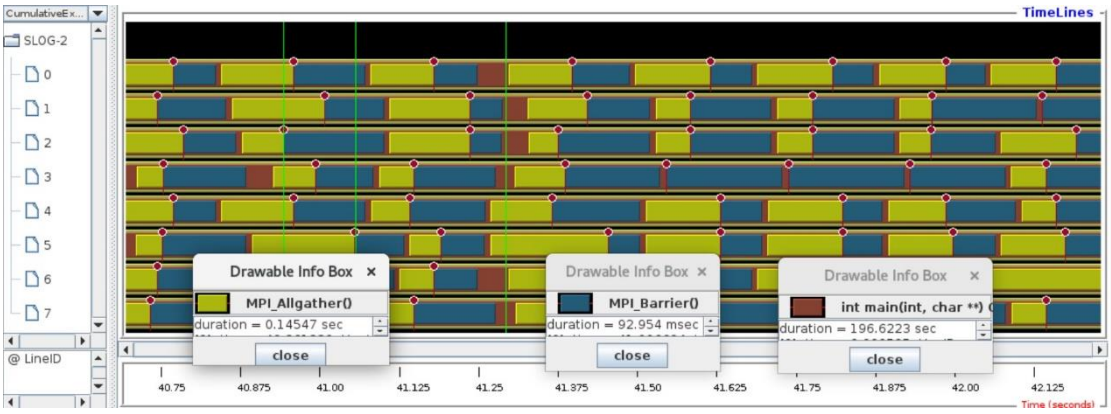


תמונה 10. ניתוח סקלסקה – ALLgather.

מתמונה 8 ניתן לראות את החלוקה לפקודות תקשורת יוצאות ונכנסות. כאן ניתן לראות את הפקודה שהיא שידור וקליטת משתנה לכלל המעבדים, והיא מבוצעת על ידי כל הדרגות, ואכן ניתן לראות זאת בצד ימין.

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 11. ניתוח ג'אמפשוט.

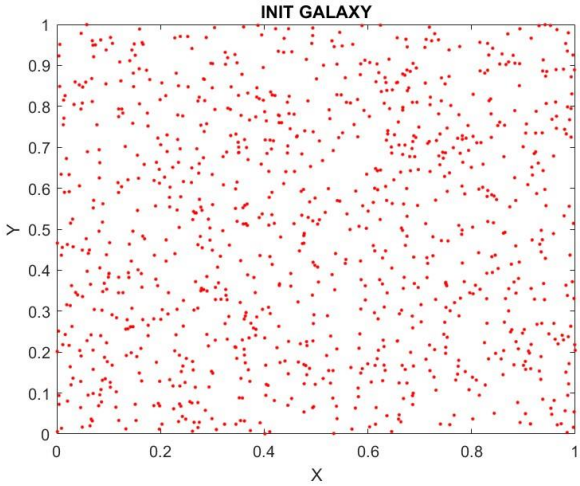
בגרף הנ"ל ציר ה X הוא ציר הזמן וציר ה Y מתייחס למספר המעבד. כל צבע מצויין לידו בריבוע לאיזו פקודה הוא מתייחס.

נבחין מהתמונה כי הפונקציות המרכזיות שלוקחות את זמן הריצה של המערכת הן ALLgather וכן Barrier.

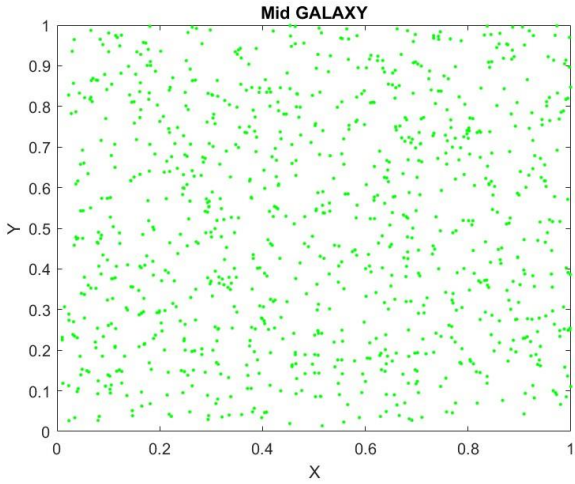
תמונות של הגלקסיה

בהרצה של הקוד קיבלנו מיקומים ומהירויות של הכוכבים בהתחלה אמצע וסוף. כדי להמיר את המיקומים לנקודות במרחב ולהפיק מהם תמונה יצרנו קוד במטלב (שיצורף כנספח).

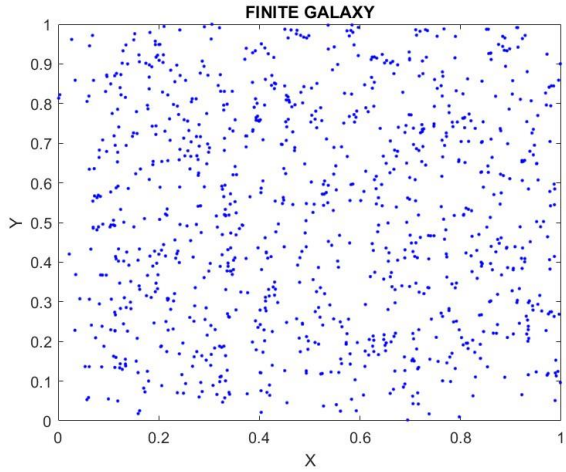
התקבלו התמונות הבאות:



תמונה 12. תמונת הגלקסיה באתחול המערכת.



תמונה 13. תמונת הגלקסיה באמצע ריצת המערכת.



תמונה 14. תמונת הגלקסיה בסוף ריצת המערכת.

ניתן לראות שישנה תנועה של הגופים לפי הבעיה הנתונה.

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 ירידה).

דרך לשיפור האלגוריתם:

-הפקודה barrier היא פקודה של בזבוז במשאבים ועדיף לא להשתמש בה אלא אם כן חייבים. ייתכן כי ניתן לוותר על ההדפסות ואז נחסוך בזמן הריצה וכן במשאבי המחשב. ישנן פקודות מקבילות אליהן שלא למדנו, ובעזרת ניתן לאפשר לתהליכים לעבוד בזמן המתנה לקבלת מידע ממעבדים אחרים.

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 מהבאים לא נחשב יתרון של עיבוד מקבילי?
- מהירות גבוהה וביצועים טובים יותר
- מודל תכנות פשוט יותר
- ניצול טוב יותר של משאבי המחשב
- סקאלביליות משופרת

2) מה מהבאים לא גישה נכונה בעיבוד מקבילי?
- תכנות עם זכרון משותף
- תכנות עם זכרון מופץ
- תכנות סריאלי
- תכנות GPU