אוניברסיטת בן גוריון בנגב

הפקולטה להנדסה

המחלקה להנדסת חשמל ומחשבים

מבוא לעיבוד מקבילי 361.1.3621

תרגיל בית מספר 4

מגישים:

ניר שניידר 316098052

ויסאם סאלח 211734504

מרצה הקורס:

דר. גיא תל-צור

תאריך אחרון להגשה: 23/1/2023

רקע:

בתרגיל זה התבקשנו לבצע חישוב של ה-semi-empirical binding energy formula חשבת את מסת גרעין האטום כתלות במספר הנוירונים ומספרו האטומי על פי הנוסחה:

$$E_B = a_V A - a_S A^{2/3} - a_C \frac{Z(Z-1)}{A^{1/3}} - a_A \frac{(A-2Z)^2}{A} \pm \delta(A,Z)$$

את חישוב זה נבצע בעזרת HTCONDOR.

HTCondor - מאפשר ביצוע של מטלות חישוביות רבות על פני תקופת זמן ארוכה. הקונדור משתמש במשאבים זמינים על מנת לבצע חישובים במקביל באמצעות ניהול תורים.

התקשורת עם קונדור מתבצע בעיקר בעזרת קובץ Submit File, המסביר לקונדור באיזה קבצים להשתמש לצורך חישוב.

התכונה עובדת בסגנון התאמת צרכי המשתמש לתשתית הנדרשת לביצוע המשימה. ההתאמה נעשית על פי מספר רב של מאפיינים כגון: אופן השימוש במחשב, תצורת עבודה ביניהם ,יחסי עבודה בין מחשבים ועוד.

במטלה אנו משתמשים ב-universe מסוג standard, כי היא מספקת סביבת הפעלה פשוטה וקלה לשימוש להפעלת קובצי הפעלה עם single-thread שאינם דורשים סביבה או משאבים מיוחדים.

העבודה:

במטלה התבקשנו לבצע 200 הגשות ל-HTCondor כאשר כל אחת תחשב 200 נקודות. בכך נדמה ערכים של N,Z = [1, 200] לטובת 200x200 חישובים.

לבסוף יש לקרוא את ה200 קבצי פלט שנוצרו עקב ה-condor_submit ולייצר היסטוגרמה התתאר לנו את הפתרון המוכר למשוואה הנ"ל.

הסבר על הקוד:

כתבנו 2 תוכניות אחת בשפת C והשנייה בשפת פייתון: האחת תבצע את החישוב עבור 200 נקודות של הערכים N=[1, 200] ועבור Z קבוע, התוכנית השנייה תיקח את קבצי הפלט מריצת הקונדור תאגד את כלל הנקודות שנוצרו מהחישוב (סה"כ 200x200 נקודות) ותייצר היסטוגרמה של הנקודות וערכיהן.

- תוכנית ראשונה – Generate_files:

אלה הן הספריות בהן השתמשנו לצורך כתיבת הקוד:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
```

הגדרת משתנים בהתאם לכתוב בויקיפדיה תחת העמודה של Least-squares fit(1):

```
// Constants for the semi-empirical mass formula according to Least-squares fit (1) in MeV units
const double a_V = 15.8;
const double a_S = 18.3;
const double a_C = 0.714;
const double a_A = 23.2;
const double a_P = 12;
```

הגדרנו אותם כ-const מכיוון שאלו מקדמים קבועים שלא עתידים להשתנות ולא נרצה שישנו אותם.

פעולה לחישוב ערכי הנקודות N,Z בהתאם למשוואה:

```
// binding energy of N and Z
double binding_energy(int N, int Z)
{
    int A = N + Z;
    double E_B = a_V * A - a_S*pow( X: A,  Y: 2.0/3.0) - a_C*pow( X: Z,  Y: 2)/pow( X: A,  Y: 1.0/3.0) - a_A*(pow( X: A-2*Z,  Y: 2)/A) + delta_residue(A, Z);
    return E_B;
}
```

פעולה נוספת לטובת חישוב השארית דלתא בהתאם לכתוב בויקיפדיה תחת העמודה של Least-squares fit(1):

```
// compute delta(A, Z) according to the formula
double delta_residue(int A, int Z)
{
    double delta = a_P * (pow(A, 0.5));
    if (A % 2 == 0 && Z % 2 == 0)  // even-even
    {
        return delta;
    }
    else if (A % 2 != 0 && Z % 2 != 0)  // odd-odd
    {
        return -delta;
    }
    else  // even-odd, odd-even
    {
        return 0;
    }
}
```

הפעולה תקבל כקלט (מקובץ ה-submit של הקונדור) ערך עבור Z כאשר כל בכל פעם מתוך 200 הריצות ערכו ישתנה בהתאם למספר הריצה. לטובת הכנסת ערך נקלט למשתנה Z כתבנו את השורה הבאה:

```
int Z = atoi(argv[1]);  // scan value for Z
```

כמו כן הגדרנו את המערך הבא לטובת שמירת תוצאות הריצה:

```
char binding_energies[200][40];
```

כעת, בכל ריצה נרצה לחשב 200 נקודות עבור טווח ערכים של 1 עד 200 למשתנה N.

```
// loop over N with a constant Z (per run) and calculate the mass
for (N = 1; N <= 200; N++)
{
    double M = binding_energy(N, Z);
    if (M <= 0)
    {
        M = -1;
    }
    sprintf(binding_energies[N-1], "%d - %d - %f", N, Z, M);
}
```

בקוד הנ"ל ניתן לראות לולאה אשר רצה 200 פעמים. נבצע את חישוב בהתאם לפעולה binding_energy ובמידה והערך שלילי נאתחל את המשתנה M שמכיל את התוצאה למינוס 1 אשר יהווה אינדיקטור בעת קריאת הערכים (יפורט בהמשך). כל חישוב יוכנס למערך binding_energies כמחרוזת המכילה את הערכים N, Z, M אשר מופרדים ע"י ' – '.

בסוף התוכנית הדפסנו את המערך binding_energies כדי שימלאו לנו הקבצים שיוצרו על ידי הקונדור בערכים הנכונים, על מנת שנוכל להשתמש בהם אחר-כך כאשר נצייר את ההיסטוגרמה המתאימה והסופית:

```
// output
for (i = 0; i < 200; i++)
{
    printf("%s\n", binding_energies[i]);
}

return 0;
```

- תוכנית שנייה – Plot_files:

במצב זה יש לנו 200 קבצים המכילים 200 חישובי נקודות כ"א בתבנית שהוצגה לעיל.
כעת, נרצה לקרוא את הערכים הרלוונטים מקבצים אלה וליצור גרף המתאר את התוצאות.

ראשית, השתמשנו בimports הנחוצים לצורך המשימה:

```
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

רק נרחיב כי הפקודה matplotlib.use('Agg') מגדירה את הbackend של matplotlib למצב 'Agg' backend שזו לטובת הצגת גרפים בהקשרים לא אינטרקיביים כמו שמירת גרף בקובץ כפי שנתבקשנו לבצע.

הגדרנו מערכים לטובת קריאה והכנסה של הערכים המתאימים בהתאמה:

```
N = []
Z = []
M = []
```

הגדרנו מחרוזת שמהווה את התחילית הזהה לכל קובץ שנוצר לנו ע"י הקונדור:

```
# List of file names
file_name = 'job_'
```

כעת נרצה לעבור את כל ה-200 קבצים בתיקייה, לפתוח קובץ קובץ ולקורא שורה אחר שורה את התוצאות. נבצע הפרדה של כל מחרוזת בכל שורה ע"י ' – ' שהוצג לעיל ונכניס את הערכים למערכים המתאימים N,Z,M.

```
# Iterate over the file names
for i in range(200):
    # Read file
    with open(file_name + str(i) + ".out.txt") as f:
        lines = f.readlines()
        for line in lines[:-1]:
            a, b, c = line.split(" - ")
            # build arrays
            N.append(int(a))
            Z.append(int(b))
            M.append(float(c))
```

- תוכנית שלישית – submit.txt:

```
Universe=standard
Executable=Generate_files
Arguments=$(Process)
Output=job_$(Process).out.txt
Error=job_$(Process).error.txt
Log=log.txt
Queue 200
```

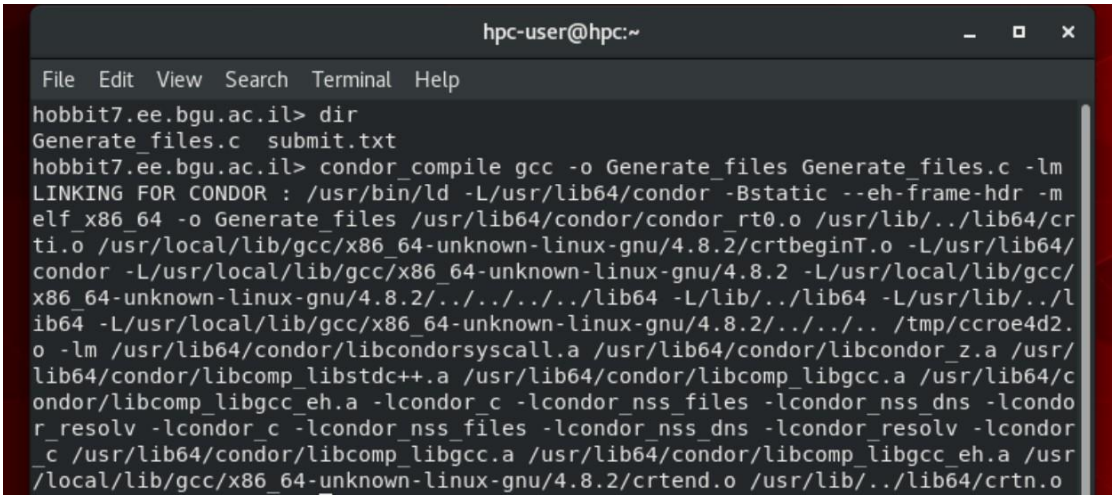
כך ש-:

- $Universe$: סביבת הביצוע, כפי שציינו קודם, בחרנו ב- standard כי היא מספקת סביבת הפעלה פשוטה וקלה לשימוש להפעלת קובצי הפעלה עם single-threaded שאינם דורשים סביבה או משאבים מיוחדים.

  חשוב לציין, בהתחלה השתמשנו ב- vanilla כי זהו סוג סטנדרטי שכאשר יש מעבר בין מחשבים החישוב מתחיל מהתחלה, אך הקבה מההרצות נפלו, לכן העדפנו להחליף ל- *standard* מהסיבות שפרטנו למעלה.
- $Executable$: שם הקובץ ההרצה שלנו בו השתמש הקונדור.
- $Arguments$: הארגומנטים שהתוכנית שלנו מקבלת, במקרה שלנו זה מספר התהליך אשר משתנה באופן סדרתי מריצה לריצה על משאבים שונים (ממספר אותם מ-0 עד 199).
- $Output$: מיקום ושם קובץ הפלטים של התוכנית.
- $Error$: מיקום ושם קובץ השגיאות של התוכנית.
- Log: מיקום ושם הקובץ שמכיל מידע על ביצוע העבודה, כולל הפלט והשגיאה הסטנדרטים.
- $Queue$: מספר הפעמים שהעבודה צריכה להתבצע, במקרה שלנו, התבקשנו לבצע את העבודה 200 פעמים.

הרצת התוכנית:

כפי שהזכרנו קודם, השתמשנו ב- $Universe = standard$, והלכנו לפי ה- HTCondor Tutorial שגיא העלה לאתר הקורס, בצילומי מסך הבאים נראה בדיוק איך הרצנו:



```
hobbit7.ee.bgu.ac.il> ls -lh Generate_files
-rwxr-xr-x 1 wesams pp2023 6.5M Jan 23 22:34 Generate_files*
hobbit7.ee.bgu.ac.il> strip Generate_files
hobbit7.ee.bgu.ac.il> ls -lh Generate_files
-rwxr-xr-x 1 wesams pp2023 1.6M Jan 23 22:35 Generate_files*
```

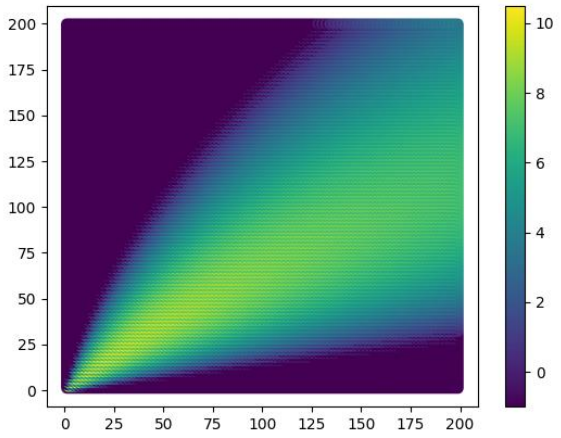
```
hobbit7.ee.bgu.ac.il> condor_submit submit.txt
Submitting job(s).....................................................................
.....................................................................................
.....................................................
200 job(s) submitted to cluster 2904.
```

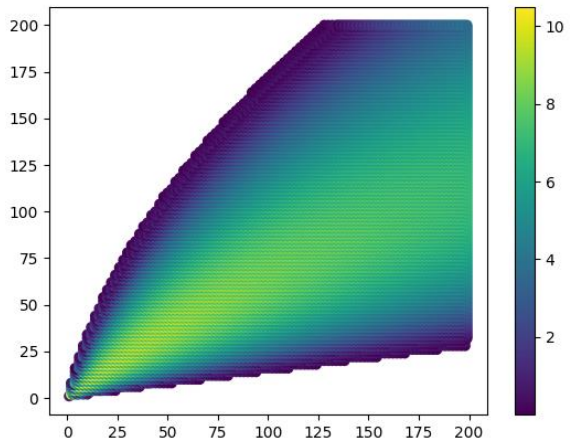
ניתן לראות שהתבצעו 200 משימות תחת cluster יחיד עם מס' זיהוי 2904.

וכך יצרנו (לפי הקוד submit.txt שלנו) 200 קבצים ל- output, 200 קבצים ל- error וקובץ אחד ל- Log.

ניתוח תוצאות:

כפי שהובטח, נסביר מדוע היינו צריכים לסמן את הערכים השליליים. הערכים השליליים בסופו של דבר מהווים את הרקע לגרף התוצאה כפי שניתן לראות כאן:





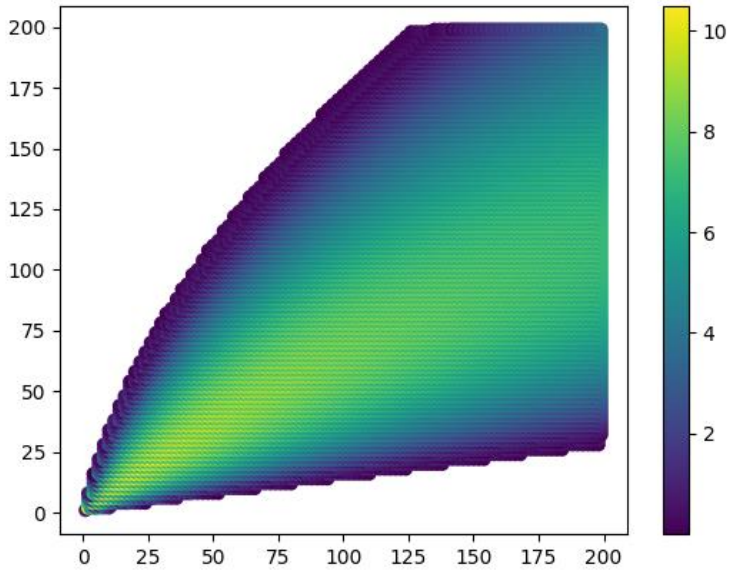
תהליך זה בוצע ע"י השורה הבאה בקובץ Plot_files:

```
# remove background
M = [np.nan if x == -1 else x for x in M]
```

לבסוף כל שנותר לבצע הוא הדפסת הגרף הנ"ל ע"פ הערכים שקיבלנו ושמירתו:

```
# save plot
plt.scatter(N, Z, c=M)
plt.colorbar()
plt.savefig('/users/agnon/pp2023/nirschne/HW4/out/Plot.png')
```

לבסוף קיבלנו את התוצאה הבאה:



- נריץ את הפקודה $condor\_q$:

- מטרת פקודה זו לראות מצב תור העבודות.

בצילום מסך הבאה רואים איך הרצנו את הפקודה $condor\_q$, בנוסף רואים שהתור מלא אך לא מעבודות שלנו, אלא מסטודנטים אחרים, ה- OWNER אצלנו בשם "wesams", נראה בהמשך את הסינון שעשינו כדי להראות את התור עבור המשתמש שלנו.

תמונה לתחילת התור:

```
hobbit7.ee.bgu.ac.il> condor_q


-- Submitter: hobbit7.ee.bgu.ac.il : <132.72.50.35:55305> : hobbit7.ee.bgu.ac.il
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
2827.0   yaakovme        6/6  14:23   0+00:00:02 H  0   0.0  mandelbrot.m
2828.0   yaakovme        6/6  14:36   0+00:00:01 H  0   0.0  mandelbrot.m 20
2829.0   yaakovme        6/6  14:37   0+00:00:01 H  0   0.0  mandelbrot.m 20
2831.0   yaakovme        6/6  14:58   0+00:00:02 H  0   1.5  mandelbrot.exe 20
2832.0   yaakovme        6/6  15:01   0+00:00:02 H  0   1.5  mandelbrot.exe 20
```

תמונה לסוף התור:

```
2894.46  tzuria          6/23 19:42   0+00:00:01 H  0   0.0  run.sh 46
2894.47  tzuria          6/23 19:42   0+00:00:00 H  0   0.0  run.sh 47
2894.48  tzuria          6/23 19:42   0+00:00:00 H  0   0.0  run.sh 48
2894.49  tzuria          6/23 19:42   0+00:00:00 H  0   0.0  run.sh 49

358 jobs; 0 completed, 0 removed, 200 idle, 0 running, 158 held, 0 suspended
```

ביצענו סינון כדי לקבל רק את העבודות מהמשתמש שלנו:

```
hobbit7.ee.bgu.ac.il> condor_q -constraint 'Owner == "wesams"'


-- Submitter: hobbit7.ee.bgu.ac.il : <132.72.50.35:55305> : hobbit7.ee.bgu.ac.il
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
hobbit7.ee.bgu.ac.il>
```

וניתן לראות שהתור ריק, כלומר כל העבודות הסתיימו.

- נריץ את הפקודה $condor\_history$:
  - מטרת פקודה זו לקבל מידע על העבודות שהושלמו.

```
hobbit7.ee.bgu.ac.il> condor_history -constraint 'Owner == "wesams"'
 ID      OWNER            SUBMITTED     RUN_TIME ST   COMPLETED CMD
2904.199 wesams          1/23 22:37   0+00:00:00 C   1/23 22:37 /users/agnon/pp
2904.198 wesams          1/23 22:37   0+00:00:00 C   1/23 22:37 /users/agnon/pp
2904.197 wesams          1/23 22:37   0+00:00:00 C   1/23 22:37 /users/agnon/pp
2904.196 wesams          1/23 22:37   0+00:00:00 C   1/23 22:37 /users/agnon/pp
2904.195 wesams          1/23 22:37   0+00:00:00 C   1/23 22:37 /users/agnon/pp
2904.194 wesams          1/23 22:37   0+00:00:01 C   1/23 22:37 /users/agnon/pp
2904.193 wesams          1/23 22:37   0+00:00:01 C   1/23 22:37 /users/agnon/pp
2904.192 wesams          1/23 22:37   0+00:00:01 C   1/23 22:37 /users/agnon/pp
2904.191 wesams          1/23 22:37   0+00:00:01 C   1/23 22:37 /users/agnon/pp
2904.190 wesams          1/23 22:37   0+00:00:01 C   1/23 22:37 /users/agnon/pp
2904.189 wesams          1/23 22:37   0+00:00:01 C   1/23 22:37 /users/agnon/pp
```

ניתן לראות בצילום המסך חלק מהעבודות שהושלמו על ידי המשתמש שלנו ("wesams") עם מספר הזיהוי המתאים.

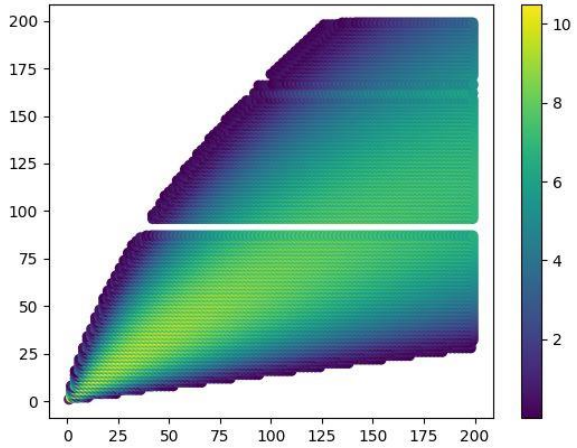
מסקנות ואתגרים:

- **מסקנות:**

- במהלך הרצה השתמשנו בכוח החישובי של ההוביטים כדי לייעל את הרצה בעזרת מאגר חישובי גדול יותר.
- נוכחנו לראות כי אכן מערכת הקונדור הינה מערכת יעילה מכוון שמאפשרת לנו לבצע חישוב של מטלות רבות על יד שימוש במשאבים של מחשבים אחרים (שאפשרו את הגישה אליהם) או הוביטים שונים. כך נוכל לבצע חישובים גדולים שלוקחים זמן רב באופן יעיל.
- במשימה שלנו החישובים היו בלתי תלויים אחד בשני, ולכן יכלנו להריץ 200 תהליכים שונים כך שכל תהליך ביצע את חישובו בנפרד והוציא קובץ output משלו שאינו תלוי בתוצאה של התהליך האחר.

- **אתגרים:**

תחילה השתמשנו בקובץ הקונדור ב-vanilla universe אך קיבלנו שוב ושוב את התוצאה הבאה:



הדבר הגיוני מכיוון שכפי שנאמר בהרצאות, ישנם 200 מחשבים התומכים בקונדור באוניברסיטה ומכיוון שעלינו להריץ 200 פעולות סביר מאוד כי יהיו משאבים אשר לא ניתנים לשימוש. כאשר ניסינו לבצע את ההחלפה ל-standard universe עבור התוכנית המקורית שהייתה רשומה בפייתון קיבלנו את השגיאה הבאה:

```
Submitting job(s)
ERROR: You are trying to submit a "standard" job to Condor. However, this instal
lation of Condor does not support the Standard Universe.
$CondorVersion: 8.6.11 Aug 10 2018 BuildID: RH-8.6.11-1.el6 $
$CondorPlatform: X86_64-RedHat_6.10 $
```

בשל כך החלטנו להמיר את התוכנית Generate_files לתוכנית C ובכך הצלחנו להשמיש את ה-standard universe ולהתגבר על הבעיה ובכך לקבלת את התוצאה הרצויה כפי שהצגנו קודם לכן.