



המחלקה להנדסת תוכנה

סימולציה של דינמיקה מולקולרית על גביו מעבד גרפי

חיבור זה מהוות חלק מהדרישות לקבלת
תואר ראשון בהנדסה

מאה
ודים קורס

יולי 2010

אב תש"ע



המחלקה להנדסת תוכנה

סימולציה של דינמיקה מולקולרית על גבי מעבד גרפי

חיבור זה מהוות חלק מהדרישות לקבלת
תואר ראשון בהנדסה

מאה
ודים קורס

מנחה: ד"ר יהודה חסין

תאריך:

אישור המנחה:

תאריך:

אישור מרכז הפרויקטים:

תקציר

מטרת הפרויקט היא ראשית להציג את הבסיס והוא דינמיקה מולקולרית - מהי דינמיקה מולקולרית, מה היא מאפשרת ואייר היא עובדת. שנייה, להציג את בעיית הסימולציה של דינמיקה מולקולרית – המבנה של הסימולציה, חישוב כוחות אינטגרציה. ווסףית להציג את הבעיה במימוש סימולציה של דינמיקה מולקולרית כאשר נתיחה למימוש רגיל ופושט לפי הרעיון הבסיסי, מימוש יעיל בעזרת מבני נתונים, מימוש מקבילי באופן כללי, ומימוש מקבילי ייעיל על גבי GPU שהוא הרחבה של המימוש המקבילי הכללי.

במימוש הסימולציה נעשה שימוש בשני מבני נתונים במטרה לשפר את זמן הריצה, הראשון הוא רשימת שכניות, והשני הוא Spatial Hash. המימוש מציע שני שיטות לעובדה מקבילתית על גבי GPU, כאשר לכל שיטה יש את היתרון והחסרונות שלה – יתרונות וחסרונות אלו מתוארים בעזרת גרפים.

הסימולציה מומשה בשפת C/C++, והשימוש ל-GPU נעשה באמצעות ארכיטקטורת CUDA, שהיא ארכיטקטורה ייעודית לחישובים בצורה מקבילה.

נעשו בדיקות של המימוש והשוואות בין גרסאות הסימולציה (גרסה CPU וגרסה GPU). הבדיקות לגראסת GeForce 285GTX מס' 7 Core Intel ובדיקות ל-GPU נעשו על כרטיסי מס' NVIDIA GeForce 9600GT ו-NVIDIA GeForce 9600GT של חברת NVIDIA.

בנוסף, נשתה השוואת מול גרסה בסיסית של סימולציה שנכתבה ע"י כימאים וזמן ריצה של הסימולציה מגע לימים ואף שבועות – נראה אין השימוש במבנה הנתונים, ושימוש ב-GPU, משפר את זמן הריצה של עד פי-146 בצד, ופי יותר מ-970 בצד.

הצהרה

העבודה נעשתה בהנחיית ד"ר יהודה חסין,
המכללה האקדמית להנדסה ירושלים – המחלקה להנדסת תוכנה.

החיבור מציג את עבודותי האישית
ומהוות חלק מהדרישות לקבלת תואר ראשון בהנדסה.

תודות

ברצוני להודות לד"ר יהודה חסין, על העזרה הרבה וההכוונה
בפרויקט, על העצות, הרעיונות והזמן.

ותודה רבה לפרופסור תמר רץ על ההכוונה בנושא הדינמיקה
המולקולרית.

תוכן עניינים

6.....	מסגרת הפרויקט.....	1
6.....	סימולציה של דינמיקה מולקולרית.....	1.1
7.....	מוניחים והגדרות.....	2
7.....	מוניחים.....	2.1
7.....	הגדרות.....	2.2
8.....	תיאור הבעיה.....	3
8.....	הבעיה.....	3.1
8.....	הבעיה מבחינה הנדסית.....	3.2
8.....	שימוש הסימולציה.....	3.2.1
8.....	שימוש הסימולציה בצורה עיליה.....	3.2.2
9.....	שימוש הסימולציה בצורה מקבילה.....	3.2.3
9.....	שימוש הסימולציה בצורה עיליה על גבי המעבד הגרפי.....	3.2.4
10.....	תיאור הפתרון.....	4
10.....	פתרון בסיסי.....	4.1
10.....	מבנה נתונים.....	4.2
10.....	רשימת שכניות.....	4.2.1
13.....	מבנה Spatial Hash.....	4.2.2
14.....	שימוש הסימולציה.....	4.3
18.....	תיאור מערכת שמומשה.....	5
18.....	גרסת CPU.....	5.1
18.....	גרסת GPU.....	5.2
19.....	Equilibrate.....	5.3
21.....	ממשק משתמש.....	5.4
22.....	בדיקות.....	6
22.....	בדיקות שימוש התוכנה.....	6.1
22.....	בדיקות המערכת ומסקנות.....	6.2
22.....	השוואה מול הגרסה המקורית שנכתבה ע"י כימאים.....	6.2.1
23.....	השוואה של TPP,BPP,CPU בגז.....	6.2.2
24.....	השוואה של TPP,BPP,CPU בצבר ביןוי.....	6.2.3
24.....	השוואה של TPP,BPP,CPU בצבר חזק.....	6.2.4
24.....	השוואה של TPP על GeForce 9600GT מול TPP על GeForce 285GTX.....	6.2.5
25.....	השפעת רשימת השכניות בגז.....	6.2.6
25.....	השפעת רשימת השכניות בצבר.....	6.2.7
25.....	התפלגות זמינים בין גז לצבר.....	6.2.8

26.....	השוואה לפתרונות בספרות.....	7
27.....	רשימת ספרות.....	8
28.....	נספחים.....	9
28.....	תרשימים מודולים של הסימולציה	9.1
29.....	תרשימים Data Flow של הסימולציה	9.2
30.....	תרשימים מחלקות.....	9.3
30.....	תרשימים מחלקות מפענה קבצי הגדרות.....	9.3.1
30.....	תרשימים מחלקות ממשק תלת מימד מעל OpenGL	9.3.2
31.....	תרשימים מחלקות כללי ממשק משתמש	9.3.3
32.....	GPU	9.4
33.....	CUDA	9.5
36.....	גרף 1 – מגרטת היכאים ועד ל-GPU.....	9.6
36.....	גרף 2 - TPP vs. CPU - גז	9.7
37.....	גרף 3 - TPP - גז	9.8
37.....	גרף 4 - BPP vs. TPP vs. CPU – צבר ביוני	9.9
38.....	גרף 5 - CPU – BPP vs. TPP vs. GPU – צבר חזק	9.10
38.....	גרף 6 – GeForce 285GTX vs. GeForce 9600GT – גז	9.11
39.....	גרף 7 – השפעת רשימת השכניות בגז	9.12
39.....	גרף 8 – השפעת רשימת השכניות בצבר ביוני	9.13
40.....	גרף 9 - פילוג זמנים בגז	9.14
40.....	גרף 10 - פילוג זמנים בצבר חזק	9.15
41.....	צלומי מסך של המשק	9.16
41.....	תצלום מסך 1 – בזמן ריצה של הסימולציה	9.16.1
41.....	תצלום מסך 2 – תצוגה גרפית	9.16.2
42.....	תצלום מסך 2 – תצוגה תלת-ממדית	9.16.3

1 מסגרת הפרויקט

1.1 סימולציה של דינמיקה מולקולרית

אנו מביצים סימולציות – סימולציות מחשב – במטרה להבין את המאפיינים של המכלולות מבחינת המבנה ואת קשרי הגומלין ביניהם. זה משמש לנו כהשלה לניסויים קונבנציונליים, ומאפשר לנו למדוד תכונות חדשות שלא ניתן לגלוות בדרךים אחרות.

ישנו שתי שיטות מרכזיות להדמיה מולקולרית – דינמיקה מולקולרית (MD) ומונטה קרלו (MC), כאשר ישנו טכניקות נוספות נוספות המשלבות את שתי השיטות.

מהי דינמיקה מולקולרית –

דינמיקה של התנגשויות מולקולריות עוסקת בחקר המנגנונים של תהליכי אלמנטריים (כימיים ופיזיקליים) במטרה להבין מה מתתרחש ברמה המולקולרית כאשר תהליך כימי או פיזיקלי יוצא לפועל.

באמצעות דינמיקה מולקולרית ניתן לגלוות מנגנוני ריאקציות, ואף להבין מהם התנאים בהם יתרחש תהליך. ניתן להשתמש בדינמיקה מולקולרית לתהליכים בפזה גזית, לתהליכים המתרחשים בתמיסות, לתהליכים המתרחשים על-פני משטחים, ועוד. דינמיקה מולקולרית מיושמת גם לתהליכים גם לינאים חיובים ושליליים, כאן כאשר הם מבודדים והן כאשר הם מומסים בתמיסה.

דינמיקה מולקולרית מספקת את ההסברים הבסיסיים של מערכות ריאקטיביות רבות: מתהליכים המתרחשים באטמוספירה ועד תהליכי שריפה, קטליזה ופעולות אנזימטיות. במקרה אחר, הדינמיקה המולקולרית נותנת כלים להבנת כל ענפיו השונים של הכימיה.

במקרה שלנו, במדחפים של מטוסי קרב מתרחשות ריאקציות כימיות על גבי הלהבים – אנו מעוניינים ללמוד מה מתתרחש ואיזה ריאקציות מתרחשות בין להב המנווע שמתמקד מהר למכלולות האויר (בעיקר חמוץ וחנקן).

2 מונחים והגדרות

2.1 מונחים

- גז – אטומים לא דחוסים.
- צבר – הפרק מגז, אטומים דחוסים.
- תא – איבר תלת ממדי בחלוקת תלת ממדית של הקופסה של רשימת השכניות.
- Boxland, Boxl – הקופסה התלת ממדית בסימולציה (לא מדובר בקופסה של רשימת השכניות).
- Block Per Particle – BPP
- Bucket – Bucket – תא בעمرך של ה-Spatial Hash.
- Cell – תא.
- Context Switching – מעבר בין עבודה ב-CPU, לעבודה ב-GPU.
- Kernel – קוד הרץ על ה-GPU.
- Lennard Jones – חישוב לפי פוטנציאל לנרד-ג'ונס.
- Many-Body – חישוב כוחות בין מספר אטומים גדול מ-2.
- Nitrogen – N – חנקן.
- Noble Gas – NG – גז אציל (ניואון, ארגון וכו..)
- Oxygen – O – חמצן.
- Timestep – איטרציה בסימולציה.
- Thread Per Particle – TPP

2.2 הגדרות

- יחידות הזמן בסימולציה נמדדות ב-fs, או בקיצור 10^{-15} sec .
- יחידות המרחק בסימולציה נמדדות ב-Å, ångström $100.00 \times 10^{-12} \text{ meter}$.
- בסימולציה היחס לאטום הוא לחלייק.
- יחידות הזמן במדידת זמן הריצה של הסימולציה נמדדות במילישניות.

3 תיאור הבעיה

3.1 הבעיה

динמיקה מולקולרית מורכבת מפרטן צעד-צעד נומי של הנוסחאות הקלאליסיות של תנועה - קלומר נוסחות לחישוב כוחות, תאוצה, מהירות, מיקום ועוד.

את החישובים העיקריים בסימולציה אפשר לפרק לשני חלקים:

- חישוב כוחות – נגזרת של הפוטנציאלי ($\text{פוטנציאל לנרד-ג'ונס או פוטנציאל רב-גופי}$). במקרה של פוטנציאלי לנרד-ג'ונס, החישוב נעשה בין כל הזוגות, ובמקרה של פוטנציאלי רב-גופי, החישוב נעשה בין שלושה אטומים (במקרה שלם).
- אינטגרציה – חישוב המהירות והמיקום לפי התאוצה שהתקבל מהנוסחה $\frac{F}{m} = a$, כאשר מהירות היא אינטגרציה של התאוצה ($\int a dt = v$) והמיקום הוא אינטגרציה של המהירות ($\int v dt = x$).

3.2 הבעיה מבחינה הנדסית

3.2.1 מימוש הסימולציה

כאשר מתכוונים את הסימולציה צריך לזכור את כוח החישוב שיש לנו – כך שגודל הסימולציה יהיה תואם לכוח החישוב (גודל סימולציה – כמות חלקיקים וכמוות הצעדים\זמן שאנו רוצים לדמות) וזאת ב כדי זמן הריצה יהיה סביר. למחרות זאת, הסימולציה צריכה להיות מספק ארוכה (בכמויות הצעדים או הזמן) ב כדי שתהיה רלוונטית למרוחק הזמן המציאוטי.

3.2.2 מימוש הסימולציה בזרה עילית

הסימולציה שקיימת היום מומשה בזרה לא עילית לפי מבנה פשוט של נוסחות בלבד ללא שימוש במבני נתונים ואלגוריתמים. המימוש נעשה ע"י כימאים.

לכן קודם כל, יש למש את הסימולציה בזרה עילית על גבי המעבד הכללי, תוך כדי שימוש במבני נתונים ואלגוריתמים עיליים.

בדינמיקה מולקולרית – החלק היכי חשוב והיכי כבד מבחינות חישוב הוא חישוב הכוחות. ביום הסימולציה בזמן חישוב הכוחות עוברת על כל הזוגות האפשרים של האטומים, ומכיון ששיעור הכוחות תלוי במרחב בין זוג האטומים, והכוח שימושי בין שני אטומים קטן מאוד מרחקם, אנו נרצה למש מבנה נתונים שיאפשר לנו לבצע חישוב כוחות רק בין אטומים קרובים.

3.2.3 *הימוש הסימולציה בצורה מקבילה*

המעבר בין חישוב בצורה טורית לחישוב בצורה מקבילה הוא מעבר לא קל – צריך לתכנן כל חלק וחלק של הסימולציה בצורה נcona על מנת למנוע מצבים של שימוש במשאים משותפים, שיתוף מידע נכון בין התהיליכונים הרצים במקביל (אם יש חישובים שתלויים בחישובים אחרים – OCDI שהחישובים אלו יהיו נconaים צריך שהמידע יהיה עדכני לגבי כל התהיליכונים) ולשמור על סyncron בנקודות קritisיות של שיתוף מידע כך שלא יקרה מצב שתהיליכון אחד סיים וטהיליכון אחר רק התחיל את העבודה.

3.2.4 *השימוש בסימולציה בצורה עיליה על גבי המעבד הגרפי*

על מנת שאכן הסימולציה תעבור מהר יותר על גבי המעבד הגרפי – צריך לא רק לתכנן אותה בצורה מקבילה, אלא גם בצורה עיליה – לחסוך גישה לחיצון גלובלי, ולנסות להשתמש כמה שיותר באוגרים ובזיכרון המשותף.

בנוסף לנסה להשתמש במבני נתונים יעילים על הארכיטקטורה של CUDA.

4. תיאור הפתרון

4.1 פתרון בסיסי

הפתרון הבסיסי והפשוט למימוש סימולציה הוא:

1. אתחול מקומות התחלתיים ומהירות התחלתיות של האטומים.
2. חישוב כוחות בין כל אטומים: $F = \frac{F}{M} = a = (r)VV$ – כאשר הכוח הוא נגזרת הפוטנציאלי וממנו מתקבלת התאוצה.
3. הזזה של האטומים למקום החדש לפי אינטגרציה.
4. הזזה של הזמן קדימה: $\Delta t = t + t$
5. יש לחזור על צעדים 4-2 נוצרה.

פתרון זה הוא פתרון פשוט ואינו יעיל – במקרה שלנו, יש לנו שני סוגי חישוב כוחות, אחד עבור גז אציל – חישוב מסוג לנרד-ג'ונס – שהוא חישוב בין זוגות אטומים - $(a^2)0$, וחישוב רב-גופי בין אטומי חמצן וחנקן - $(a^3)0$. אנו נשפר את זמן הריצה באמצעות מבני נתונים המפורטים בהמשך.

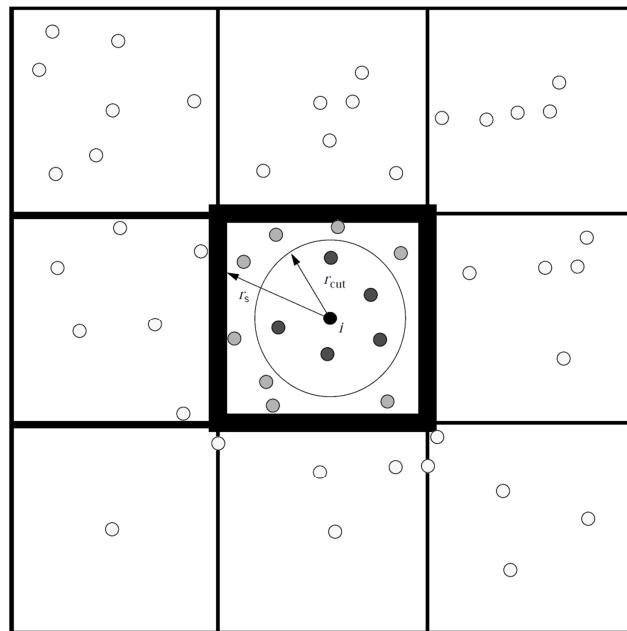
4.2 מבני נתונים

4.2.1 רישימת שכניות

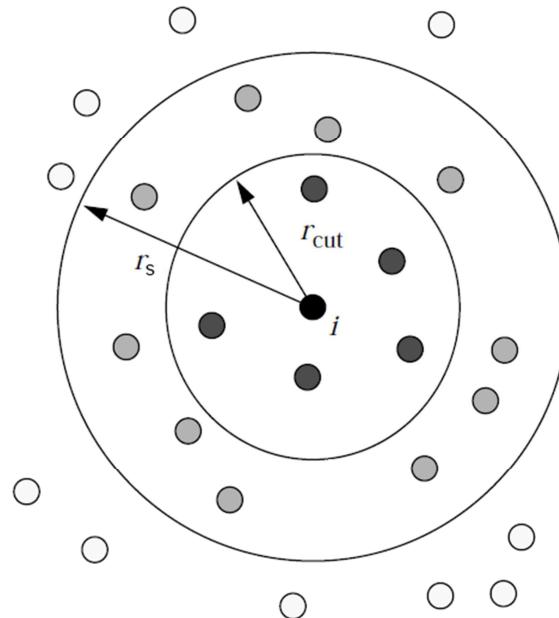
הכוח בין אטום לאטום תלוי במרחק ביניהם, אך חישוב כוחות בין אטומים שהמרחק שלהם גדול מהסף, זה חוסר עילוות – לכן צריך למשתמש במבנה נתונים חכם שיחזיק את כל האטומים הקרובים (השכנים) של כל אטום. בניית רישימת השכנותיות תבוצע כך: נחלק את המרחב לתאים בגודל מוגדר RS, ונכניס את האטומים הנמצאים בכל תא לרישומה של התא. לאחר מכן עבור כל אטום נבנה רשימה שמכילה את האטומים באותו תא, והאטומים הנמצאים בתאים השכנים לתא.

21	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

הסיבה שצריך גם את התאים מסביב היא שאם אטום נמצא בקצה של התא, ולא נכליל את השכנים מסביב לתא, החישוב כוחות יהיה שגוי כי יכול להיות שיש לאטום שכנים קרובים, אך בכלל שהם מצויים בתא אחר, הם לא יכנסו לחישוב.



בזמן הריצה – עבור כל אטום הנמצא בראשימה נבדוק אם המרחק בין לאטום הנוכחי קטן מ- r_s .
הסוף או המרחק המקסימלי בין זוג אטומים זה r_{cut} .



גודל התא בראשימה יהיה $X + RS$, כאשר X הוא מרחק כלשהו נוסף.

אנו נבחר מספר גדול M -CUT כי האטומים תמיד נמצאים בתזוזה ואנו לא מעוניינים לחשב כל פעם מחדש את רשימת השכניות. לכן ה-X הוא המרחק הנוסף לסוף על מנת שם האטום בהרצאה הקודמת לא היה קרוב לאטום אחר, אך לאחר כמה הריצות הוא התקדם לכיוון האטום, אנו נכלול אותו בחישוב.

את רשימת השכניות צריך לחשב מחדש אם יש אטום שהמרחק שזז גדול מ- $RS - RCUT$, אך מכיוון שתמיד קיים אטום נוסף שיכול לזרז אותו המרחק לכיוון האטום, יש לחלק את $RS - RCUT$ בשתיים. לכן את רשימת השכניות יש לחשב מחדש כל פעם שקיים אטום שזז מרחק גדול מ- $\frac{RS - RCUT}{2}$.

על מנת לא לבצע את החישוב הנ"ל כל הזמן, לאחר בניית רשימת השכניות אנו נבצע חישוש של מהירות ה- C י גבואה מכל מהירות, ולפי הנוסחה $vt = x$ אנו נבודד את t , ונחשב את ה-time step שבו צריך לבנות מחדש את רשימת השכניות.

$$x = vt \rightarrow t = \frac{x}{v} = \frac{\frac{RS - RCUT}{2}}{v} = \frac{RS - RCUT}{2v}$$

$$\rightarrow \text{list-regeneration-timestep} = \frac{t}{dt}$$

רשימת השכניות מורידה את סיבוכיות הזמן מ- $O(n^2)$ ל- $O(n * x)$, כאשר $x < n$ - וכך אפשר להתייחס אל x קבוע (כי לא צריך לעבור על כל האטומים אלא על מספר קבוע קטן בהרבה מכמות האטומים - למשל 500 מול 500).

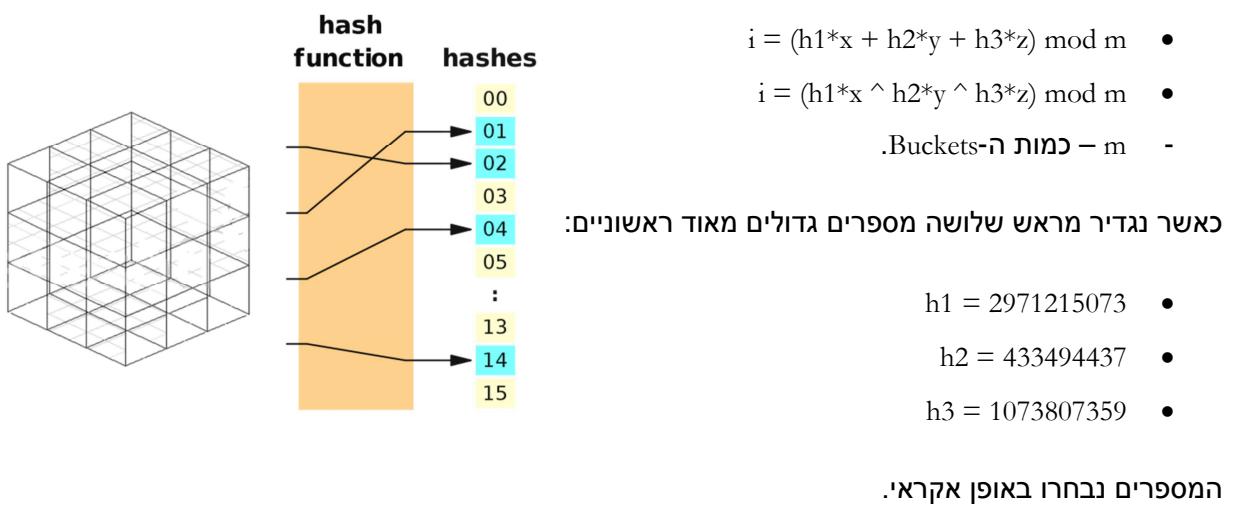
4.2.2 Spatial Hash

הבעיה ברשימת השכניות היא שבזמן אתחול הרשימה יש צורך לאפס את כל התאים, ואם יש לנו למשל 500 אטומים ו-1000000 תאים, נדרש לעבור על כל ה-1000000 תאים כאשר לכל היותר בשימוש יהיו לנו 500 תאים, ולכן אנו נרצה למנוע את הטיפול בתאים הריקים – ובשביל זה יש את Spatial Hash.

בහינתן קואורדינטות תלת-ממדיות של אטום, ייחזר לנו את ה-Bucket של התא שמשויר לקואורדינטות.

בגלל שיש לנו פחות Buckets מאשר כמות התאים, יכולים להיות התנגשויות, אך על מנת לטפל בהתנגשויות, נבצע רעיון דומה לרעיון של Cuckoo Hashing [5], משתמש בשתי פונקציות Hash, ואיתרציה.

ב-Spatial Hash בSIMD נעשה שימוש בשתי הפונקציות הבאות:



אם לאחר הפונקציה הראשונה תהיה התנגשות עם תא קיים, נפעיל פונקציית Hash נוספת לחישוב התא. אם בכל זאת תהיה התנגשות, נבצע איתרציה (ריצה על גבי המערך $-1+1$) עד שנגיע לתא שאינו תפוס.

- Spatial Hash לנו מה נותן איזה.

- שימוש ב-Spatial Hash מאפשר להריצה להיות תלוי בכמות האטומים ולא בכמות התאים.
- ובנוסף בהינתן הקואורדינטות של האטום, Spatial Hash מאפשר לנו לגשת לתא בו נמצא האטום בזמן של $O(1)$.

.Bucket בימוש נקראת compute_cell_bucket – בגלל שהוא מקשרת בין תא ל-Bucket.

4.3 מימוש הסימולציה

Pseudocode No' 1:

```

md-main
    for each atom 1
        set initial position from input
        set initial velocity from input
        force 5
        calculate initial acceleration
    set generate-n-list-timestep 2 = 0
    while timestep ≠ target timestep
        if list-regeneration-timestep = timestep 3
            generate_neighbor_list
            list-regeneration-timestep = calculate new timestep
        for each atom
            predict 4
            force 5
            correct 6
            if abs(atom[x,y,z]) > boxl / 2 7
                set positions in the box
                invert velocities
                force
            set new acceleration

```

1. איתחול הנתונים – מקובץ קלט.
2. list-regeneration-timestep – מסמן לנו متى צריך לעשות בנייה מחדש של רשימת השכניםות.
3. בניית רשימת השכניםות (הסביר בהמשך).
4. predict - פונקציה שմבצעת ניחוש של מה שיקרה – איפה החלקיק יהיה, לפי התאוצה והמהירות ברגע זה ולפי הנתונים הקודמים.

למה עושים predict – אנו מעוניינים לחשב את הכוח שייהי ועל מנת לעשות זאת אנו צריכים לדעת איפה כל אטום יהיה – אך איננו יכולים לדעת, אנו יכולים רק לנחש איפה האטום יהיה, לפי ההתנהגות שלו עד עכšíו.

בנוסף – השימוש predict-force-correct זו שיטה שנותנת לנו אינטגרציה יותר מדויקת.
למה צריך אינטגרציה – כי force מחזיר לנו כוח, ואנו צריכים מיקום.

$$a = \frac{F}{m} = x''$$

.5 – הfonקציה hei חסובה בסימולציה, היא מבצעת את חישוב הכוחות.

Pseudocode No' 2:

```

force [atom i]
    for each atom j in the neighbor list of atom i1
        if distance [i,j]<rcut2
            if (i is NG or j is NG)
                perform lennard-jones-forces [i,j]3
            if (otherwise)
                perform many-body-forces [i,j]4

```

.5.1. מכיוון שבחישוב כוחות הכוח דועך עם המרחק, אנו מעוניינים רק באטומים הנמצאים בראשית השכנים של אטום i .

.5.2. רשימת השכניםות (הסביר בהמשך) בונה לנו רשימת שכנות בעלת מרחק של RS , אך לחישוב כוחות אנו מעוניינים במרחב של $RCUT$. (איור 2).

.5.3. חישוב כוחות בין זוג אטומים, לפי פוטנציאלי לנרד-ג'ונס מתקיים רק בין זוגות אטומי גז אציל, או בין אטום גז אציל לאטום חנקן או חמצן.

.5.4. חישוב כוחות Many-Body בין שלשות אטומי חנקן או חמצן, כאשר החישוב זהה מחלוקת לשני חישובים:

- Bond Order 1 – חישוב הכוח המשפיע אטום i על האטומים j .

- Bond Order 2 – חישוב הכוח שמשפיע על אטום k , שהוא השכן של האטומים i, j .

.6. correct – לאחר חישוב הכוחות, הfonקציה מבצעת תיקון של המיקומים, מהירות ותאוצה לפי הכוח שחוشب.

.7. הסימולציה מדמה אוסף של אטומים בתוך קופסה סגורה, לכן אחרי כל ביצוע של correct, צריך לבדוק אם החלקיק לא יצא מהתוכנה, ואם אין אז לבצע תיקון למיקומים ולהפוך את הכיוון של המהירות, לאחר זה לבצע שוב חישוב כוחות בעזרת force כי המרחק בין האטומים השתנה ולעדכן תאוצה לפי החוק התנועה השני של ניוטון – $F = ma$.

מימוש רשיימת השכניות יתבצע לפי הפסאודו-קוד הבא:

Pseudocode No' 3:

```
generate_neighbor_list
    for each atom
        n = compute_cell_bucket [x, y, z] 1
        add atom to cell [n] list 2
    for each atom
        n = compute_cell_bucket [x,y,z]
        nn = cells around cell n 3
        list [atom] = list [n] + lists [nn] 4
```

- .7.1. compute_cell_bucket – פונקציה שmaps את המיקום של האטום במרחב התלת-מימדי לתא במערך.
- .7.2. הוספה של האטום לרשימה שמקושרת לתא זה.
- .7.3. כל התאים מסביב לתא זה, 26 תאים לכל היותר.
- .7.4. הרשימה הסופית של כל אטום היא הרשימה של התא של האטום + הרשימה של התאים מסביב. את הסימולציה בפרויקט נמש בשני גרסאות: גרסת CPU וגרסת GPU.

את גרסת ה-GPU נמש בעזרת CUDA על גבי מעבדי גרפיים (GPU) של חברת NVIDIA. הסבר כללי על GPU ו-CUDA – נספחים 10.4, 10.5 בהתאם.

מימוש הפונקציות על גבי CUDA:

- הפונקציות predict ו-correct הם פונקציות פשוטות, המימוש שלהם הוא אותו מימוש כמו המימוש על גבי ה-CPU. הייעילות של המימוש על גבי GPU תלויה בלבד בכמות האטומים בסימולציה. המימוש יהיהiesel כאשר $\text{Context Switching Time} + \text{Kernel Time} < \text{CPU Time}$ – אך זמן ה- $\text{Context Switching Time}$, אף זמן ה- Kernel Time , גדול מזמן החישוב ב-CPU – אף עדין עדיף להשתמש במימוש על גבי GPU על מנת לא לבלוט את זמן הריצה על העתקה נתונים מה-GPU ל-CPU וזרה. הקונFIGורציה של ה-Kernel היא פשוטה ביותר, כל אטום הוא תהליכון ומモות הבלוקים נקבעת לפי כמו האטומים בסימולציה \ כמו התהליכיונים לבлок.

- מכיוון שרישימת השכניות היא די גדולה, ואי אפשר להעתיקה לזיכרון המשותף, אנו משתמש בזיכרון מיטמן לטקסטורות. זיכרון מיטמן לטקסטורות אינם יותר עיל מהזיכרון המשותף, אך הוא מאפשר גישה יותר מהירה לזיכרון הגלובלי מאשר גישה רגילה דרך מערכיים ומצבייעים.

- אנו משתמש ב-API CUDA Driver, כי הוא מאפשר לנו לבנות את הקונפיגורציה לפני זמן הריצה – מה שיחסוט הרבה זמן ריצה כי אנו מבצעים הרבה קריאות ל-Kernel. (למשל בסימולציה עם 10000 צעדים אנו נבצע לפחות 30000 קריאות).
- בנוסף CUDA Driver קליטת יותרiesel מבחן זמן הריצה.

- את הפקציה force אפשר למש בשני דרכי:

(ההתיחסות פה היא אל שני סוגי החישוב כוחות, גם אל לנרד-ג'נס וגם ל-Body-Body (Many Body))

 1. באמצעותה שיטה כמו המימוש של הפקציות predict ו-correct, כלומר כל אטום הוא תהליכיון, וכל אטום\תהליכיון מכיל לפחות שרצה על כל השכנים של האטומים כמו שמתואר בpseudo-קוד של .Pseudocode No' 2 – force

- לשיטה זו נקרא TPP, Thread Per Particle, כי לכל אטום נקצת תהליכיון.
2. הדרך השנייה היא – כל אטום הוא בлок, וכל שן של האטום הוא תהליכיון בבלוק, דרך זו יותר יעילה עבור חישובים כבדים יותר, אך מגבילה אותו בכמות השכנים לאטום לכמה תהליכיונים שאפשר להפעיל בבלוק שהוא 512 תהליכיונים.
- לאחר שכל התהליכיונים סיימו לחשב את הכוח, התהליכיון הראשון בכל בлок יבצע חיבור של כל תוצאות החישוב עבור הבלוק – שהוא מייצג את האטום שעבורו בוצע חישוב הכוחות.
- לשיטה זו נקרא BPP, Block Per Particle, כי לכל אטום נקצת בлок.

Pseudocode No' 4:

```
force-bpp

atom-index = tex2D [thread-id, block-id]

shared_memory[thread-id] =
    force-based-on-particle-type [block-id, atom-index]

synchronize

if thread-id == 0
    force =  $\sum_{i=0}^n shared\_memory [i]$ 
```

5. תיאור מערכת שטומשה

לשימוש ארבעה רכיבים:

- 7.5. גרסת CPU של הסימולציה.
- 7.6. גרסת GPU של הסימולציה.
- 7.7. Equilibrate – תוכנה לייצרת קונפיגורציות התחלתיות.
- 7.8. ממתק משמש.

5.1 גרסת CPU

הסימולציה מומשא לפי האלגוריתמים והפסאודו-קודים שתוארו. מומשא בצורה של Console Application, עקב זאת שטטרת היא לבצע חישובים מתמטיים ואין נדרש ממתק גרפי במקורה הזה (הממתק הגרפי נבנה בנפרד).

קיים קובץ הגדרות שמודבר לתוכנית כפרט, ובמקורה שאין התוכנית טוענת את קובץ ההגדרות בירית המחדל. כאשר קובץ ההגדרות מכיל את ההגדרות של הסימולציה בצורה קובץ טקסט, ובו כל ההגדרות הדרושים להפעלה של הסימולציה.

פורטטיביות גבוהה - מעבר למערכת הפעלה מסווג לינוקס לא דרוש שייכתוב של הקוד – רק החלפה של טימר הטיקים לטימר טיקים של לינוקס. (Windows.GetTickCount() ב-(), uptime -() בלינוקס).

5.2 גרסת GPU

גרסת GPU עובדת בדיק באותה צורה כמו גרסת CPU, מלבד פונקציות החישוב הבאות:

- Force •
- Predict •
- Correct •

גרסת GPU מומשא בעזרת API Driver מהסיבות שהוסברו, ומכליה את שני שיטות מיימוש החישוב כוחות – TPP ו-BPP, כאשר את השיטה קבועים בקובץ ההגדרות, יחד עם כמות הבלוקים שהמשתמש הוצה להשתמש בסימולציה.

כאשר משתמשים בשיטת TPP, כמות הבלוקים מתייחסת רק ל-`predict`, `correct` וחישוב התוצאות שמתבצע לאחר חישוב הכוחות אם יש פגעה בקובסא. בשיטת BPP, כמות הבלוקים מתייחסת גם לחישוב הכוחות.

על מנת להתגבר על בעיות הריצה עם שיטת BPP, בהרצה של יותר מ-512 התהיליכונים בבלוק (אי אפשר להפעיל יותר מ-512 תהיליכונים בבלוק ב-CUDA) – נוצר מגנון Fallback – שזמן הסימולציה, כאשר נוצרת רשימה שכנים חדשה, ולפחות אחת מהרשימות של אותן כלשהוא גדולה מ-512 חלקיקים, המנגנון יפעיל במקום את שיטת TPP לחישוב הכוחות את שיטת BPP לחישוב הכוחות, עד אשר תוגרל רשימה חדשה שאינה מכילה רשימה כלשהיא גדולה יותר מ-512 שכנים.

Equilibrate 5.3

בשביל להשתמש בסימולציה – צריך קודם כל ליצור קונפיגורציה המכילה מיקומים ומהירות ומסות של האטומים, במצב הקרוב לשינוי משקל.

תוכנת Equilibrate עובדת לפי הפסאודו-קוד הבא (דומה לעקרון של פסאודו-קוד 1 עם מספר שינויים):

Pseudocode No' 5:

```
equilibrate
    choose-positions1
    choose-velocities2
    force
    calculate initial accelerations
    while timestep ≠ target timestep
        for each atom
            predict
            force
            correct
            if abs(atom[x,y,z]) > box / 2
                set positions in the box
                invert velocities
                force
                set new acceleration
        each 50's timestep3
            choose-velocities
```

1. הגרלת מיקומים בצורה אקראית בהתפלגות אחדה כאשר צריכים שיתקיים הכללים הבאים:

- בקונפיגורציה שמכילה חישוב כוחות בין אטומים בודדים – צריך רק לבדוק שהאטומים אינם קרובים יותר מדי.
- בקונפיגורציה שמכילה חישוב כוחות בין זוגות אטומים - אטומים שהם זוג (למשל כמו זוגות אטומי חנקן) יהיו מספיק קרובים כדי שתתרחש ביניהם ריאקציה.
- בקונפיגורציה שמכילה חישוב כוחות בין זוגות אטומים - אטומים שאינם זוג (למשל אטומי חמצן וחנקן) יהיו מספיק רוחקים כדי שלא תיווצר התפוצצות.
- המרחקים בין האטומים קבועים לפחות יחס דחיסות בהתאם לסוג קונפיגורציה שאנו רוצים, למשל אם אנחנו רוצים גז שבו זוגות אטומים רחוקים מזוגות אחרים, או צבר שבו זוגות אטומים קרובים לזוגות אחרים.

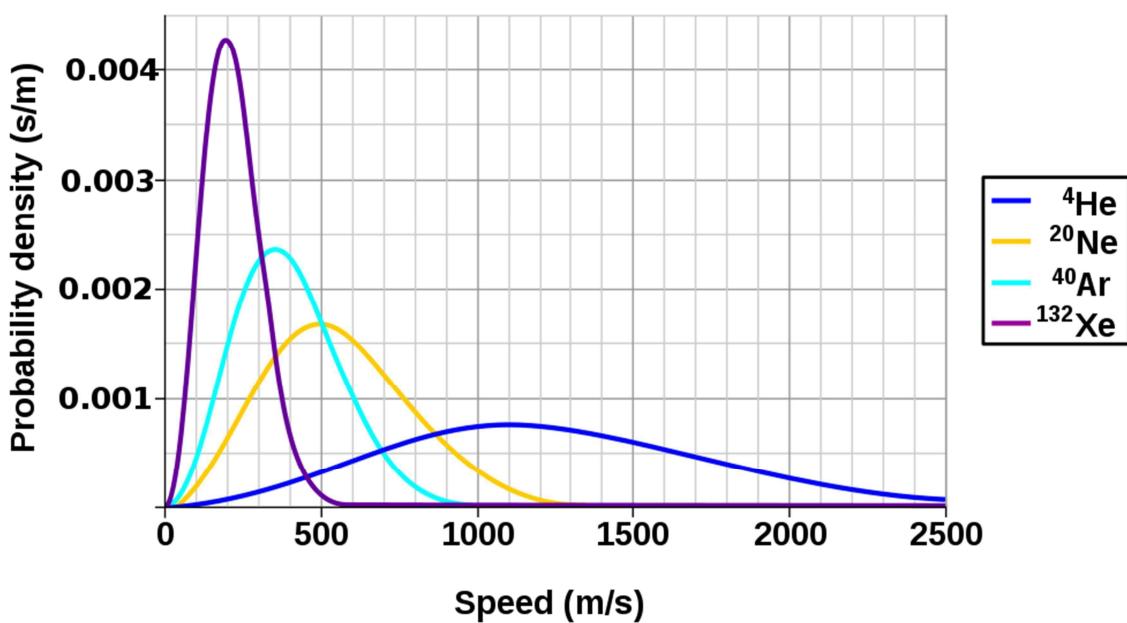
2. הגרלת מהירותים בצורה אקראית לפי התפלגות מקסואל-בולצמן.

התפלגות מקסואל-בולצמן - היא התפלגות המשמשת בפיזיקה ובכימיה לתיאור התפלגות גודל של וקטור, שכל אחד מרכיביו מתפלג באופן נורמלי ובלתי תלוי. השימוש הנפוץ ביותר שלו הוא לתיאור התפלגות המהירותים של חלקיקים בגז אידיאלי, אך היא יכולה לתאר, בשינוי הפרמטרים גם, לדוגמה, את התפלגות התנע או האנרגיה שלהם.

באופן פורמלי, אם Z, Y, X הם משתנים מקריים נורמליים בלתי-תלויים בעלי תוחלת 0 וסטיית תקן a אז המשתנה המקרי Z מוגדר על ידי $Z = \sqrt{X^2 + Y^2 + Z^2}$ ומתפלג בתפלגות מקסואל-בולצמן עם פרמטר a .

$$\text{כאשר במקרה שלנו } a = \sqrt{\frac{kT}{m}}.$$

להלן דוגמא של התפלגות מהירות של מספר גזים אטומים:



3. כל מספר צעדי-זמן, במקרה שלנו כ-50 צעדי-זמן, מגבירים מהירותים מחדש. הסיבה לכך היא על מנת ליצב את הקונפיגורציה. עקב שהוא מגבירים בצורה אקראית אין זה אומר שהמהירותים תקינות יוכל להיות שלאחר כמה צעדי זמן המהירותים יגדלו בצורה משמעותית (אם למשל יש אטומים קרובים) – لكن נגריל מהירותים שוב ושוב עד שהקונפיגורציה תת'יצב, עם מהירותים שמתאימות לטמפרטורה ההתחלתית.

5.4 ממשק משתמש

המשק נכתב בעזרת .NET Framework

בעזרת המשק אפשר בקלות לקבוע את הנתונים הדרושים ולהפעיל את הסימולציה ולאחר סיום הסימולציה - לראות את התוצאות במרחב תלת ממד, ובתצוגה גרפית של האנרגיות, מרכז מסה, תנע, וטמפרטורה.

מעבר של המשק למערכת הפעלה מסווג לינוקס לא דורש עבודה רבה, עקב זאת שאפשר להשתמש בLINQ ב-.NET Framework, שצורת העבודה בה מעוד מצכירה את QT Framework ..NET Framework

התצוגה הגרפיה משתמש במשק ה-GDI של .NET Framework ..

התצוגה התלת ממדית משתמש במשק OpenGL. אף על פי כן שהIMPLEMENTATION הראשי נכתב תחת מערכת ההפעלה OpenGL, נבחר Windows, נמצואות התלת ממדית ולא Direct3D מספר סיבות:

- OpenGL פורטబיל הרבה יותר מאשר Direct3D, מאפשר שימוש במגוון רחב של שפונות ומערכות הפעלה.
- OpenGL בעלת דרישות חומרה יותר נמוכות מאשר Direct3D.
- מעבר לגרסה חדשה של OpenGL, לא דורש שיכטוב מחדש של קוד, והוא ביכולת המערכת לפעול.
- לעומת Direct3D 9, Direct3D 10-11 מדורש כתיבה מחדש של כמעט כל הקוד, ובנוסף עובד רק בWindows Vista ומעלה.
- Direct3D בשימוש יותר למשחקים, כאשר OpenGL בשימוש יותר בגרפיקה מקצועית.

שפונות תכנות בפרויקט:

סימולציה - C\ C++ With CUDA Extensions

משק משתמש - C++ With .NET CLR Extensions

עזרים גרפיים – OpenGL

כלים: Microsoft Visual Studio 2008, מערכת הפעלה: Microsoft Windows

תרשימים ואירועים:

תרשימים מודולים של הסימולציה: ראה נספח 9.1.

תרשימים של הסימולציה: ראה נספח 9.2.

תרשימים מחלקות – הסימולציה כתובה ב-C, לכן עבורה אין תרשימים מחלקות, אך קיימים תרשימים מחלקות עבור המשק ומפענחו קבועי ההגדירות – ראה נספח 9.3.

תצלומי מסך של המשק: ראה נספח 9.16.

6 בדיקות

6.1 בדיקות שימוש התוכנה

על מנת לבדוק שהסימולציה עובדת בצורה תקינה, בודקים כל מספר צעדים שהאנרגיה נשמרת. חישוב האנרגיה (אנרגייה פוטנציאלית ואנרגיה קינטית) מתבצע על כל האטומים, בלי רשיית השכניות – על מנת לקבל את מצב הסימולציה המלא – חישוב זה נעשה כל מספר צעדים ולא כל צעד ולכן אינו משפיע על זמן הריצה.

בנוסף, לבדיקות נוספות בודקים גם את:

- טמפרטורת המערכת – נמדדת במעלות קלוריין.
- תנע בשלושת הצירם.

לא בוצעו בדיקות למשק עקב שהוא אינו החלק החשוב בפרויקט – אלא רק כלי עזר.

6.2 בדיקות המערכת ומסקנות

מפורט המעבדים שנבדקו:

- 2.66Ghz - Intel Core i7 920 – ליבת Nehalem.
- 648Mhz – GeForce 285GTX - GPU בעל 30 SM.
- 650Mhz – GeForce 9600GT - GPU בעל 8 SM – להשוואה בלבד מול GeForce 285GTX.

6.2.1 השוואת מול הגרסה המקורית שנכתבה ע"י כימאים

- כבר 125 אטומים, מתוכם 97 מסוג גז אציל ניאון, 14 אטומי חנקן, 14 אטומי חמצן – יש לשים לברשימית השכניות מה שעובד עקב חזק הzcבר. 1000 צעדי זמן.

סימולציה	זמן ריצה
1. גרסה מקורית	77375ms
2. גרסת הפרויקט בלי רשיית השכניות	12641ms
3. גרסת הפרויקט עם רשיית השכניות	7360ms
4. גרסת GPU עם רשיית השכניות	529ms

6.2.2 השוואת CPU, TPP ו-BPP ברג

- ג', מ-100 עד 6400 אוטומים בקצב יצות של פי-2, כאשר חצי המ אוטומי חנקן, וחצי המ אוטומי חמוץ. 10000 צעדי זמן.

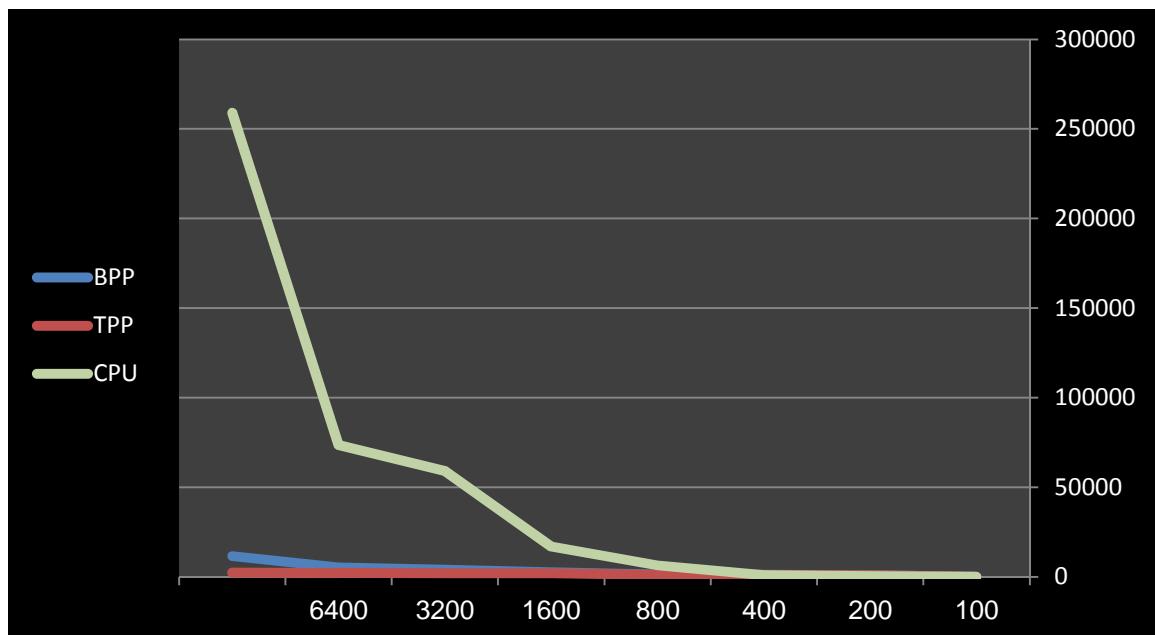
אפשר לראות שבג' עדיף להשתמש ב-TPP ולא ב-BPP, עקב זאת שכמות האוטומים השכנים לכל אוטום המ בסביבות ה-6-2 אוטומים, ובמקרה של שימוש ב-BPP, בכל בלוק מופעלים כמות תהיליכונים ככמות השכנים. לעומת זאת ב-TPP כמות התהיליכונים היא

כמות אוטומים	BPP	TPP	CPU
100	847ms	823ms	375ms
200	1021ms	977ms	782ms
400	1278ms	1085ms	6343ms
800	2429ms	1932ms	16934ms
1600	3804ms	1993ms	59054ms
3200	5062ms	2014ms	73537ms
6400	11578ms	2215ms	259061ms

בהתאם לכמות האוטומים המוצאים בבלוק – בדוגמא פה היו 30 בלוקים, אשר כמות התהיליכונים

$$\text{בלוק היא} \left[\frac{\text{כמות האוטומים}}{\text{כמות הבלוקים}} \right].$$

בסוף אפשר לראות שכמות אוטומים נמוכה, ה-UP נותן זמינים טובים יותר – הסיבה לכך היא שכאשר שבגלל שיש לנו קצת חלקיקים – זמן השהייה ב-Kernel אינו גדול משמעותית או אפילו קצר מזמן זה – Context Switching.



גרפים נפרדים – נספחים 9.7, 9.8

6.2.3 השוואה של TPP, BPP ו-CPU בצבר בגין

- צבר בגין, מ-100 עד 800 אוטומים בקפיצות של פי-2, כאשר חצי הם אוטומי חנקן, וחצי הם אוטומי חמן. 1000 צעד זמן.

אפשר לראות כי בצבר ה-BPP מוציא תמיד זמנים טובים יותר גם מה-CPU וגם מה-TPP, בכל כמות של אוטומים. גם כאן TPP נמצא בעיית ה-Context Switching Time.

כמות אוטומים	BPP	TPP	CPU
100	943ms	8371ms	6727ms
200	3812ms	20354ms	18289ms
400	8732ms	33640ms	31711ms
800	16527ms	62017ms	84563ms

गרפ – נספח 9.9.

6.2.4 השוואה של TPP, BPP ו-CPU בצבר חזק

- צבר חזק, מ-100 עד 400 אוטומים, בקפיצות של 100 אשר חצי הם אוטומי חנקן, וחצי הם אוטומי חמן. 1000 צעד זמן.

כמות אוטומים	BPP	TPP	CPU
100	2887ms	83722ms	136806ms
200	13708ms	213239ms	407422ms
300	32612ms	335657ms	648697ms
400	64981ms	650806ms	1297391ms

गרפ – נספח 9.10.

6.2.5 השוואה של TPP על GeForce 285GTX מול GeForce 9600GT

- ג', מ-100 עד 800 אוטומים בקפיצות של פי-2, אשר חצי הם אוטומי חנקן, וחצי הם אוטומי חמן. 1000 צעד זמן.

אפשר לראות כי לכרטיס המסדר החלש, חלש פי-4 עד-6, אף על פי כן שמכיל פי-3 פחות מעבדים. כאמור יש להתייחס לכמות הרגיסטרים הקטנה יותר ולזיכרון הגלובלי האיטי יותר. ההסבר לקפיצות החזקות בזמן הריצה – חוסר ברגיסטרים, וכן צריך לעבור במספר בלוקים גדול יותר.

כמות האוטומים	285GTX	9600GT
100	823ms	3822ms
200	977ms	4049ms
400	1085ms	4231ms
800	1932ms	6842ms
1600	1993ms	7083ms
3200	2014ms	7574ms
6400	2215ms	12315ms

गרפ – נספח 9.11.

6.2.6 השפעת רשיימת השכניות בגז

- גז, 200 אטומים, כאשר חצי הם אוטומי חנקן וחצי הם אוטומי חמצן. 10000 צעדי זמן. גודל של BOXLAND הוא 10560.

כפי שניתן לראות, השימוש בסימולציה של גז עם רשיימת שכניות מקטין את זמן הריצה פי-970 מסימולציה של גז בלי רשיימת שכניות. Full – בלי רשיימת שכניות.

RS	RCUT	TIME
10	8	782ms
20	16	906ms
40	32	2594ms
80	64	21094ms
160	128	167609ms
full	full	757797ms

גרף – נספח 9.12

6.2.7 השפעת רשיימת השכניות בצביר

- צברBINONI, 200 אטומים, כאשר חצי הם אוטומי חנקן וחצי הם אוטומי חמצן. 1000 צעדי זמן. גודל של BOXLAND הוא 1056.

הSHIPOR הוא לא חזק כמו בגז במעבר מסימולציה בלי רשיימת שכניות לSIMOLCIA עם רשיימת שכניות, עקב זאת שגם בגודל הרשימה הכי קטן יש הרבה שכנים, בגלל הצבר. גраф – נספח 9.13.

RS	RCUT	TIME
10	8	18289ms
20	16	322641ms
40	32	1840812ms
full	full	2040735ms

6.2.8 התפלגות זמנים בין גז לצבר

- צבר חזק וגז, 200 אטומים, כאשר חצי מהם חנקן וחצי מהם חמצן. 1000 צעדי זמן.

אפשר לראות שההבדל העיקריים הם רק בזמן חישוב הכוחות, כאשר במקורה של הצבר, יש יותר שכנים, והחישוב כוחות השני – משתמש הרבה בזיכרון הגלובלי, אך זמן הריצה שלו יוקח הרבה יותר זמן מהחישוב כוחות הראשוני.

פעולה	פילוג זמנים בגז	פילוג זמנים בצביר
בנייה רשיימת שכניות	6.94ms	3.15ms
ביצוע	17.01ms	17.03ms
ביצוע	15.97ms	15.81ms
ביצוע	4572.68ms	32.01ms
ביצוע	8992.54ms	Bond Order 1
כל הפעולות יחד	13605.14ms	Bond Order 2

לעומת זאת חישוב הכוחות במקורה של הגז, מתנהג בדיק הפור, והסיבה לכך היא כמות קטנה מאוד של שכנים, כאן כמות הגישות לזכרון הגלובלי קטנה ולקוון זמן הריצה שלו הולך בלבד על החישובים.

ההבדל בזמןי בניית רשיימת השכניות הוא בגלל זמן בניית הרשימה תלוי בכמות השכנים בהתאם, ובגלל שבצבר יש הרבה יותר שכנים מאשר בגז, זמן הריצה שלו הרבה יותר גדול.

גרף – נספחים 9.14, 9.15

7 השוואה לפתרונות בספרות

במאמר של אנדרסון, לורנד וטרווסט [2], גם ממשים דינמיקה מולקולרית בעזרת CUDA. המימוש תלוי בכמות התאים.

אם הם השתמשו בזיכרון מיטמן טקסטורה בשביל רשימת השכניות אך הגישה לרישימה מבוצע בעזרת גישה חד ממדיות.

ב-A-CUDA הגישה הכי אופטימלית לזכרון הטקסטורה היא גישה דו-ממדית – כר תוכנן זיכרון מיטמן זה.

בנוסף המימוש הסופי של בניית הרשימה במאמר הוא על גבי ה-GPU, אנו משתמשים ב-Spatial Hash, שמציריך הרבה גישה אוטומטית לזכרון, ולכן למשו על ה-GPU או בקוד טורי ולא מקבילי, המימוש פחות עיל, אך בגלל שהוא תלויים בלבד בכמות האטום ולא בכמות התאים, המימוש הסופי יותר עיל. בנוסף עקב זאת שתת הרשימה לא בונים מחדש כל צעד אלה רק מתי צורך, הפגיעה בזמן הריצה עקב המעבר בין GPU חזקה ל-CPU היא פגיעה קטנה ולא מורגשת.

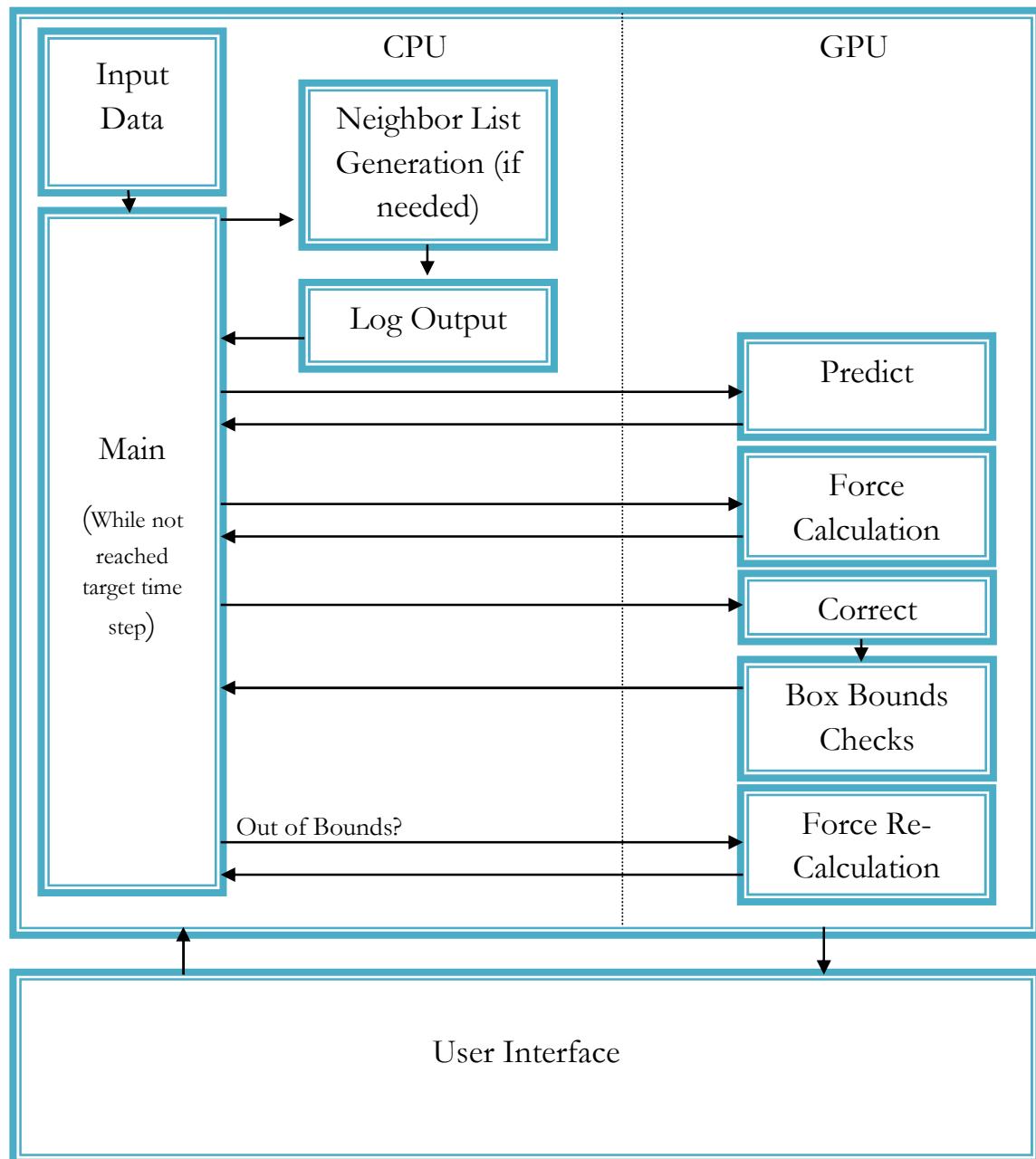
במאמר השיטה לעובדה עם האטומים היא שיטה הדומה לשיטת ה-BPP, אך היא אינה שיטה יעילה כאשר מספר האטומים לא מספיק גדול בשביל להעסיק את כרטיס המשך במלואו – השיטה הנוספת שמומשה במאמר – שיטת ה-BPP, מאפשר להעסיק את הכרטיס יותר בעזרת פיצול העבודה של חישוב הכוחות למספר תהילכנים – רעיון דומה לרענון הממומש ב-n-body simulation בספר GPU Gems [4]. פרק 31.

8 רשימת ספרות

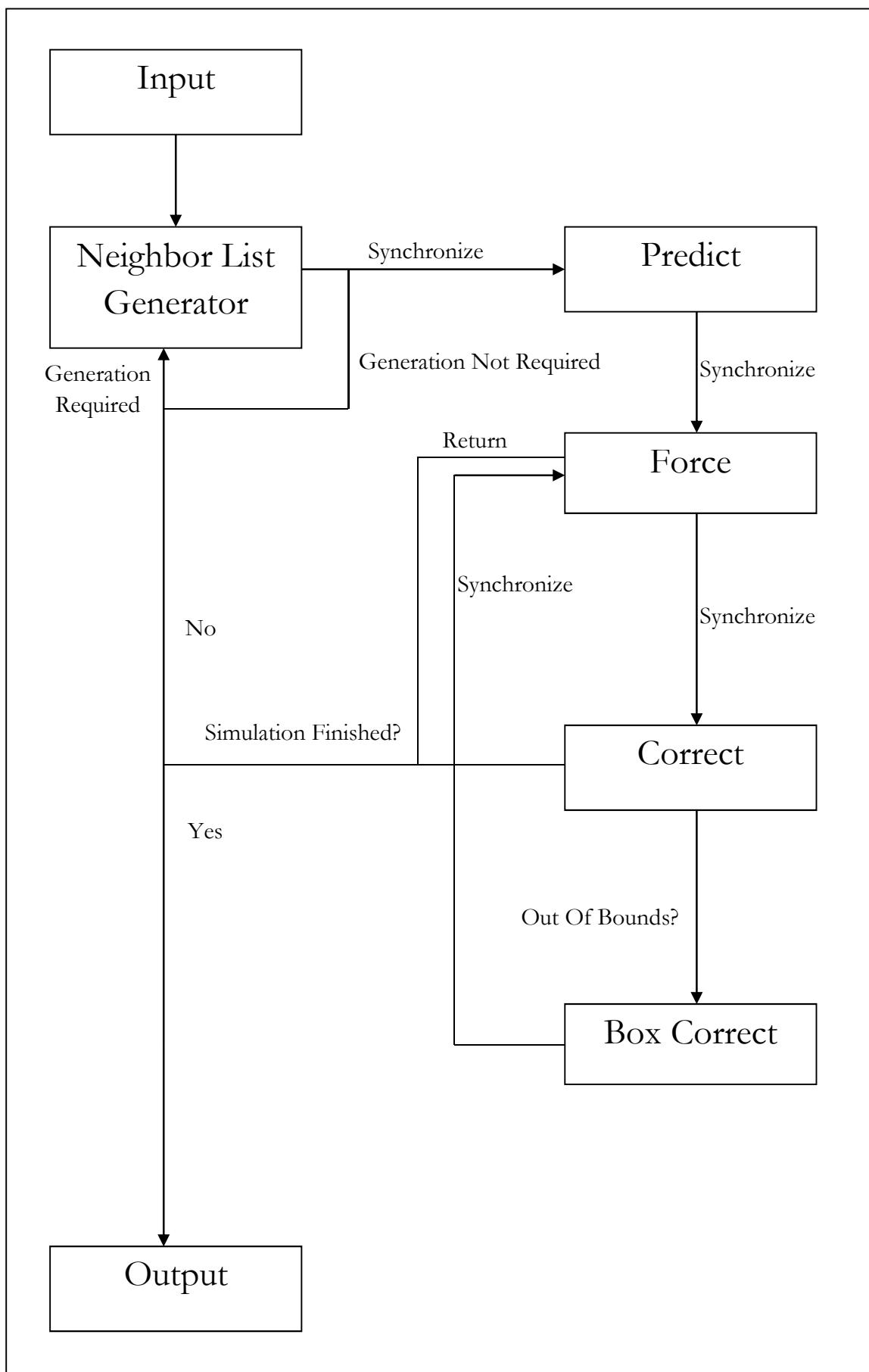
- [1] Allen, Michael P.
Introduction to Molecular Dynamics Simulation, 2004.
- [2] Anderson, Joshua A.; Lorenz, Chris D.; Travesset, A.
General Purpose Molecular Dynamics Simulations Fully Implemented on Graphic Processing Units, 2008.
- [3] Farber, Rob
"Dr. Dobb's CUDA, Supercomputing for the Masses",
<http://www.ddj.com/architect/207200659>, 2009.
- [4] Nguyen, Hubert
"GPU Gems 3"
Addison Wesley Professional, 2007.
- [5] Pagh, Rasmus; Rodler, Flemming Friche
Cuckoo Hashing, 2001.
- [6] Stone, John E.; Phillips, James C.; Freddolino, Peter L.; Hardy, David J.;
Trabuco, Leonardo G.; Schulten, Klaus
Accelerating molecular modeling applications with graphics processors
Journal of Computational Chemistry, 28:2618-2640, 2007.
- [7] NVIDIA CUDA™ Programming Guide, 2009.
- [8] NVIDIA CUDA™ C Programming Best Practices Guide, 2009.
- [9] Yao, Zhenhua; Wang, Jian-Sheng; Cheng, Min
Improved O (N) neighbor list method using domain decomposition and data sorting
- [10] ר' ליאו, ת' נחום-ץ' .
דינמיקה מולקולרית של תהליכיים כימיים, האוניברסיטה הפתוחה, 2008.

9 נספחים

9.1 תרשימים מודולרים של הסימולציה

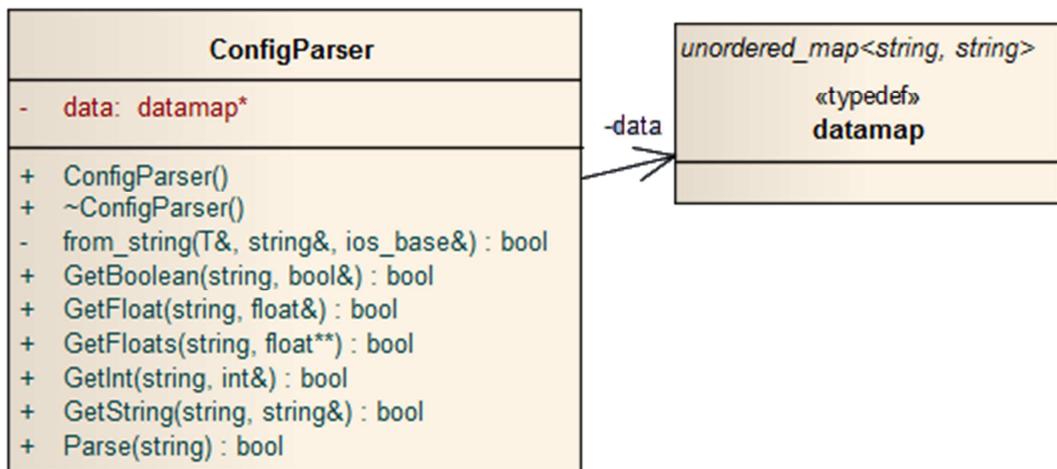


9.2 תרשימים Data Flow של הסימולציה

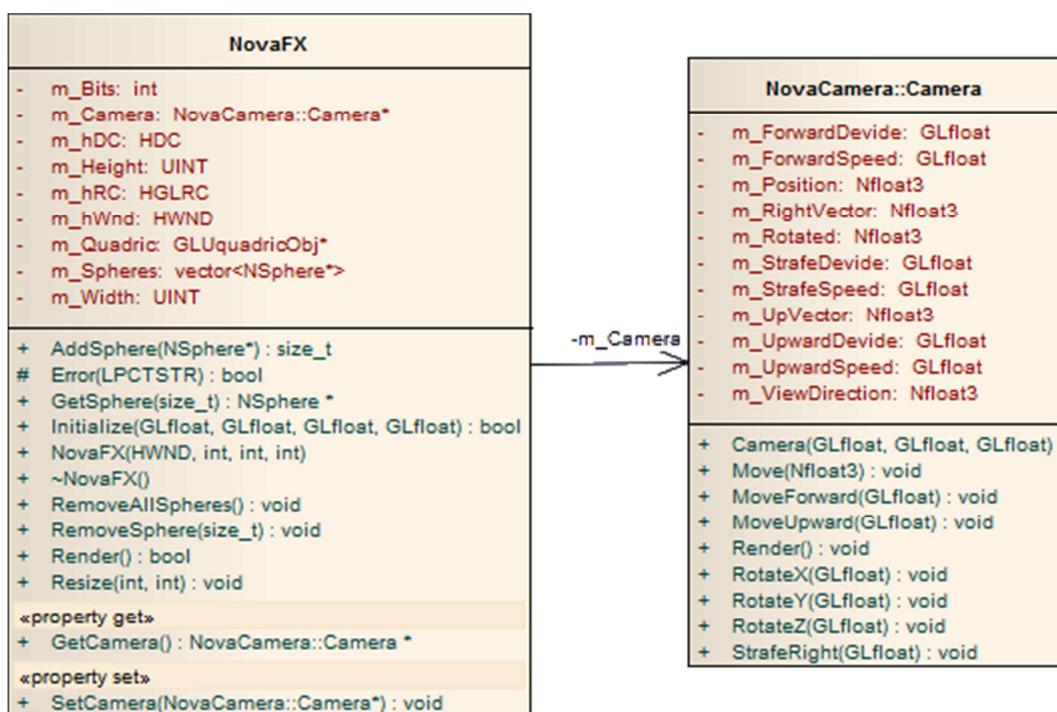


9.3 תרשימים מחלקות

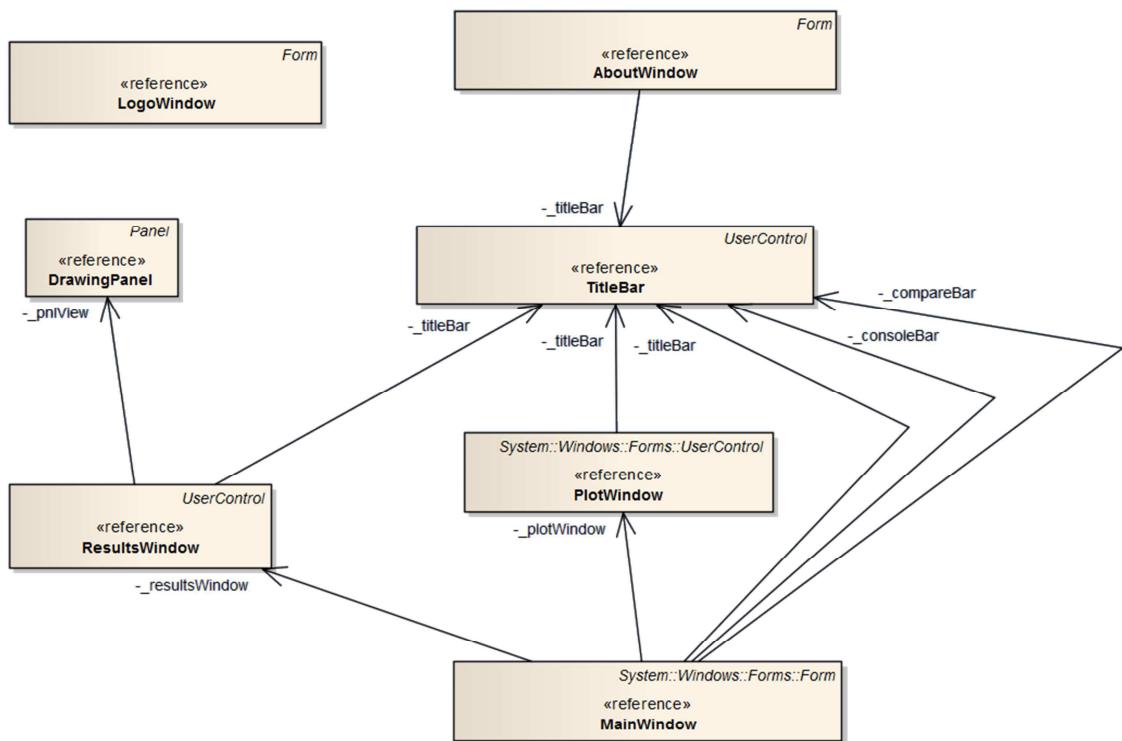
9.3.1 תרשימים מחלקות מפענחו קבצי הגדרות



9.3.2 תרשימים מחלקות ממשק תלת מימד מעיל OpenGL

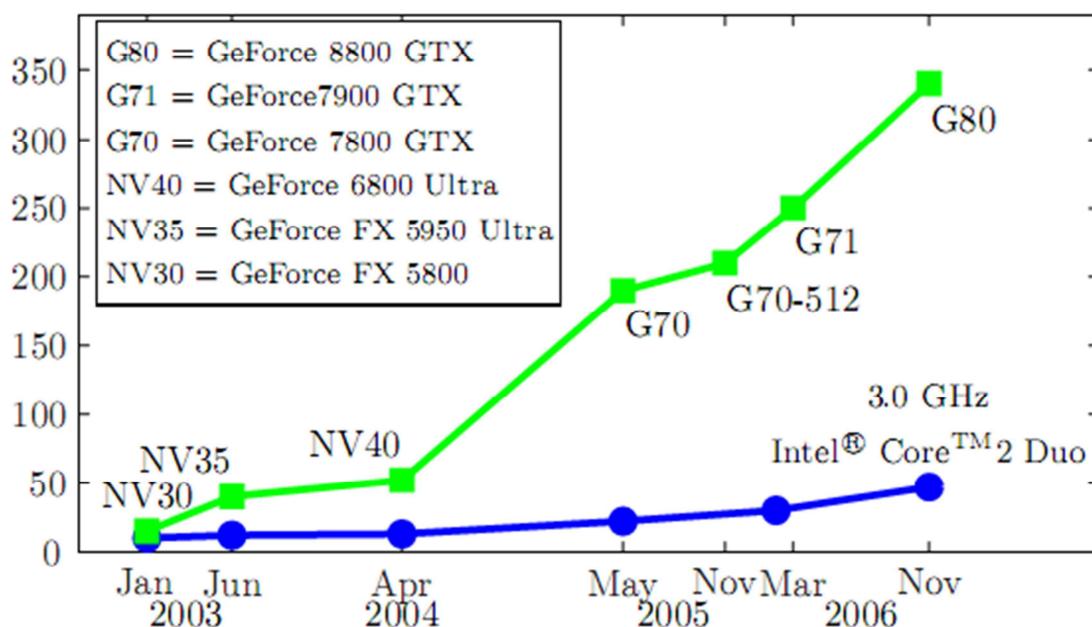


9.3.3 תרשימים מחלקות כללי ממשק משתמש



GPU 9.4

הוא מעבד "יעוד" שמטרתו היא הורדת עומס מהמעבד הראשי הנוצר עקב חישוביים גרפיים. הוא נמצא כיום בשימוש במחשבים אישיים, מערכות מושבצות מחשב, טלפונים ניידים, תחנות עבודה וكونסולות משחקים (למשל PlayStation™). מעבדים גרפיים מאוד ייעלים בעיבוד גרפיקה ממוחשבת, ובגלל המבנה המקבלי – מספר רב של מעבדים העובדים בקבוצות – הם הרבים יותר ייעלים מאשר מעבדים כליליים.



CUDA 9.5

זה מודל תוכנות מקבילי וסיבת תוכנה אשר מאפשרת לנצל את כוח החישוב של המעבד הגרפי (GPU) לעיבוד מידע (לעומת חישובים גרפיים שבדרך כלל נהוגים).

עיצובו לפי מספר מטרות: (Compute Unified Device Architecture) CUDA

1. CUDA ביסודה היא הרחבה קצירה לשפת C שמאפשרת הטמעה ישירה של אלגוריתמים מקבליים.
2. מתכנתים יכולים להתמקד בעיצוב ובנינה של אלגוריתמים מקבליים ופחות להתמקד במימוש שלהם (לעומת מימוש בשפת Assembly).
3. CUDA מאפשר חישובים ריבגוניים על ידי קר שאפליקציות יכולות להשתמש גם במעבד הכללי וגם במעבד הגרפי ייחודי. כאשר קוד טורי רץ על המעבד הכללי וקוד מקבילי רץ על המעבד הגרפי.
4. CUDA מאפשר שילוב חישובים על המעבד הגרפי באפליקציות קיימות בלי הצורך בשכתוב מחדש של כל התוכנית.

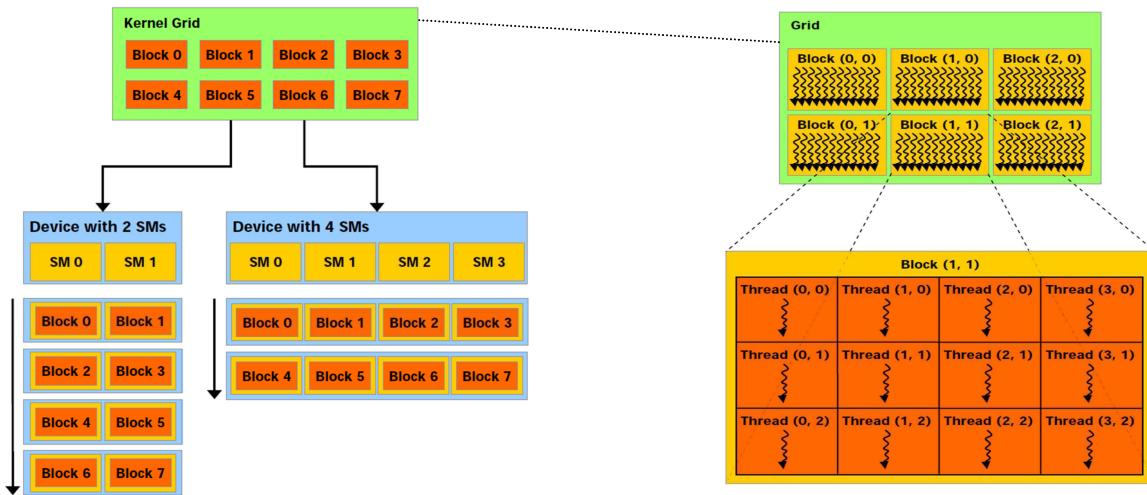
הארQUITטורה של CUDA –

המעבד הגרפי בניי ממספר רב של מעבדי DSP – והם מוחולקים לקבוצות של 8 מעבדים. קבוצות אלו נקראות SM – SIMT Multiprocessor, וכל קבוצה שכזו יש זיכרון משותף (בדומה לזכרון Cache במעבד כללי) שזמן הגישה אליו הוא מהיר מאוד – קר שאפשר לשתוף מידע בין כל 8 מעבדים והshitופ הזה יעשה בצורה מהירה, בעוד זיכרון המשותף לכל מעבד בקבוצה יש אוגרים ויחידות MAD – Multiply & Add – נקרא גם MAC.

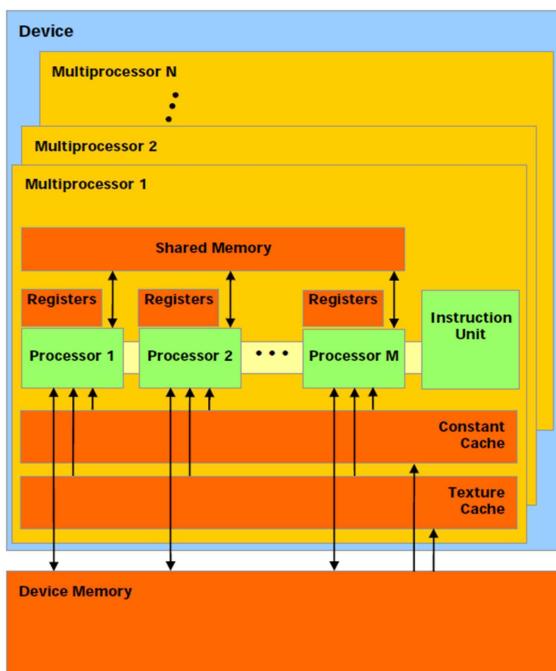
shitopf מידע בין הקבוצות נעשה דרך הזיכרון הגלובלי – שմבחןת נפח הוא ענק, אך זמן הגישה אליו הוא איטי (לעומת הזיכרון המשותף).

אupon העבודה של CUDA על המעבד הגרפי הוא כך: כל התהיליכונים מסודרים בבלוקים (למשל 8 בלוקים של 125 תהיליכונים). כל SM מקבל בלוק ומරיז את כל התהיליכונים שבבלוק. בין התהיליכונים בבלוק אפשר למסנן אך בין הבלוקים אין בחומרה אפשרות למסנן.

סדר התהליכיונים ב-CUDA על גבי ה-SM:



השוואה בין מעבד כללי למעבד גרפי:



חומרה והיררכיות הזיכרון על גבי המעבד הגרפי:

בכל SM יש 8 מעבדי DSP, 8192 רегистרים בגרסה

1.0 של CUDA ו-16384 רегистרים בגרסה 1.2 של

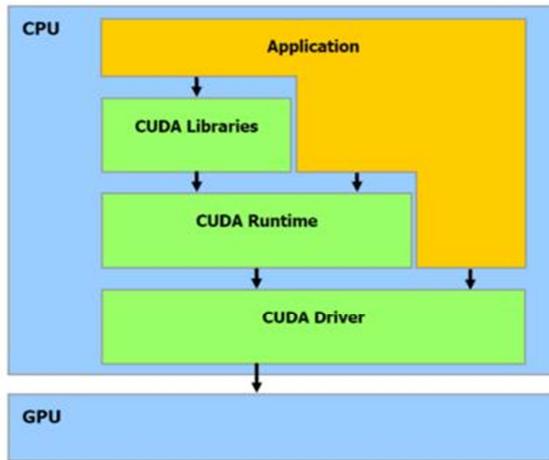
.CUDA

בנוסף לכל SM יש מטמון לזכרון קבועים בגודל

.KB6-8, ומטמון לזכרון טקסטורת בגודל KB8

קיימות שני גישות לעובדה עם CUDA בצד ה-CPU:

CUDA Driver API ו CUDA Runtime API,



הינה שכבה API שמומנשת מעל שכבת ה-API, CUDA Runtime API, ומאפשרת מספר דברים:

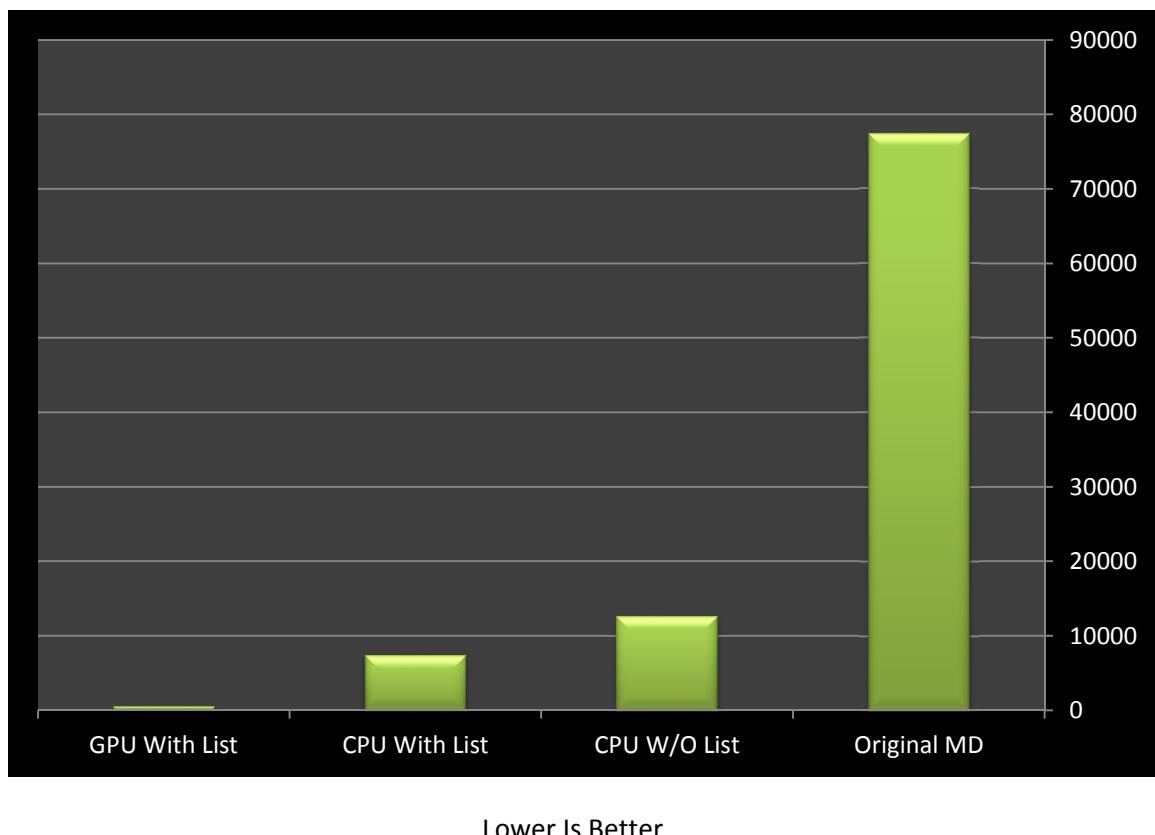
- קוד קצר יותר וקריאה יותר.
- שילוב של Kernel CUDA ייחד עם קבצי הרצה של התוכנית במקום קבצי cubin של קוד מכונה של CUDA שצריך לטעון בזמן ריצה כמו שעושים ב-API .Driver
- קל יותר לתכנת בעזרתו.
- מאפשר Device Emulation – הריצה של הקוד ל- CUDA על גבי ה-CPU.

לעומת זאת CUDA Driver API מאפשר:

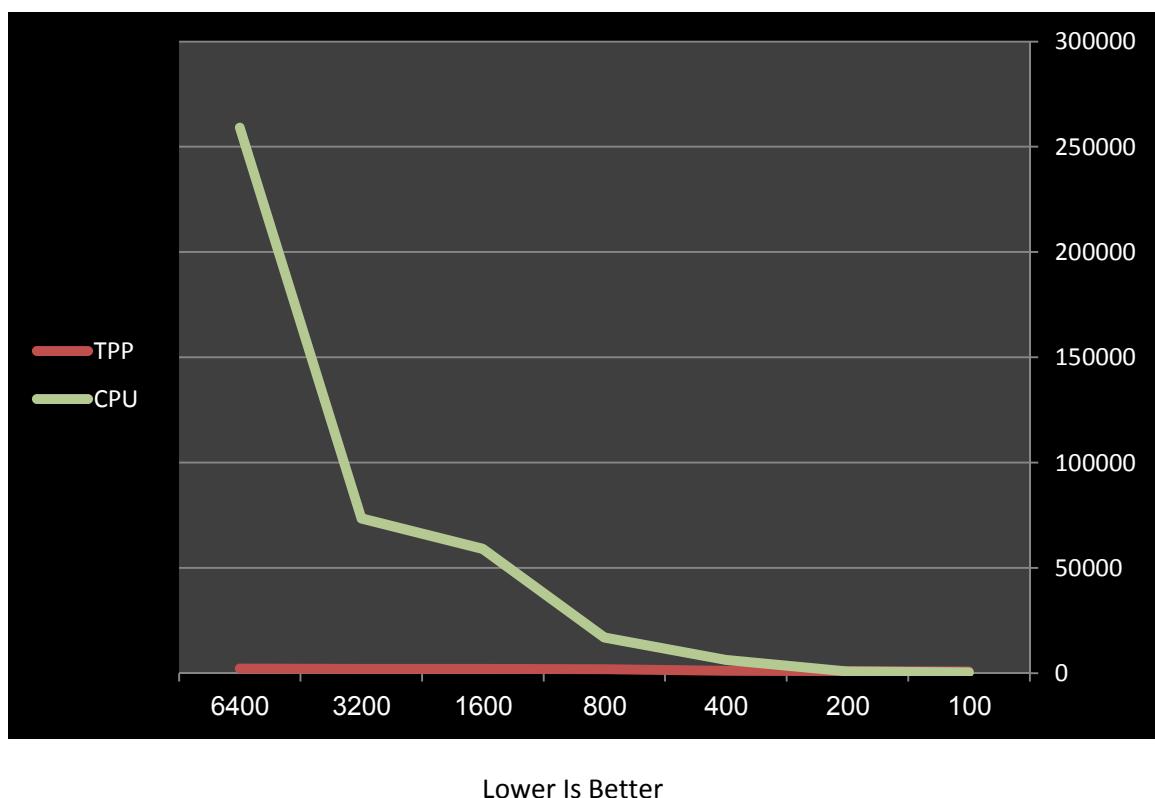
- אפשרות לגשת ליותר מידע על החומרה, למשל לבצע שאילתא של כמות הזיכרון החופשי או בשימוש על גבי הכרטיס מסך אפשר רק דרך CUDA API.
- שליטה יותר חזקה ב-CUDA, ויעילות גדולה מ- API CUDA Runtime .
- מאפשר להגדיר את הקונפיגורציה של Kernel לפני הריצה עצמה של הסימולציה (ב- CUDA Runtime API בניית הקונפיגורציה נעשית כאשר קוראים ל- Kernel). ואם יש שינוי באחד המרכיבים בקונפיגורציה אפשר רק לשנות את הערך השהתנה ולא צריך לבנות את כל הקונפיגורציה מחדש.
- פורטabilitות בין מערכות הפעלה ושפות (Fortran, Python וכו').
- שימוש במספר כרטיסי מסך באותו התהיליכון ב- CPU, מה שחייב את הצורך לՏונר ולהעביר מידע בין תהילכנים ברמת ה- CPU.

שני ה-API הם Mutually Exclusive, כלומר רק אחד מהם יכול להיות בשימוש.

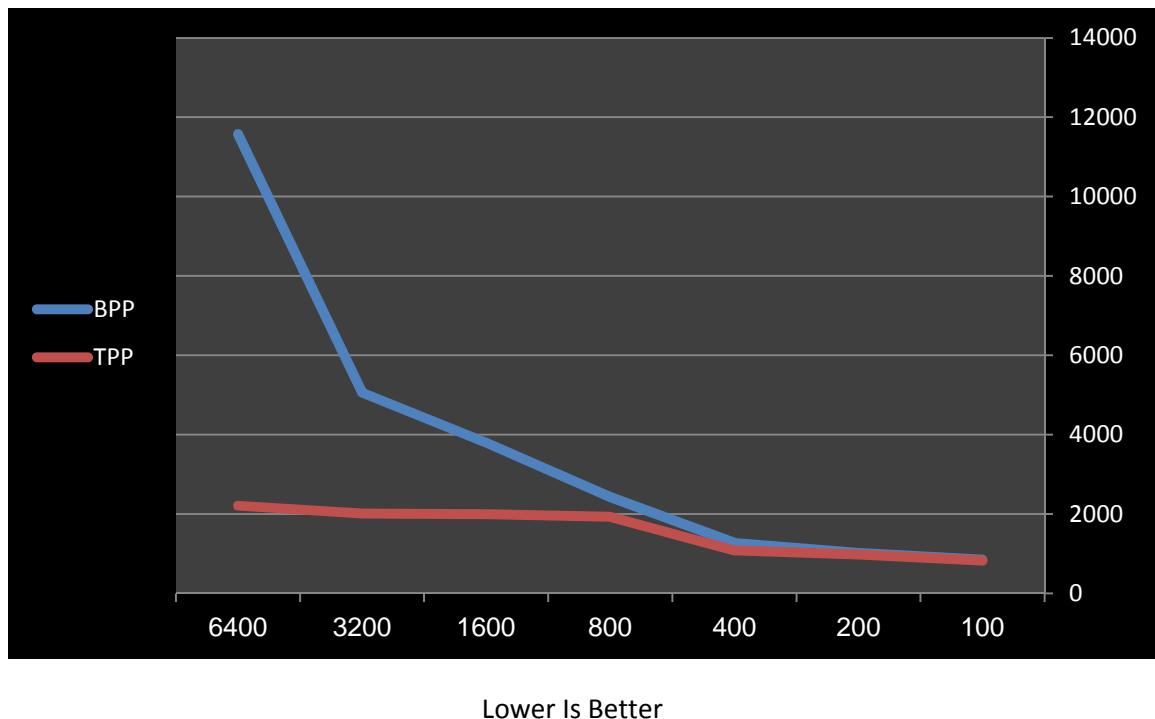
9.6 גרפַּט – מגרסת ה-C**וּמָאִים ועַד 7-GPU.**



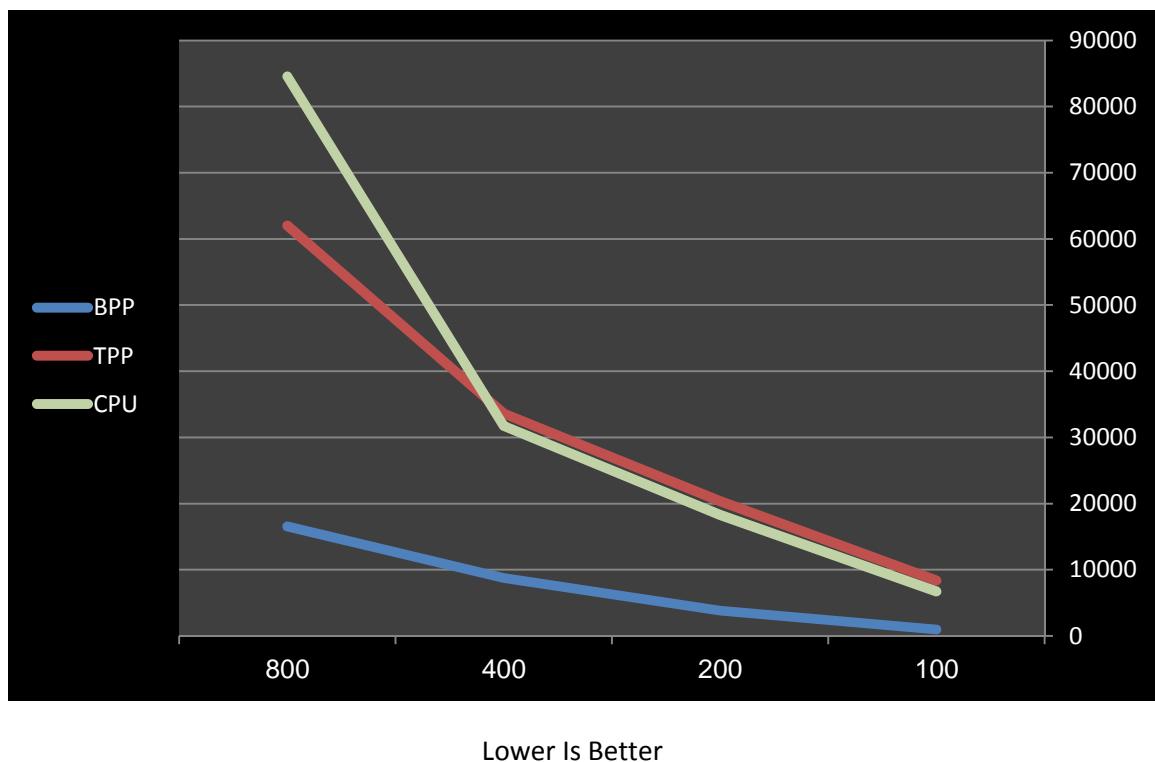
9.7 גרפַּט – TPP vs. CPU - 2



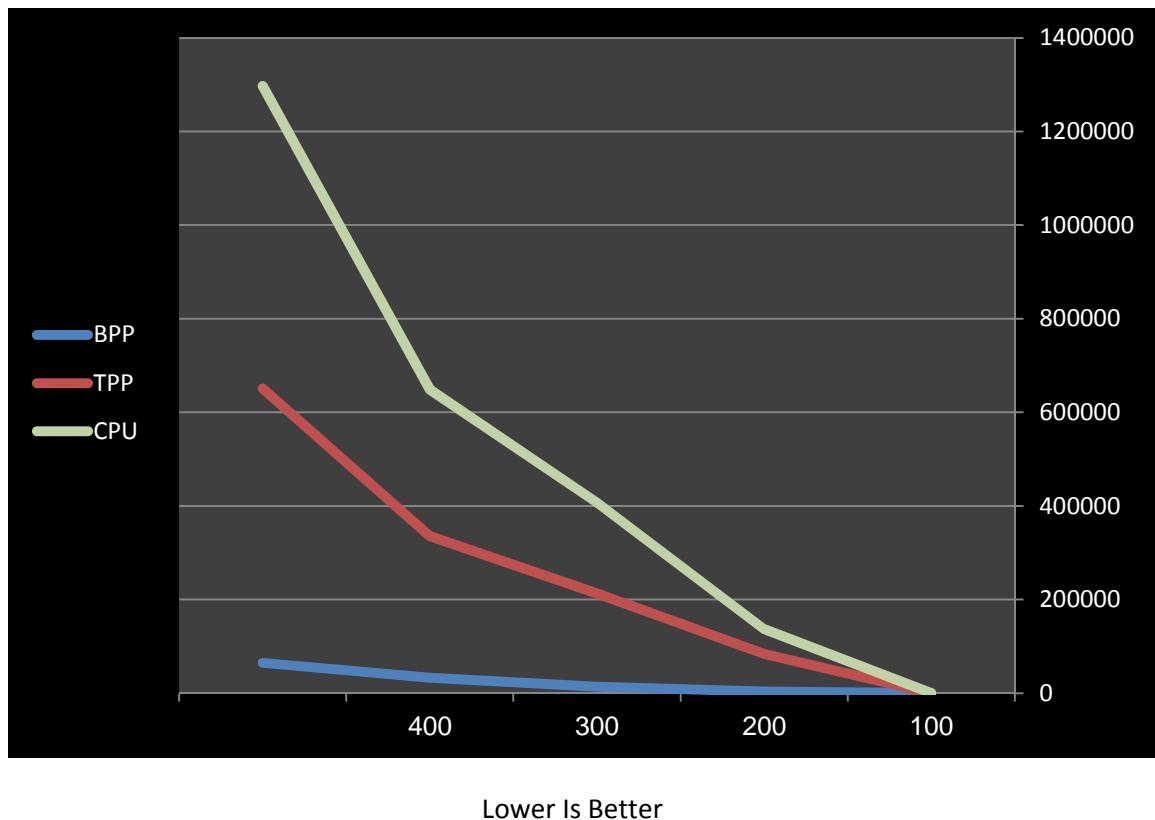
תג - BPP vs. TPP - 3 גראף 9.8



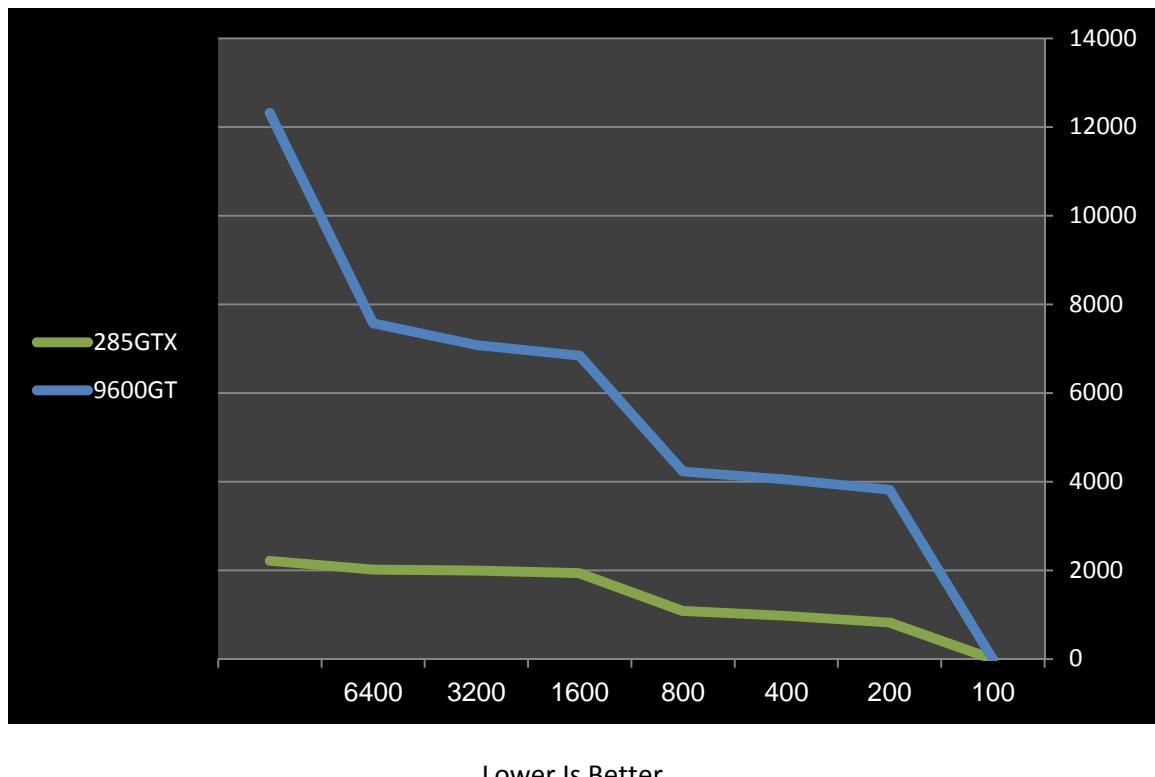
תג - BPP vs. TPP vs. CPU - 4 גראף 9.9



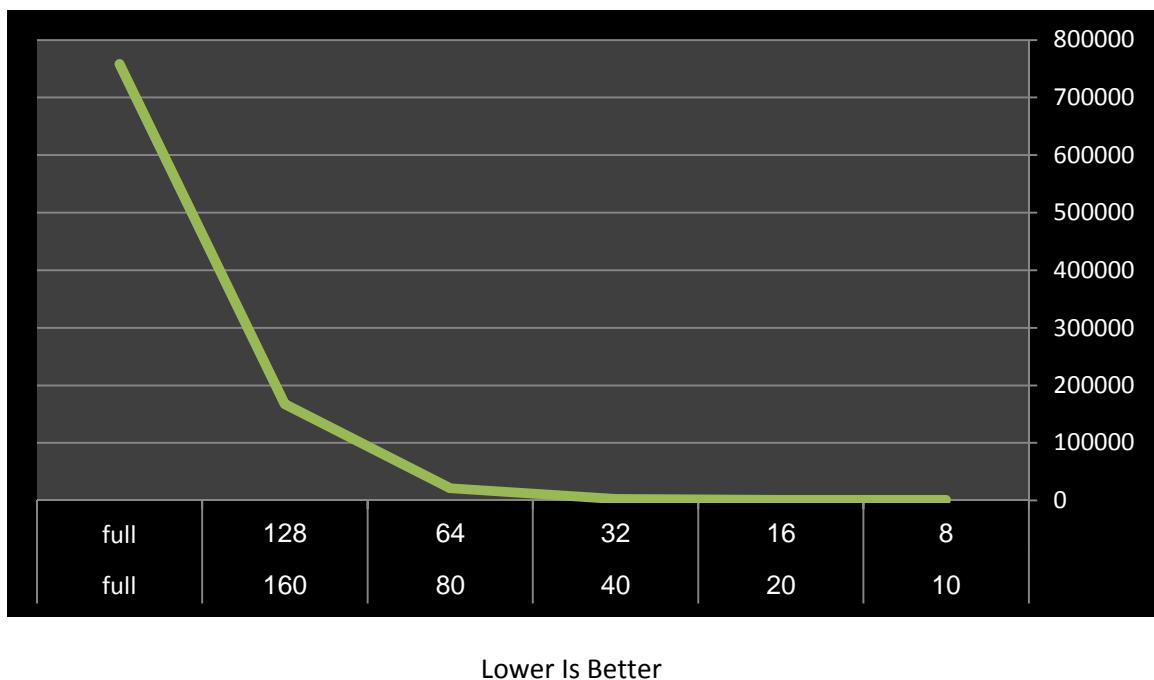
גראף 9.10 – צבר חזק – TPP vs. BPP vs. CPU



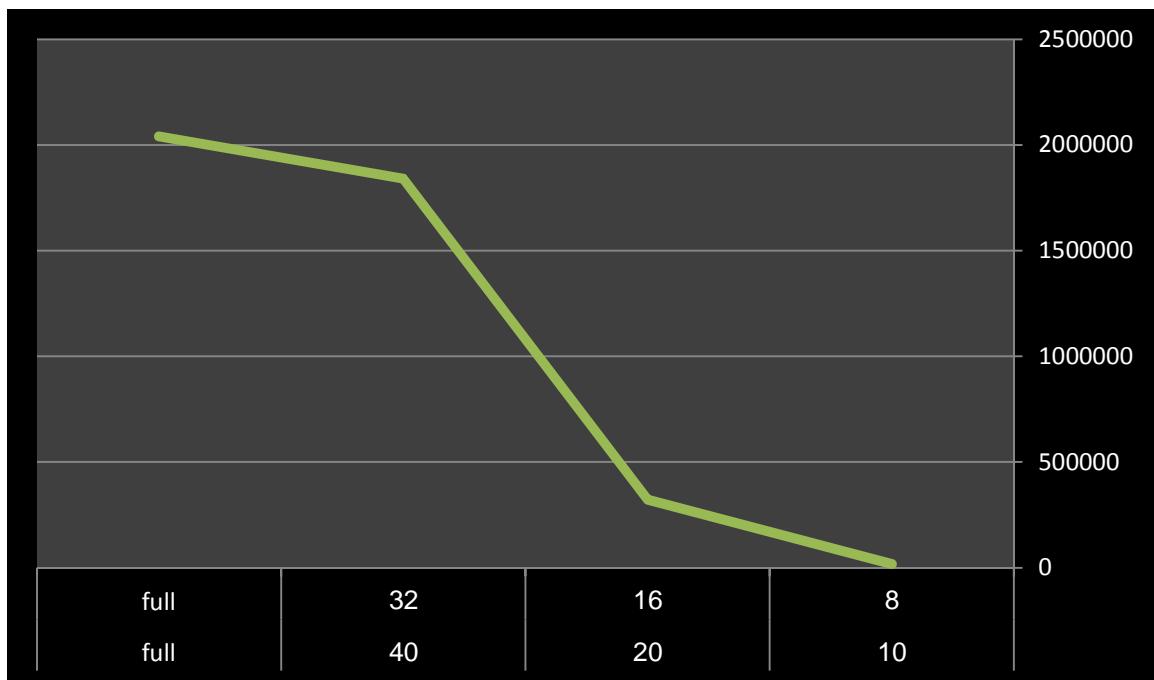
גראף 9.11 – GeForce 285GTX vs. GeForce 9600GT



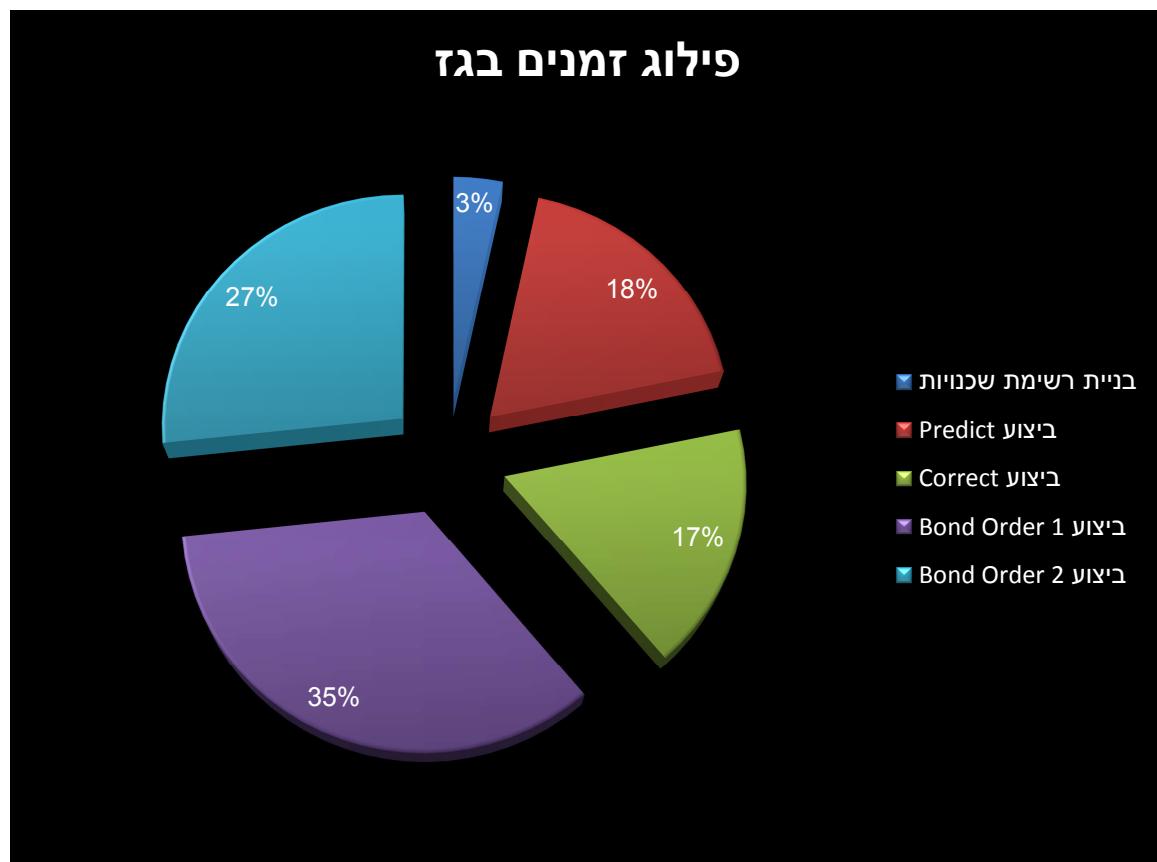
גרף 7 – השפעת רשיית השכניות בגז



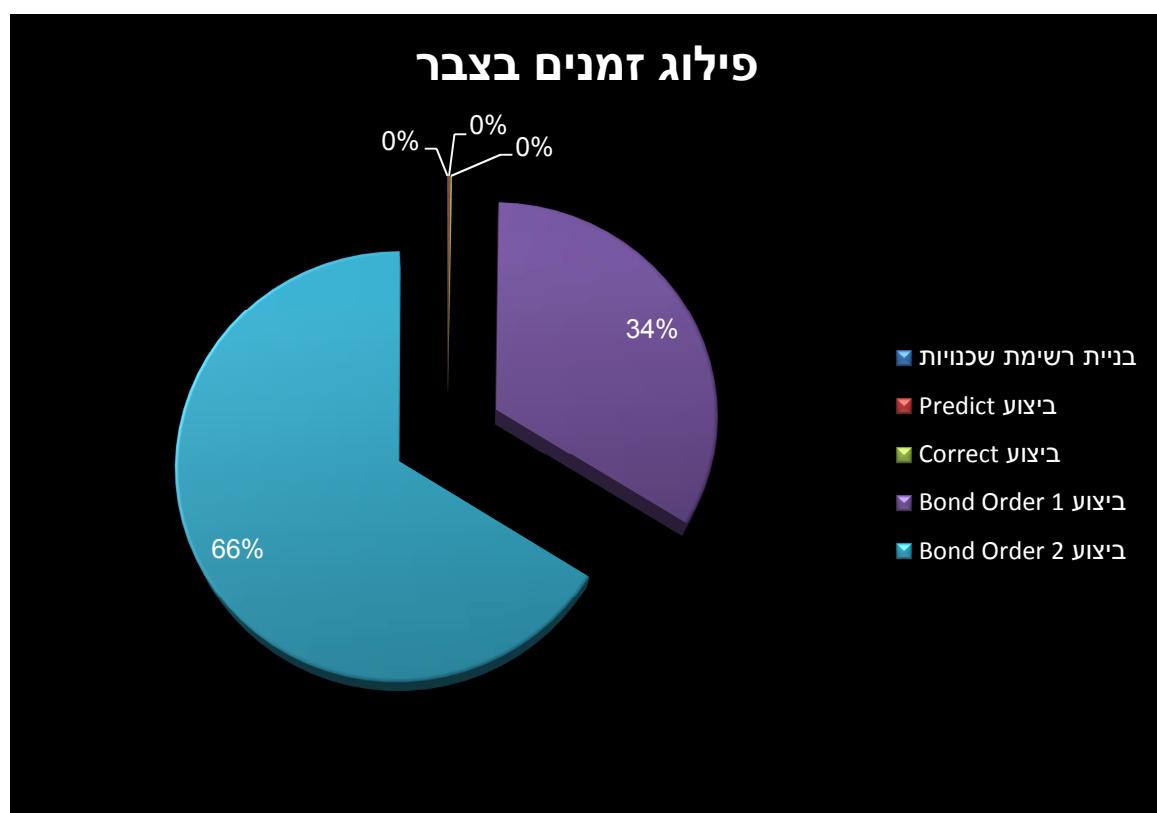
גרף 8 – השפעת רשיית השכניות בציבר בגין



9.14 גרפ 9 - פילוג זמנים בಗז

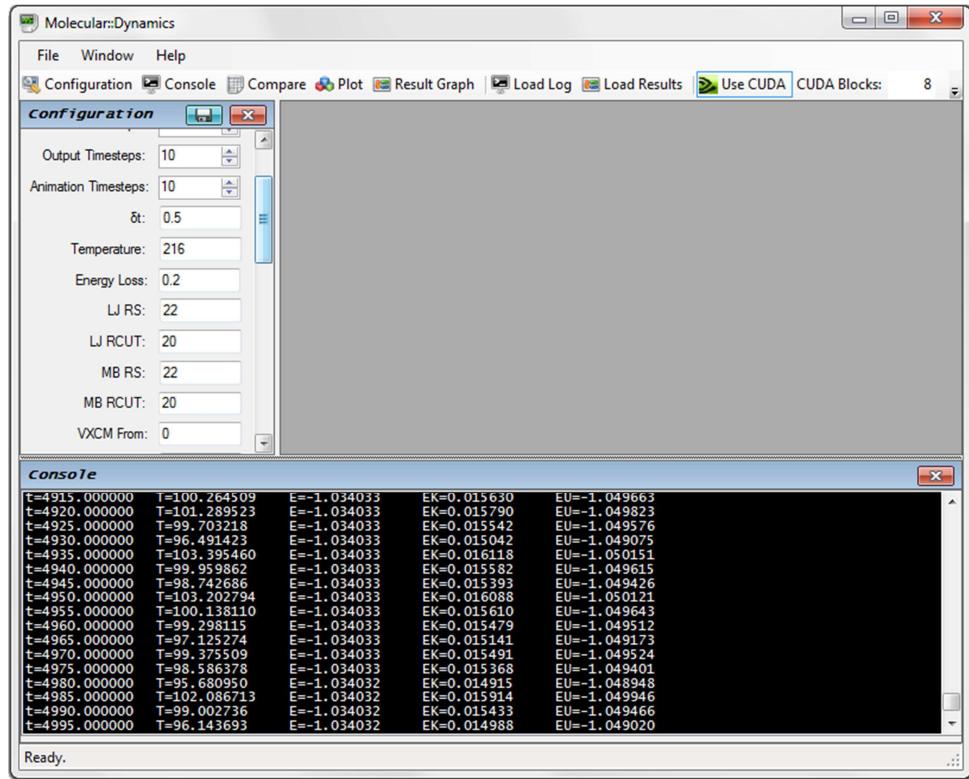


9.15 גרפ 10 - פילוג זמנים בצבר חזק

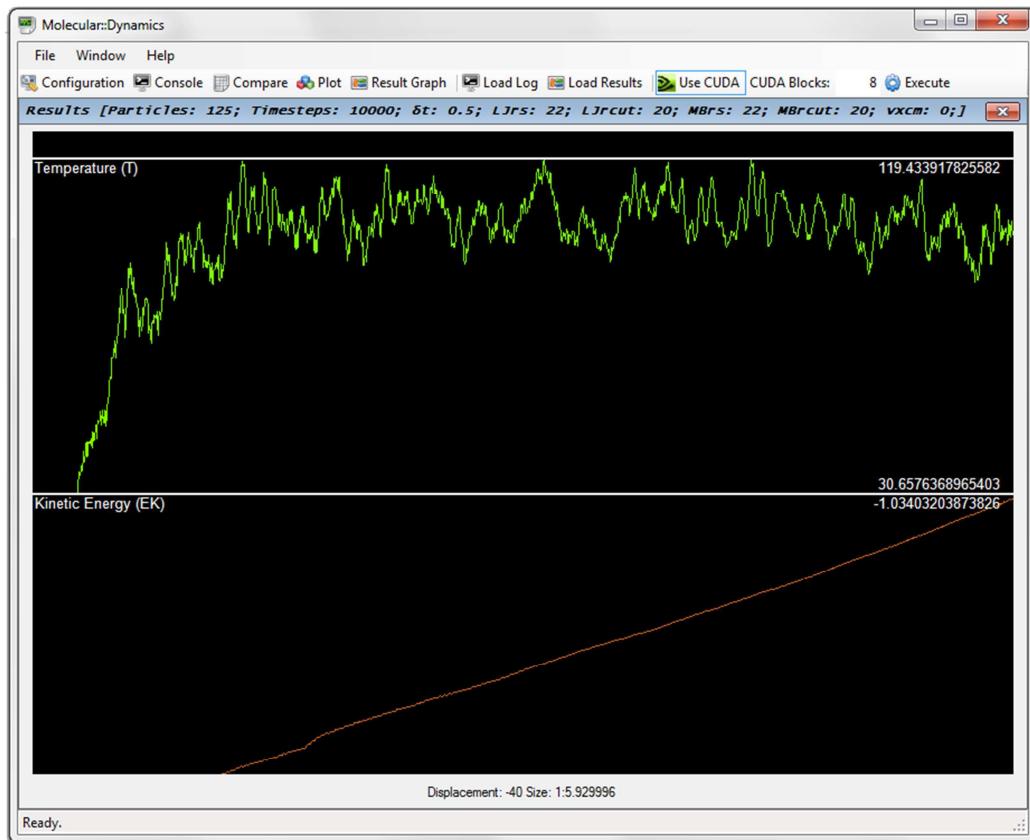


9.16 צילומי מסך של הממשק

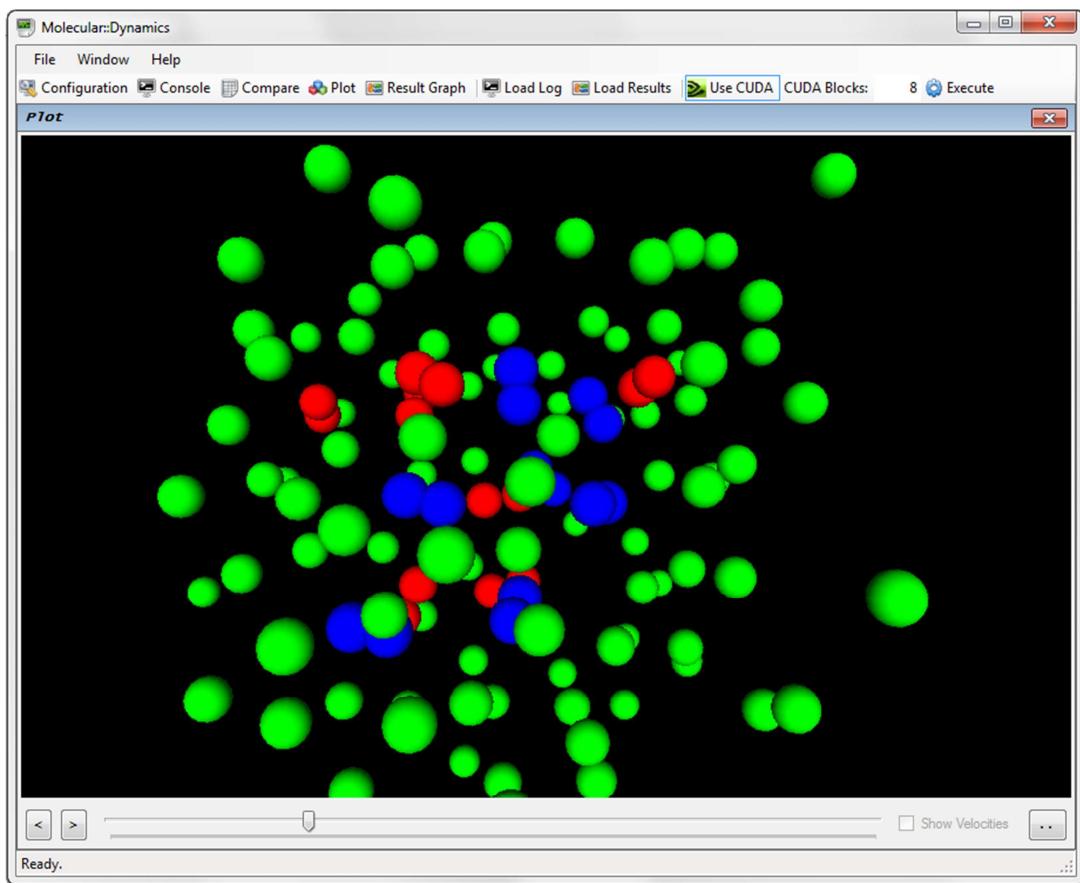
9.16.1 תצלום מסך 1 – בזמן ריצה של הסימולציה



9.16.2 תצלום מסך 2 – תצוגה גרפית



9.16.3 צלום מס' 2 – תצוגה תלת-ממדית



Abstract

The project's goal is first of all to show the basis - molecular dynamics - what is molecular dynamics, what it allows and how does it work. Explain the problems of molecular dynamics simulation - the structure of the simulation, calculating forces and integration. And finally present the implementation problems of molecular dynamics simulation as we address the simple and plain implementation as the basic idea, effective implementation using data structures, parallel implementation in general, and efficient parallel implementation on the GPU that is the extension of the general parallel implementation.

In the implementation of the simulation, two data structures were used to improve the running time, the first is a neighbor list and the other is Spatial Hash. The project shows two methods for implementation for parallel execution on the GPU, where each method its advantages and disadvantages.

The simulation is implemented in C/C++, and the GPU version is implemented using the CUDA architecture, which is a dedicated architecture for parallel execution.

Test and comparisons were made for both versions (GPU and CPU). The CPU tests were executed on Intel Core i7, and the GPU tests on NVIDIA GeForce 285GTX and GeForce 9600GT.

In addition, a comparison was made with a basic version of the simulation that was written by chemists and its run time reaches days and weeks – I will show how to use data structures, and the GPU architecture to improve the run time up to by 146 in a cluster and up to by more than 970 in gas.



SOFTWARE ENGINEERING DEPARTMENT

Molecular Dynamics Simulation on GPU

by
Vadim Kuras

Supervisor: Dr. Yehuda Hassin

Approved by the Supervisor:

Date:

Approved by the Projects' Coordinator:

Date:



SOFTWARE ENGINEERING DEPARTMENT

Molecular Dynamics Simulation on GPU

by
Vadim Kuras

July 2010

Av 5770