עיבוד מקבילי – עבודה 2

מגישים:

דור מולאי – 205870637
שניר מוסקוביץ' – 206293128

הקדמה:

הבעיה איתה אנו מתמודדים היא הדמיה של בעיית N-body בשני ממדים (נאיבית) באמצעות מחשוב מקביל עם MPI. בעיית ה-N-body כוללת הדמיית יחסי הגומלין הכבידתיים בין N גופים שמימיים, כגון כוכבים, במרחב דו-ממדי.

האתגר העיקרי בבעיה זו הוא המורכבות החישובית. בגישה הנאיבית, כל כוכב מקיים אינטראקציה עם כל כוכב אחר, וכתוצאה מכך מורכבות חישובית של O(n^2). ככל שמספר הכוכבים גדל, זמן החישוב גדל באופן אקספוננציאלי.

כדי להתמודד עם אתגר זה, אנו משתמשים במחשוב מקביל עם MPI כדי לפזר את עומס העבודה על פני מספר תהליכים. כל תהליך אחראי על הדמיית תת-קבוצה של הכוכבים, והם מתקשרים ומסנכרנים את החישובים שלהם כדי לעדכן את המיקומים והמהירויות של הכוכבים במדויק.

אתגר נוסף הוא טיפול בתנאי הגבול של הסימולציה. הכוכבים ממוקמים תחילה בתוך תחום מרובע בגודל 100 שנות אור על 100 שנות אור. אם כוכב יוצא מהאזור המוגדר, עלינו להחליט כיצד לטפל בו. לדוגמה, אנו יכולים לשקף את הכוכב בחזרה פנימה או לגרום לו להיכנס מהקצה השני של הדומיין.

בנוסף, כדי להבטיח סימולציות מדויקות, עלינו לבחור ערכים מתאימים לשלב הזמן (dt) ולמספר הכולל של שלבי הזמן (כדי שהסימולציה תרוץ מספיק זמן – 2 דקות). שלב הזמן קובע את הפירוט של הסימולציה, והמספר הכולל של שלבי הזמן קובע את משך ההדמיה. חשוב לבחור ערכים אלו בקפידה כדי להשיג תוצאות משמעותיות ויציבות.

לבסוף, עלינו למדוד את הביצועים של הקוד המקביל שלנו כדי להעריך את יעילותו. אנו ננתח גורמים כגון שמך זמן הביצוע הכולל של האלגוריתם עם מספר שונה של תהליכים ומהירות המושגת על ידי הקבילה כדי להעריך את האפקטיביות של היישום שלנו.

מהלך הניסוי:

את הבעיה אנו פותרים באמצעות שימוש ב- MPI 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 כוכבים.

תהליך המאסטר (rank=0) אחראי על חישוב הזמן הכולל של האלגוריתם – מודד זמן התחלה וזמן סיום לאחר שקיבל את המידע משאר התהליכים.

תהליך המאסטר שולח לכל תהליך מספר seed שונה כדי לשמור על אקראיות הנקודות.
בפונקציית ה- init כל תהליך יוצר כוכבים ולכל כוכב מגריל מיקום (x, y) בטווח [0,100] (בשנות אור), מהירות לפי הפרמטרים בעבודה וזווית כדי לקבוע את כיוון המהירויות (vx, vy).

לאחר מכן משתמשים בפונקציית MPI_Allgather, שהיא פונקציה האוספת את המידע מכל התהליכים ומשתפת את כל המידע עם כל התהליכים, כלומר אחרי הקריאה לפונקציה כל תהליך מכיר את המיקום של כל הכוכבים במערכת.

לאחר מכן כל תהליך נכנס ללולאת for שרצה כמספר הכולל של צעדים שהגדרנו במערכת.
בכל איטרציה כל תהליך משתמש בפונקציות הבאות:

- **compute forces** – פונקציה זו מחשבת עבור כל כוכב (שהתהליך אחראי עליו) את הכוח הכולל שמופעל עליו משאר הכוכבים האחרים. כדי לחשב זאת אנו עושים המרה למרחק כדי לייצג אותו במטרים.
- **update positions & velocities** – בפונקציה זו משתמשים בכוחות שחישבנו בפונקציה הקודמת, מחשבים תאוצה ושינוי במהירות של כל כוכב ובהתאם לכך גם את השינוי במיקום שלו. עובדים ביחידות של מטר לשנייה ובסוף ממירים חזרה את המיקום לשנות אור, ואת המהירות לשנות אור ל- timestep.
בנוסף בפונקציה זו אנו מטפלים בכוכבים שיצאו מהתחום המוגדר, כוכב שיצא מ"קיר" מסוים יכנס חזרה מ"הקיר" הנגדי.
- **MPI Allgather** – לאחר החישוב שוב פעם משתפים את המידע החדש עם כל שאר התהליכים כך שכל תהליך ידע את המיקום החדש של כל הכוכבים, דבר אשר הוא יצטרך באיטרציה הבאה.
- בנוסף בתחילת, אמצע וסוף האלגוריתם אנו שומרים את המיקום של כל הכוכבים ומוציאים לקובץ טקסט בשביל ה- plot (**saveCoordinatesToFile**).

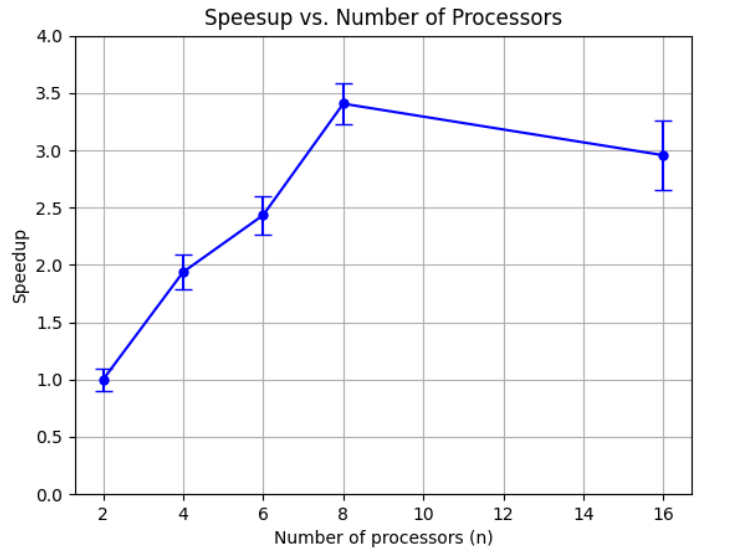
בסוף משחררים את הזיכרון שהוקצה לכל תהליך, מודדים את הזמן שלקח לאלגוריתם וסוגרים את התוכנית.

- הערה לגבי צילומי המסך של הגלקסיה – כיוון שיש הרבה כוכבים אנו רואים בתמונות אי סדר, אך באופן כללי לאחר מעקב עם מספר קטן של כוכבים אנו רואים התכנסות שלהם ולאחר מכן שוב פעם פיזור וחוזר חלילה.
- האלגוריתם נבדק גם על מקרי קיצון של שני כוכבים עם מהירות התחלתית 0, כמו שציפינו הכוכבים נמשכו אחד לשני בצורה סימטרית לחלוטין (מסה זהה) תוך כדי החלפות מקומות (הכוכב הימני עבר שמאלה והשמאלי ימינה וכו').

סיכום, תוצאות ומסקנות:

תחילה נציג את גרפי ה- Speed-Up (לפי העבודה רשום למדוד לפי היחס של שני תהליכים- $\frac{t2}{t_p}$),
כאשר הערכים המוצגים בכל נקודה הם ממוצע של 15 הרצות לכל מספר של תהליכים
$(p = 2,4,6,8,16)$.

- קיבלנו את התוצאות מהרצת הקוד על מחשבי האוניברסיטה, ספציפית על מחשב hobbit1.



ניתוח הגרף:

אנו יודעים שיש למכונה 8 ליבות, לכן מצפה לראות זמן קצר יותר של האלגוריתם עד שמגיעים לעבודה עם 8 ליבות, מכאן והלאה נראה ירידה קלה כיוון שהעבודה לא מתחלקת באופן שווה בין כולם (לפי האלגוריתם של המעבד).

החישוב הוא לפי הנוסחה שראינו בהרצאה:

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

כאשר t_s הוא זמן ההרצה אשר מדדנו עבור $n = 2$ (שני תהליכים כפי שכתוב בהנחיות העבודה).
כיוון שיש למכונה 8 ליבות, אנו מצפים שערך ה- *Speed-Up* יעלה עד אשר $n = 8$, ומכאן והלאה נקבל שחלוקת העבודה בין הליבות כבר לא שווה ולכן נראה את הגרף יורד.

בתוצאות ניתן לראות אישוש להשערה, עד אשר $n = 8$ אנו רואים עליה בגרף. עליה קצת שונה בכל טווח מספרים אך עדיין עליה.
לערכי n גדולים יותר נוצר מצב שחלק מהליבות צריכות להריץ יותר מתהליך אחד, לכן אנו רואים את הגרף בירידה קלה כאשר n = 16. אנו מניחים שיש עליה גם כיוון שזמן החלפת המידע לוקח יותר זמן עם יותר תהליכים.

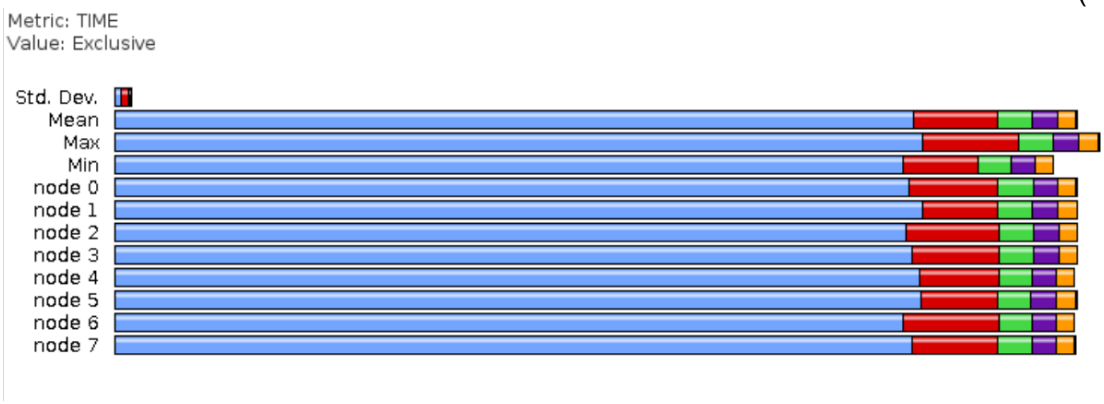
נוכל לשפר את משך זמן התוכנית על ידי שימוש באלגוריתם יותר יעיל לחישוב הכוחות המופעלים על כל כוכב, כגון חישוב כוח המופעל על ידי קבוצה של כוכבים והחלפת המידע הזה בין התהליכים.
כמובן שנקבל trade off בין זמן חישוב הכוחות לבין סנכרון המידע בין התהליכים.

תוצאות עם *Jumpshot* ו-*Scalasca*:

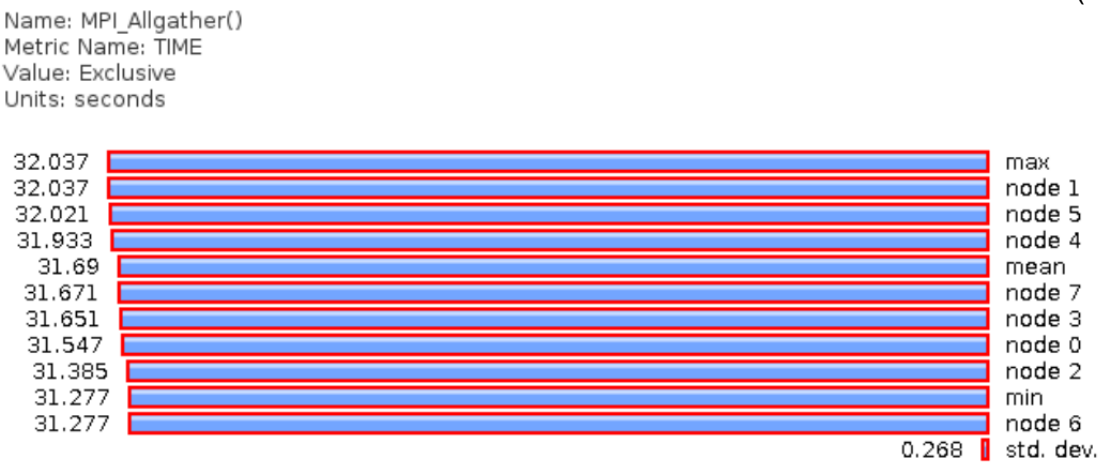
- קיבלנו את התוצאות מהרצת הקוד על ה-*VM* במחשב האישי שלנו.

נציג תוצאות עבור ההרצה כאשר $n = 8$ ונסביר תוצאות אלה.

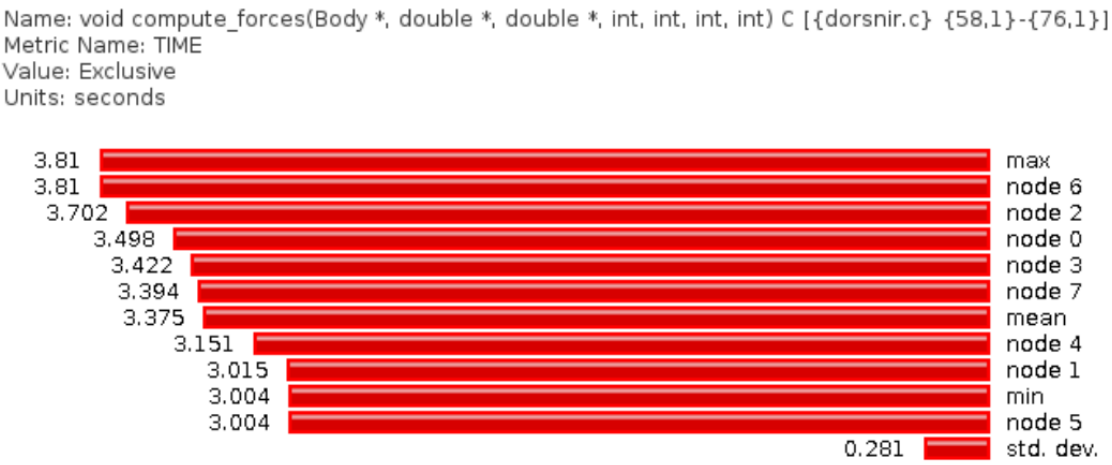
(1)



(2)

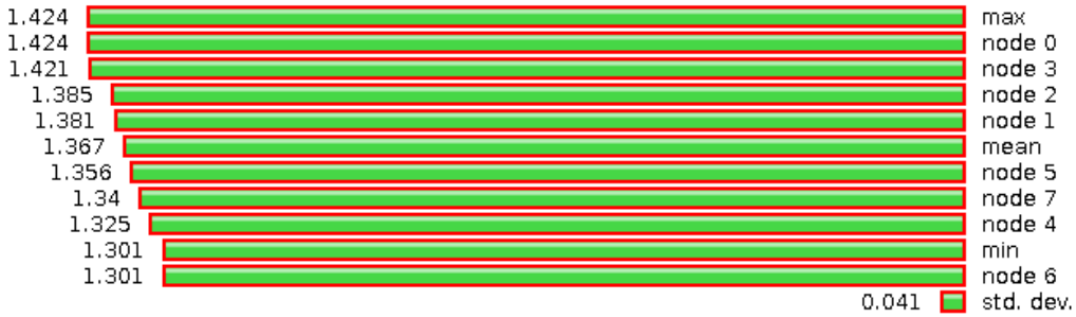


(3)



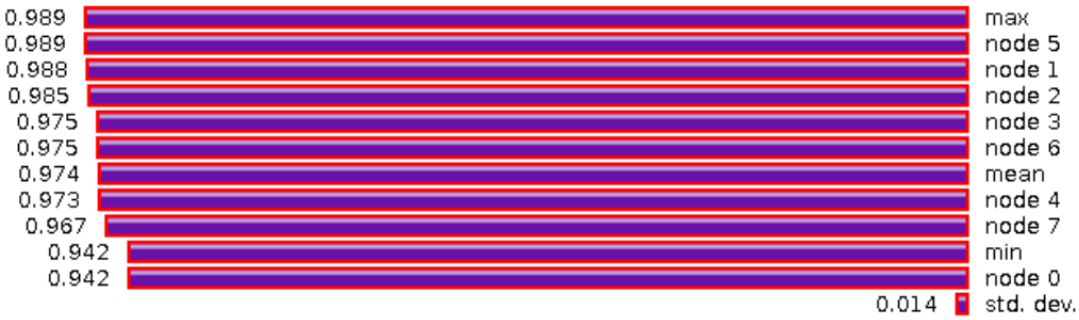
(4)

Name: MPI_Init()
Metric Name: TIME
Value: Exclusive
Units: seconds

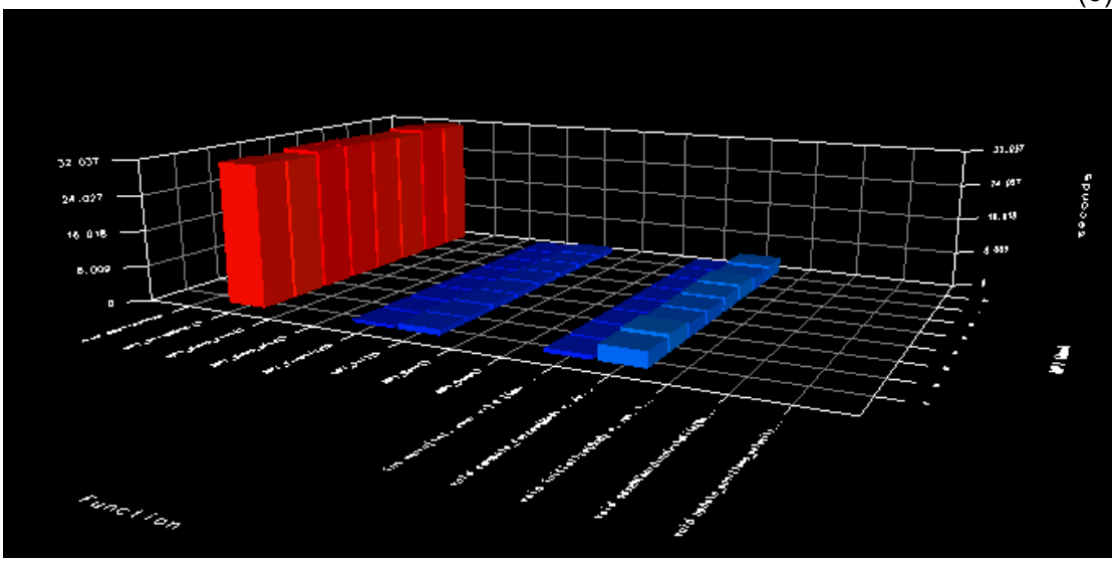


(5)

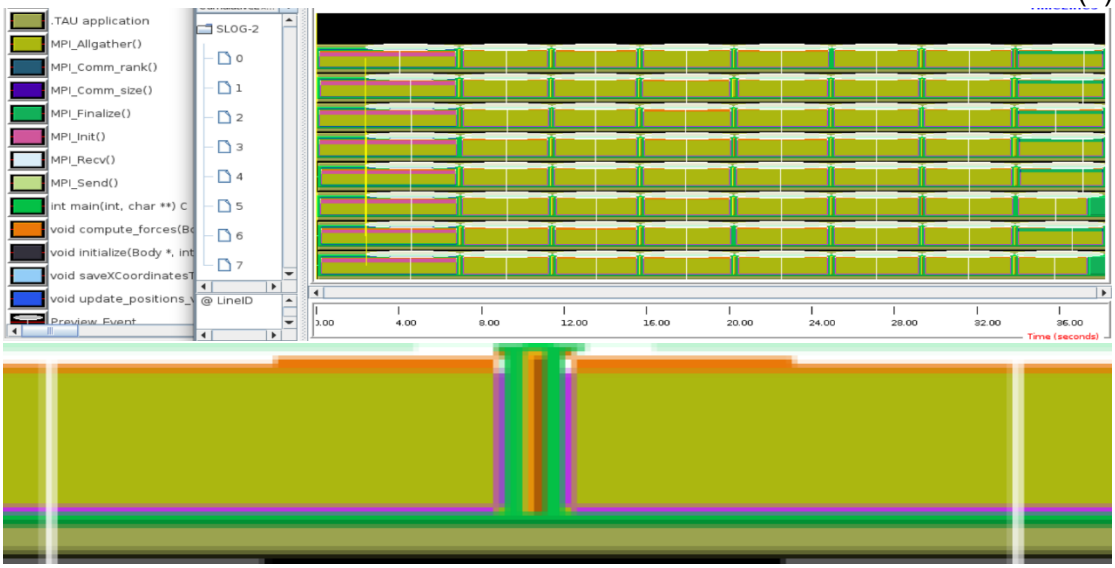
Name: int main(int, char **) C [{dorsnir.c} {121,1}-{206,1}]
Metric Name: TIME
Value: Exclusive
Units: seconds



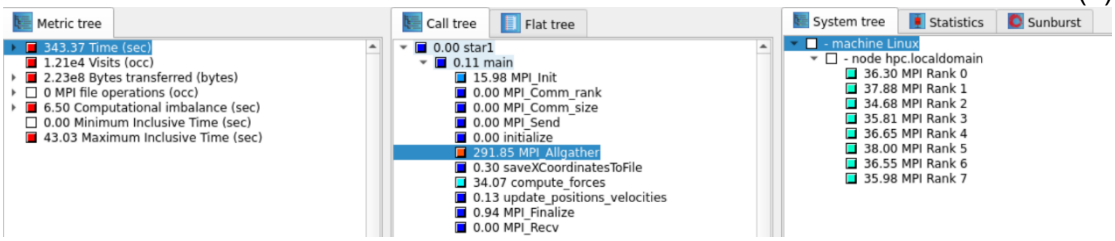
(6)



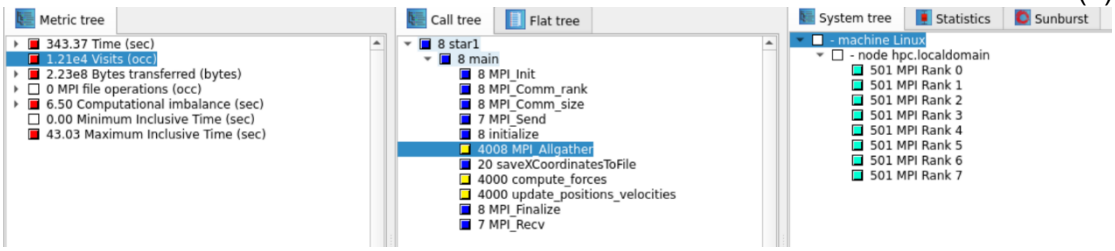
(7)


.TAU application
MPI_Allgather()
MPI_Comm_rank()
MPI_Comm_size()
MPI_Finalize()
MPI_Init()
MPI_Recv()
MPI_Send()
int main(int, char **) C
void compute_forces(Bo
void initialize(Body *, int
void saveXCoordinatesT
void update_positions_V
Preview Event
SLOG-2
0
1
2
3
4
5
6
7
@ LineID
4.00
8.00
12.00
16.00
20.00
24.00
28.00
32.00
36.00
Time (seconds)

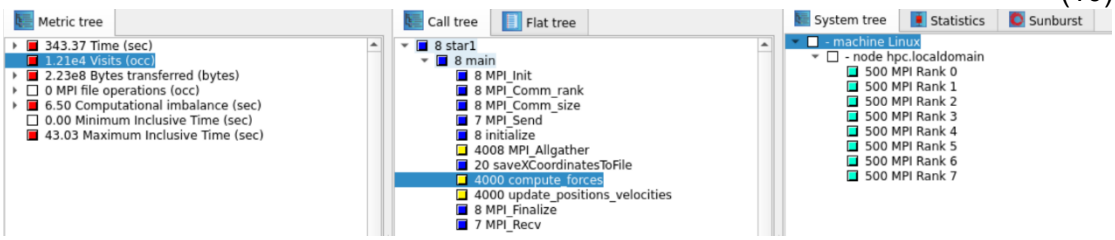
(8)


Metric tree
343.37 Time (sec)
1.21e4 Visits (occ)
2.23e8 Bytes transferred (bytes)
0 MPI file operations (occ)
6.50 Computational imbalance (sec)
0.00 Minimum Inclusive Time (sec)
43.03 Maximum Inclusive Time (sec)
Call tree
Flat tree
0.00 star1
0.11 main
15.98 MPI_Init
0.00 MPI_Comm_rank
0.00 MPI_Comm_size
0.00 MPI_Send
0.00 initialize
291.85 MPI_Allgather
0.30 saveXCoordinatesToFile
34.07 compute_forces
0.13 update_positions_velocities
0.94 MPI_Finalize
0.00 MPI_Recv
System tree
Statistics
Sunburst
- machine Linux
- node hpc.localdomain
36.30 MPI Rank 0
37.88 MPI Rank 1
34.68 MPI Rank 2
35.81 MPI Rank 3
36.65 MPI Rank 4
38.00 MPI Rank 5
36.55 MPI Rank 6
35.98 MPI Rank 7

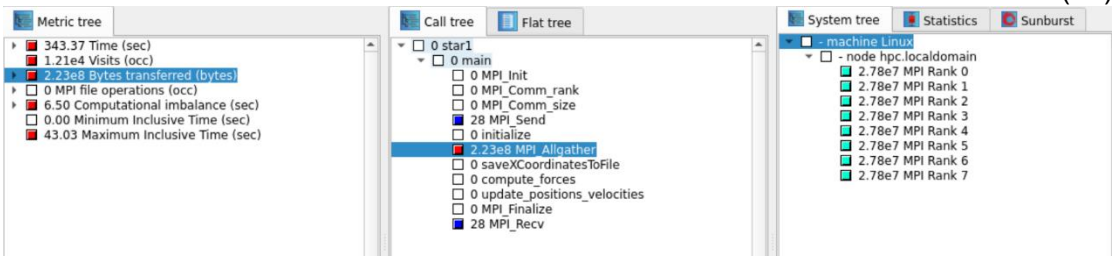
(9)


Metric tree
343.37 Time (sec)
1.21e4 Visits (occ)
2.23e8 Bytes transferred (bytes)
0 MPI file operations (occ)
6.50 Computational imbalance (sec)
0.00 Minimum Inclusive Time (sec)
43.03 Maximum Inclusive Time (sec)
Call tree
Flat tree
8 star1
8 main
8 MPI_Init
8 MPI_Comm_rank
8 MPI_Comm_size
7 MPI_Send
8 initialize
4008 MPI_Allgather
20 saveXCoordinatesToFile
4000 compute_forces
4000 update_positions_velocities
8 MPI_Finalize
7 MPI_Recv
System tree
Statistics
Sunburst
- machine Linux
- node hpc.localdomain
501 MPI Rank 0
501 MPI Rank 1
501 MPI Rank 2
501 MPI Rank 3
501 MPI Rank 4
501 MPI Rank 5
501 MPI Rank 6
501 MPI Rank 7

(10)


Metric tree
343.37 Time (sec)
1.21e4 Visits (occ)
2.23e8 Bytes transferred (bytes)
0 MPI file operations (occ)
6.50 Computational imbalance (sec)
0.00 Minimum Inclusive Time (sec)
43.03 Maximum Inclusive Time (sec)
Call tree
Flat tree
8 star1
8 main
8 MPI_Init
8 MPI_Comm_rank
8 MPI_Comm_size
7 MPI_Send
8 initialize
4008 MPI_Allgather
20 saveXCoordinatesToFile
4000 compute_forces
4000 update_positions_velocities
8 MPI_Finalize
7 MPI_Recv
System tree
Statistics
Sunburst
- machine Linux
- node hpc.localdomain
500 MPI Rank 0
500 MPI Rank 1
500 MPI Rank 2
500 MPI Rank 3
500 MPI Rank 4
500 MPI Rank 5
500 MPI Rank 6
500 MPI Rank 7

(11)


Metric tree
343.37 Time (sec)
1.21e4 Visits (occ)
2.23e8 Bytes transferred (bytes)
0 MPI file operations (occ)
6.50 Computational imbalance (sec)
0.00 Minimum Inclusive Time (sec)
43.03 Maximum Inclusive Time (sec)
Call tree
Flat tree
0 star1
0 main
0 MPI_Init
0 MPI_Comm_rank
0 MPI_Comm_size
28 MPI_Send
0 initialize
2.23e8 MPI_Allgather
0 saveXCoordinatesToFile
0 compute_forces
0 update_positions_velocities
0 MPI_Finalize
28 MPI_Recv
System tree
Statistics
Sunburst
- machine Linux
- node hpc.localdomain
2.78e7 MPI Rank 0
2.78e7 MPI Rank 1
2.78e7 MPI Rank 2
2.78e7 MPI Rank 3
2.78e7 MPI Rank 4
2.78e7 MPI Rank 5
2.78e7 MPI Rank 6
2.78e7 MPI Rank 7

נסביר את התמונות והתרשימים:

(1) בתרשים זה ניתן לראות זמני הפונקציות של כל תהליך בנפרד.
הצבע הכחול הוא של פונקציית *MPI_Allgather* והיא לוקחת את רוב זמן התוכנית, זה קורה כיוון שפונקציה זה משמשת כמעין *barrier*, כלומר תהליכים לא יוכלו להתקדם הלאה עד שהתהליך האחרון יגיע והמידע לא יהיה מסונכרן אצל כולם, לכן היא לוקחת הכי הרבה זמן.
הצבע האדום הוא של פונקציית *compute_forces* והיא לוקחת פרק זמן יותר קטן אך עדיין יחסית ארוך כיוון שכאן בכל שלב כל תהליך עובד בסיבוכיות של $o(n^2)$, הוא צריך לחשב את הכוח שכל כוכב מפעיל.
הצבע הירוק הוא של פונקציית *MPI_Init* והיא לוקחת פרק זמן קטן יותר.הצבע הסגול הוא של פונקציית *main* והיא לוקחת זמן מועט (פונקציית *MPI_Allgather* שנמצאת בתוכה מחושבת בנפרד).
הצבע הכתום הוא של פונקציית *MPI_Finalize* ומשך הזמן שלה הוא המועט ביותר.ניתן לראות בתמונה זו את הזמן הכולל של כל תהליך, את התהליך עם הזמן המקסימלי, המינימלי והממוצע.

(2) בתמונות 2,3,4,5 ניתן לראות את אותו פירוט עבור כל פונקציה בנפרד, ניתן לראות את משך הזמן, זמן מקסימלי, מינימלי וממוצע של כולם עבור פונקציה זו ספציפית.

(6) תרשים זה דומה לתרשים הקודם, כאשר כאן המידע מופיע בתלת מימד.
הציר התחתון הימני הוא מספר התהליך, הציר בתחתון השמאלי הוא שם הפונקציה והציר האנכי הוא משך הזמן שלקח לכל פונקציה, כך שמדרגה גבוהה יותר מסמנת זמן ארוך יותר.
המדרגה האדומה זוהי פונקציית ה- MPI_Allgather וניתן לראות שהיא אכן לוקחת הכי הרבה זמן.
המדרגה בצבע כחול בהיר היא פונקציית *compute_force* ומשך הזמן שלה הוא השני הכי גדול. שאר המדרגות בצבע כחול הן הפונקציות: *main, MPI_Init, MPI_Finalize*, והזמן שלהן קטן קצת בהתאם.
ניתן לראות שהפונקציה שמעדכנת את מיקום הכוכב לקחה מעט מאד זמן כך שלא מופיע לה מדרגה כלל.

(7) על ידי שימוש ב- *Jumpshot* ניתן לראות את מהלך העבודה של האלגוריתם. אנו רואים תחילה את משך הזמן של פונקציית *MPI_Init*, לאחר מכן אנו רואים באופן מחזורי את פונקציות *MPI_Allgather, main* פועלות כמעט בסנכרון מושלם בין התהליכים, כיוון שהתהליכים צריכים לחכות עד לאחרון כדי להשלים את הפעולה ולהתקדם הלאה.
כאשר עושים זום אין ניתן לראות את משך זמן כל הפונקציות באלגוריתם, גם את הקצרות ביותר.
ניתן לראות בהתאם לתרשימים הקודמים את משך הזמן של כל פונקציה ואת סדר פעולתם בקוד.

- בתמונות מכאן והלאה, כיוון שהרצנו את האלגוריתם על ה-*vm* ולא על ה-*hobbit* אז הורדנו בהרבה את המספר הכולל של *time_steps* כדי שהריצה תימשך כ-2 דקות.

(8) בתמונה זו ניתן לראות את משך הזמן שכל פונקציה פעלה, כאן אנו רואים את משך הזמן הכולל של פונקציית *MPI_Allgather* וגם את הזמן שכל תהליך בפרט השתמש בפונקציה זו.

(9) בתמונה זו ניתן לראות את מספר הפעמים שכל פונקציה פעלה, כאן אנו רואים את המספר הכולל של פונקציית *MPI_Allgather* וגם את מספר הפעמים שכל תהליך בפרט השתמש בפונקציה זו.

(10) בתמונה זו רואים את מספר הפעמים בהם התוכנית קראה לפונקציית compute_forces, המספר הכולל ושל כל תהליך בפרט.

(11) בתמונה זו ניתן לראות את המספר הכולל של בתים שנשלחו בתוכנית.
יש פירוט על מספר הבתים שנשלחו על ידי כל פונקציה ועל ידי כל תהליך.

שאלות רב ברירה:

1. מהי המטרה של הפונקציה *MPI_Allgather* ב-*MPI*?
   א. היא אוספת נתונים מכל התהליכים ומפיצה אותם לכל התהליכים.
   ב. הוא אוסף נתונים מכל התהליכים ושולח אותם לתהליך בודד.
   ג. הוא מפזר נתונים מתהליך בודד לכל התהליכים.
   ד. הוא משדר נתונים מתהליך בודד לכל התהליכים.

2. בהקשר של מחשוב מקביל באמצעות *MPI*, למה מתייחס המונח "*barrier*"?
   א. מבנה נתונים המשמש לאחסון ושיתוף נתונים בין תהליכים בתוכנית מקבילה.
   ב. נקודת סנכרון בה כל התהליכים ממתינים עד שהם מגיעים לאותה נקודה בקוד.
   ג. דפוס תקשורת שבו תהליכים מחליפים מסרים זה עם זה בסדר מוגדר מראש.
   ד. טכניקה לאיזון עומסים במחשוב מקביל לפיזור עבודה שווה בין תהליכים.

פתרון: שאלה 1, תשובה א'
שאלה 2, תשובה ב'