**פרויקט סופי קורס מחשוב מקבילי**

יוצר: עמית גולדשטיין.

בפרויקט הסופי של הקורס נדרש לכתוב תוכנית אשר מפענחת טקסט מוצפן בעזרת שימוש בטכנולוגיות:

OpenMP, MPI, CUDA.

בתוכנית שלי החלטתי להשתמש רק ב:

**OpenMP & MPI**

התוכנית מקבלת כקלט:

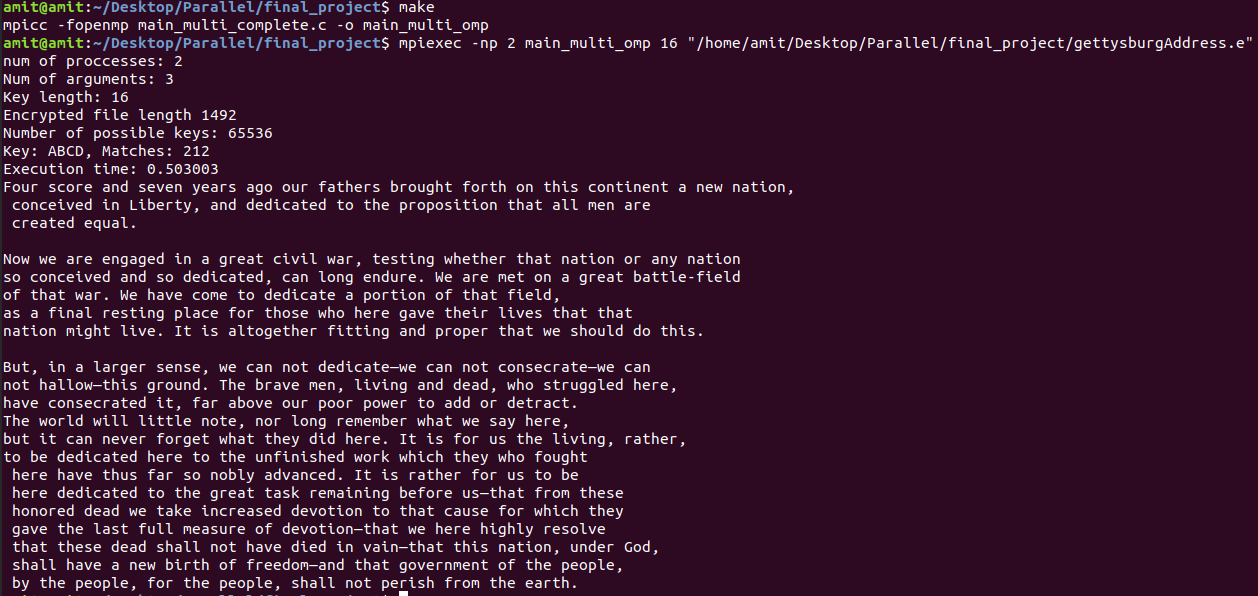
1) אורך מפתח הצפנה (כמספר סיביות, כלומר בייט אחד = שני הקסא = 8 סיביות)

2) מיקום קובץ המכיל את הטקסט המוצפן

3) קובץ המכיל מילים שצפויות להיות בטקסט הלא מוצפן (מילה אחת בכל שורה)

במקרה ולא מצורף קובץ המכיל מילים צפויות, התוכנה תשתמש במילון ברירת המחדל של לינוקס (אמריקאי אנגלי)

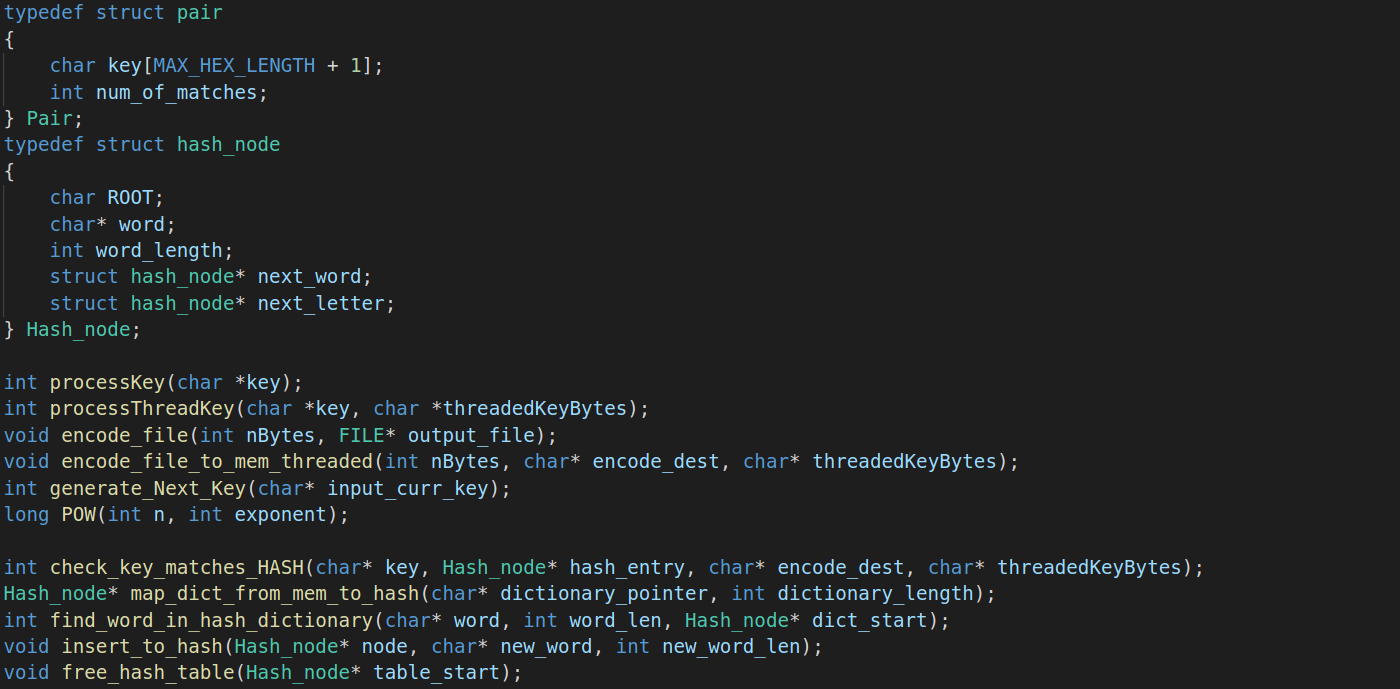
דוגמא ליצירה ולקריאה לתוכנה (במקרה זה עם מפתח באורך 2 בייטים, ללא קובץ מילון כאשר למכונה מוקצאים 2 ליבות מעבד:



**הסבר על אופן הפעולה של התוכנית**

התוכנית מחלקת את העבודה לתהליכים וחוטים, כל תהליך מקבל כמות שווה של מפתחות לבדוק וכל כמות כזאת מתחלקת לחוטים של אותו תהליך.

בדיקה של מילה נתונה האם היא צפויה נעשית בעזרת הפונק' אשר עובר על מילון שמופה לטבלאת האש, היא קודם מגיעה לרשימה מקושרת של האות ההתחלתית ועוברת על כל המילים המתחילות באותה אות, בודקת האם האורך של המילים זהה, אם כן, בודקת תו תו.

התוכנית מכילה את הפונקציות והמבנים הבאים:

**תפקידו של המבנה Pair** הוא להחזיק מפתח מסוים אשר שימש לפענוח הקובץ המוצפן ואת מספר המילים שפוענחו בעזרתו שהתאימו לקובץ המילון שסופק.

**תפקידו של המבנה Hash\_node הוא** לספק מבנה נתונים בשביל ליצור טבלאת Hash לקובץ המילון הנתון בכדי לייעל את האופן בו התוכנית מחפשת בו האם מילה שפונעחה נמצאית או לא נמצאית בו.

במבנה הזה יש משתנה ROOT אשר מתאר איזו אות מתחילה את המילה של הNODE, יש משתנה word אשר מכיל את המילה של NODE, יש משתנה word\_length אשר מכיל את אורכה, יש מצביע לNODE בשם next\_word אשר מצביע אל הNODE הבא ברשימה המקושרת **של אותה אות התחלתית** ויש מצביע next\_letter אשר מצביע אל הNODE של המילה הראשונה של האות ההתחלתית הבאה בAlphabet

**הפונקצייה hex2int** (לא מופיע בתמונה) נלקחה מהשלד שסופק במודל, תפקידה להמיר מערך של אותיות הקסא (בצורת char) לערכם המספרי (int)

**הפונקצייה processKey** לוקחת מפתח הצפנה של תווים הקסאדצימאליים בצורה של מערך תווים (char) וממירה אותם למערך אחד של סיביות שיישמש לפעולת הצפנה\פענוח בעזרת XOR. (הפונק' הזו נלקחה מהשלד שניתן במודל) הפונק' בנוסף מחזירה את מספר הבייטים שיש במפתח ההצפנה

**הפונק' processThreadKey** בדומה לprocessKey לוקחת מפתח הצפנה של תווים וממירה אותם לסיביות, ההבדל היחיד הוא שהיא כותבת לתוך משתנה שהיא מקבלת במקום לkeyBytes נתון ומשותף.

**הפונק' encode\_file** תפקידה לקחת את מפתח ההצפנה שנוצר מפונק' processKey ונשמר בkeyBytes ולפענח את הקובץ המוצפן השמור בזכרון לפי הפרמטר שהיא קיבלה (אם לא ניתן פרמטר לאן לשמור את הפענוח היא שומרת אותו בזכרון בתוך משתנה temp\_file\_in\_memory), בסוף התוכנית משתמשים בפונק' בשביל להוציא את הפענוח "הכי טוב" שנמצא אל stdout.

**הפונק' encode\_file\_to\_mem\_threaded** תפקידה הוא לשמש את החוטים השונים בפענוח הקובץ המוצפן, עקב והחוטים שונים צריכים זכרון נפרד אחד מהשני לכתוב אליו את הפענוח של המפתח (כי לכל חוט מפתח שונה ומפתחות של חוטים אינם חופפים) הפונק' הזו בדומה לencode\_file מפענחת אך היא מקבלת כפרמטר מצביע למערך תווים (שכל חוט מקצה לעצמו) במקום לקובץ.

**הפונק’ generate\_Next\_Key** מקבלת בתור פרמטר מפתח נוכחי (בצורה של מערך תווים) ומקדמת אותו לתו האפשרי הבא בהתאם לאורך מפתח ההצפנה, כלומר אם אורך מפתח ההצפנה 8 סיביות והפונק' קיבלה את המפתח "0A" כפרמטר, היא תקדם את אותו מפתח (בזכרון) ל"0B", ותחזיר 1, כלומר שהצליחה.

כאשר אם היא תקבל את המפתח "FF" שהוא האחרון האפשרי, היא תקדם אותו ל "00" ותחזיר 0, כי המפתח התאפס, הערך מוחזר משמש כדי לדעת מתי הגעת לאפשרות האחרונה\נכשלה ההתקדמות בכדי שלא תווצר לולאה אינסופית של קידומים.

**הפונק' POW** מטרתה לחשב חזקה, זה בא לידי שימוש בכדי לחשב את מספר המפתחות האפשריים המקסימאלי, אשר מהווה תנאי עצירה בשביל התהליכים אשר פועלים כל אחד לבדו מבלי לעצור אחד את השני (פחות סנכרון בין התהליכים = פחות השהיות).

**הפונק' find\_word\_in\_hash\_dictionary** היא הפונק' ה"כבדה" ביותר בתוכנית, והיא מבצעת את עיקר העבודה הדרושה, לבדוק האם מילה נתונה נמצאית במילון הנתון (בצורה של טבלאת האש).

הפונק' קודם בודקת האם המילה שניתנה כפרמטר תקינה באופן בסיסי, כלומר האם אורכה שונה מ0, האם המילה שניתנה היא NULL והאם המילון שניתן הוא NULL.

לאחר מכן היא בודקת את המילה שניתנה בUPPERCASE (כי לפעמים מילים במילון הם גם UPPER וגם LOWER וזה לא בהכרח מוצא אם הם לא בדיוק אותו דבר) ונכנסת לטבלאת ההאש, דבר ראשון שהיא עושה זה לקדם את הHASH NODE אל האות ההתחלתית הנכונה, כלומר אם המילה שניתנה היא "ORANGE" המילון ידלג בבת אחת על כל המילים שמתחילות באותיות שבאות לפני האות "O" מבלי לבדוק אותם, זה בעזרת מעבר אל הNODE המתחיל את הרשימה המקושרת של המילים המתחילות באות "O".

כאשר יגיע לרשימה המקושרת הרלוונטית אופן ההשוואה הוא קודם כל לבדוק האם האורך של המילה שניתנה כפרמטר והאורך של המילה הנוכחית ברשימה המקושרת של המילון שווים, אם לא מדלגים, אם כן אז מעלים דגל ל-1 ועוברים תו תו (הבדיקה של תו תו מתבצעת **בעזרת OMP** מכיוון שאין DATA RACE אז ניתן להשתמש בOMP פה), אם תו כלשהו שונה בין המילים הדגל יורד ל-0 ויוצאים מהלולאה של המילה הנוכחית במילון, בסוף כל לולאה של מילה ברשימה המקושרת של המילון יש בדיקה האם הדגל מורם (כלומר האם מילה "שרדה" בדיקה של תו תו), אם הוא מורם הפונק' מחזירה 1 כי הצליחה למצוא התאמה למילה במילון.

אם לא נמצאה התאמה עד שמגיעים למילה האחרונה ברשימה המקושרת של האות ההתחלתית שנמצאה בהתחלה, מוחזר ערך 0 כי לא נמצאה התאמה למילים במילון.

**הפונק' check\_key\_matches\_HASH** מקבלת מפתח הצפנה וכניסה לטבלאת האש של המילון שעליה צריך לעבור, כתובת זכרון אליה יפוענח הטקסט שלאחר מכן ייבדק, וכתובת מערך תווים אשר יכיל את התוצר של עיבוד המפתח שכרגע בודקים, היא מפעילה את הפונק' processThreadKey בשביל לעבד את מפתח ההצפנה שכרגע בודקים ולקבל את מספר הבייטים שלו, היא לאחר מכן מפעיל את encode\_file\_to\_mem עם פרמטר output\_file=NULL בכדי שהפענוח יהיה אל תוך זכרון שכתובתו נשמרת במשתנה decode\_dest, היא לאחר מכן עוברת על התווים בזכרון ומנסה ליצור מילים שלמות (כלומר מילה שנגמרת בתו רווח, בתו שורה חדשה או בnull terminator.

בנוסף לכך במהלך יצירת המילה היא גם מחשבת את אורכה, לאחר סיום יצירת מילה, היא מעבירה אותה אל find\_word\_in\_hash\_dictionary עם המילה שנוצרה + אורכה + הכניסה למילון כפרמטרים, אם מקבלת 1 היא מעלה את המונה הזמני של מספר ההתאמות למפתח שקיבלה, ובסופו של דבר כאשר הגיע לסוף הקובץ המפוענח, מחזירה את המונה הזה.

**הפונק' map\_dict\_from\_mem\_to\_hash** היא פונק' שמטרתה לקחת את המילון שמופה מקובץ אל הזכרון (בצורה של מערך אחד ארוך של CHAR באמצעות MMAP) וליצור בעזרתו טבלאת HASH אשר תשמש לחיפוש יעיל של מילים בתוכו.

אופן הפעולה של הפונק' הנ"ל הוא: להקצות זכרון לכניסה ראשונית של Hash\_Node אל תוך משתנה בשם table\_start להשים אותו במשתנה זמני current\_node ואז ליצור את הכניסות הראשונות של כל האותיות בAlphabet

לאחר מכן הוא עובר על המילון הממופה בזכרון ומתחיל לבנות מילים (באותו זמן מונה את אורכן) בעזרת מעבר על הזכרון תו תו עד קריאה של תו שורה חדשה או NULL TERMINATOR הוא ממשיך לעשות את זה בהתאמה לכניסות של טבלאת ההאש לפי הROOT של המבנה, כלומר מילים שמתחילות ב"O" יהיו ברשימה מקושרת של Hash\_node בעלי ROOT = "O".

בסיום יצירת טבלאת ההאש מוחזרת הכתובת של הכניסה הראשונה של הטבלה table\_start(זו שנוצרה בהתחלה).

**פונקציית ה-Main**

בתחילת התוכנית לפני המקבול שלה לתהליכים שונים מתבצעים כמה דברים:

1. קליטת מספר סיביות מפתח ההצפנה בעזרת atoi אל תוך משתנה key\_bit\_length
2. מציאת מספר ספרות ההקסא לפי חלוקה של key\_bit\_length וזה נשמר בnum\_of\_hex\_letters
3. יצירת טיפוס MPI חדש בשם PairMPI אשר מכיל מידע בדומה לטיפוס Pair שהוגדר קודם.

לאחר השלבים הללו מתבצע אתחול המקבול למספר התהליכים (כפי שהתוכנה מורצת, התוכנה לא תלויה במספר התהליכים, היא תעבוד גם אם תורץ על תהליך יחיד וגם אם תורץ על מספר תהליכים)

כל תהליך מקצה זכרון למבנה Pair (זה ישמש לכל תהליך לשמור מקומית את המפתח "הטוב ביותר" שלו ובסוף התוכנית ישמש כדי למצוא את הטוב מבין כל התהליכים)

תהליך הMASTER (דרגה 0) מבצע קריאה של הנתונים מהcommand line argument, מדפיס את מספר התהליכים שהורצו בתוכנית, מספר הארגומנטים שהועברו לתוכנית ובהתאם למספר הארגומנטים שניתנו מחליט בין אם לפתוח את המילון ברירת מחדל או את המילון שניתן לו כארגומנט.

לאחר ההחלטה אם לפתוח את המילון ברירת מחדל או את האופציונלי, הוא פותח file descriptors עבורם ומפעיל fstat בשביל לקבל מידע עליהם (כאשר גודל הקובץ הוא מה שצריך מזה), מדפיס את אורך הקובץ המוצפן ואז מפעיל את הפונק' **mmap** בשביל למפות את הקבצים הללו אל תוך הזכרון (בשביל גישה מהירה).

לאחר שתהליך הMASTER סיים לקרוא את הקבצים לזכרון, הוא משדר את אורכם לשאר התהליכים בעזרת Bcast, (כולם צריכים שיהיה להם את הקובץ המוצפן בכדי שיוכלו באופן עצמאי לפענח כל פעם עם מפתח שונה),

לאחר השידור מתבצע סנכרון בשביל לוודא שכולם הגיעו לנק' שיש להם גם את הקובץ המוצפן וגם את המילון.

כל התהליכים לאחר מכן מקצים זכרון שיאחסן בכל פעם את הפענוח הנוכחי שהם עובדים עליו, ואז כל העבדים (לא הMASTER) מקצים זכרון שיאחסן את הקובץ המוצפן ואת המילון בזכרון (ההקצאה מתבצעת פעם אחת מכיוון שקבצים אלו לא משתנים).

לאחר ההקצאה מתבצע סנכרון נוסף ואז הMASTER משדר לעבדים את הקובץ המוצפן והמילון שמופו לזכרון עם **mmap**.

עכשיו כל תהליך (כולל הMASTER) משתמשים בפונק' map\_dict\_from\_mem\_to\_hash בכדי ליצור לעצמם את טבלאות הHash למילונים שלהם (בעייתי להעביר טבלאת האש בין התהליכים לכן החלטתי להעביר מיפוי של MMAP בין התהליכים ואז לאפשר להם ליצור טבלאת HASH מזכרון מאשר מקריאה מדיסק).

לאחר יצירת טבלאת הHASH למילון (שנוצרת רק פעם אחת כי המילון לא משתנה) כל תהליך מקצה זכרון למערך current\_Key אשר יאחסן את המפתח הנוכחי שהם יעבדו עליו ומאפסים את המערך הזה (בשביל לקבל מפתח התחלתי שממנו החוטים בתהליכים יתקדמו למקומם הראוי)

כל תהליך מחשב את מספר המפתחות האפשריים לפי מספר אותיות ההקסא של המפתח (זה ישמש לתנאי עצירה), בנוסף מקצים זכרון למערך best\_key שם יישמר המפתח הכי טוב שהתהליך הצליח למצוא ובצמוד אליו מאותחל משתנה שנקרא best\_key\_hits אשר ישמור את מספר ההתאמות של המפתח הכי טוב הנוכחי.

**תחילת ה עבודה הכבידה ב Main**

קודם כל שומרים את מספר החוטים המקסימלי האפשרי בעזרת omp\_get\_max\_threads לתוך משתנה max\_thread\_num, ושומרים במשתנה step\_size את החישוב של max\_thread\_num \* num\_procs (מספר חוטים מקסימלי כפול מספר התהליכים שמריצים את התוכנית), זה יאפשר לנו לדעת כמה פעמים צריך לקדם את המפתח בשביל להגיע למפתח הבא הרלוונטי אל החוט שמתקדם.

בנוסף יוצרים מערך של הטיפוס Pair עם מספר תאים כמספר החוטים האפשריים (המערך הזה ישמש לנו לאסוף את התוצאות הכי טובות שכל חוט מצא, בדומה לפעולה של Gather של MPI).

תחילת הקבלה של חוטים – כל החוטים חולקים את המשתנים הבאים: מספר החוטים המקסימלי, כתובת הכניסה לטבלאת ההאש של המילון, כתובת המערך המכיל את הקובץ המוצפן שמופה בהתחלה, וכתובת למערך הPair.

בתחילת לולאת הOMP מוגדר משתנה היכיל את מספר החוט (my\_tid), משתני איטרציה iterator, p

ודגל שמצהיר על כך שחוט סיים לעבור על כל המפתחות האפשריים עבורו בשם completion\_flag

בנוסף לכך מוקצים מערכים threadedKeyBytes אשר יכיל את הסיביות של המפתח לאחר העיבוד שלו, מערך decoded\_file\_dest אשר לתוכו יישמר כל פעם הפענוח הנוכחי ועליו עוברים בשביל למצוא התאמות, מערך best\_threaded\_key אשר יכיל את המפתח הכי טוב של חוט מסויים, וthreaded\_key אשר יכיל את המפתח הנוכחי שחוט מסויים כרגע בודק.

מוגדרים גם כן 2 אינטג'רים best\_threaded\_key\_matches, threaded\_key\_matches אשר ישמרו כמה התאמות יש למפתח.

תחילה כל חוט מעתיק את המפתח current\_Key אל המערך האישי שלו threaded\_key עליו יעבוד.

אז מתחילה לולאה שרצה על iterator מאפס עד לmy\_rank + num\_procs \* my\_tid ובכל איטרציה של הלולאה החוט מקדם את המפתח האישי שלו, המטרה של הלולאה הזו היא לגרום לכך שכל חוט יתקדם למקום התחלתי בסדרת המפתחות ככה שלא יהיו שני חוטים בעלי אותו מפתח (מכיוון שלאחר מכן כל חוט "קופץ מעל" מספר קבוע של מפתחות, אם המפתחות ההתחלתיים יהיו שונים אז בחיים לא יחפפו אחד לשני)

לאחר מכן כל חוט בודק האם עבר את מספר המפתחות האפשריים, לדוג' אם יש רק 4 מפתחות אך יש לנו 2 תהליכים עם 4 חוטים כל אחד, כל חוט שהiterator שלו עבר את 4 בלולאה הקודמת שקידמה כל חוט, אז אותם חוטים יפסיקו את הפעולה שלהם ולא יבצעו שום חיפוש במילון.

אם החוט לא עבר את המספר האפשרי אז מועתק המפתח הנוכחי (לאחר הקידום שלו) אל תוך best\_threaded\_key ובנוסף החוט מבצע בדיקה במילון למצוא התאמות על המפתח הזה ושומר את זה בתוך best\_threaded\_key\_matches.

לאחר מכן מתחילה הלולאה העיקרית שבתוכה יעברו על כל המפתחות האפשריים, תנאי העצירה של הלולאת WHILE הוא שהמשתנה iterator קטן ממספר המפתחות האפשריים ושהדגל completion\_flag שווה אפס,

למקרה שלא אמרתי, בעת קידום מפתח אם הפונק' generate\_Next\_Key מקדם את המפתח חזרה ל0 היא מחזירה 0 וזה מעלה את הדגל completion flag של אותו חוט.

בכל תחילת לולאה החוט מקדם את המפתח שלו step\_size פעמים ואז בודק האם עבר את הגבול של מספר המפתחות האפשריים, אם לא הוא בודק את המפתח לראות כמה התאמות יש לו, אם יש לו יותר מהקודם הכי טוב עד כה, הוא מעדכן.

בסופו של דבר לאחר סיום כל המפתחות, כל חוט שומר במערך המשותף threads\_best\_pair\_array במיקום הmy\_tid (כלומר כל חוט שומר במקום משלו) את המפתח הכי טוב שמצא ומחכה ששאר החוטים יסיימו גם לפני יציאה מהלולאה של הOMP.

אחרי היציאה מהלולאת OMP כל תהליך מפעיל לולאת FOR בשביל לעבור על המערך המשותף של החוטים שלו כדי למצוא את המפתח הטוב מביניהם, אותו הוא שומר ב myPair אשר יועבר אחרי זה בGather אל תהליך ה MASTER

לאחר שכל התהליכים סיימו למצוא את ההכי טוב והעבירו בGather את התוצאה שלהם לMASTER

הMASTER מוצא את המפתח הכי טוב מכל התהליכים, מעבד אותו, משתמש בו לפענח את הקובץ וכותב אותו לSTDOUT בנוסף מדפיס את הזמן שלקח להריץ + הזוג (מפתח + התאמות).