# README

פרטי מגישים :

ים חורין ת.ז 318853256

## גרסה הסטטית

התוכנית מתחילה באתחול MPI וקביעת הדרגה הנוכחית והמספר הכולל של תהליכים .המספר Nוהאפסילון של סף ההתכנסות מסופקים כארגומנטים של שורת פקודה או מוגדרים לערכי ברירת מחדל .עומס העבודה מתחלק בין התהליכים, כאשר כל תהליך אחראי לחישוב השורש הריבועי לטווח מסוים של מספרים .מבחן הפונקציות נקרא עבור כל תהליך, אשר מבצע את שיטת האנפה עבור כל המספרים בטווח שהוקצה לו .הוא סופר את מספר האטרקציות הנדרשות להתכנסות ומזהה את המספר עם האטרקציות המקסימליות .לאחר שכל תהליך סיים את חלקו, התוצאות מצטמצמות לתהליך המאסטר (דרגה 0) באמצעות .MPI_Reduce לאחר מכן, תהליך המאסטר מוצא את המספר עם האטרקציות המקסימליות בין כל התהליכים .לבסוף, התוכנית מדפיסה את המספר שדרש את המספר המרבי של אטרקציות ואת הזמן הכולל שלוקח לחישוב.

int test(int n_start, int n_end, int rank, double epsilon, int *min_number_max_counter):פונקציה זו נקראת על ידי כל תהליך כדי לבצע את השיטה של Heronעבור מספרים בטווח שצוין( n_start) עד .(n_end )הוא מחזיר את המספר המרבי של איטרציות בין כל המספרים בטווח שלו .המשתנה min_number_ מתעדכן במספר שדרש את האיטרציות המקסימליות.

כתוצאה מכך, כל תהליך מטפל במספר שווה בערך של מספרים, בהתאם לערך של num_procs וטווח המספרים מ-4 עד N. זה מבטיח שעומס העבודה החישובי יתחלק באופן שווה בין התהליכים שאינם מאסטר.

עומס מתחלק באופן שווה באופן הבא :

לדוגמא עם N = 100 ויש חמישה תהליכים :

תהליך 0 יהיה המנהל

תהליך 1 יקבל את הטווח 1-25

תהליך 2 יקבל את הטווח 25-50

תהליך 3 יקבל את הטווח 50-75

תהליך 4 יקבל את הטווח 75-100

## גרסה הדינמית

התוכנית מתחילה באתחול MPI וקביעת הדרגה הנוכחית והמספר הכולל של תהליכים . הארגומנטים של שורת הפקודה N, epsilon ו chunk_size-מעובדים .אם לא מסופק, נעשה שימוש בערכי ברירת מחדל .תהליך המאסטר (דרגה 0) מחלק את עומס העבודה הכולל (N) לנתחים קטנים יותר בהתבסס על chunk_size ומקצה אותם לתהליכי עבודה באמצעות MPI_Send.כל תהליך עובד (דירוג 0 <מקבל נתח לעבוד עליו מהמאסטר באמצעות MPI_Recv ומחיל את השיטה של Heron כדי להעריך את השורש הריבועי של כל מספר בנתח . תהליך העבודה מחשב את מספר האיטרציות הנדרשות עבור כל מספר ומזהה את המספר שדרש

את האיטרציות המקסימליות כדי להתכנס לשורש הריבועי .העובד שולח את התוצאה (מספר האיטרציות והמספר המתאים) בחזרה למאסטר באמצעות MPI_Send עם התג DONE. המאסטר מקבל את התוצאות מהעובדים, מעדכן את התוצאה הגלובלית שלו ושולח עוד עבודה לעובדים בטלנים (אם זמינים) עד להשלמת כל המשימות .כאשר אין עוד עבודה לשלוח, המאסטר שולח הודעת STOP לתהליכי העבודה באמצעות MPI_Send עם מטען נתונים ריק כדי לאותת שהם צריכים להסתיים .לבסוף, תהליך המאסטר מדפיס את המספר שדרש את המספר המרבי של איטרציות ואת הזמן הכולל שלוקח לחישוב.

int test(int n_start, int n_end, int rank, double epsilon, int *min_number_max_counter): פונקציה זו נקראת על ידי כל תהליך עובד כדי לבצע את השיטה של Heron עבור מספרים בטווח שצוין (n_start ל n_end-). הוא מחזיר את המספר המרבי של איטרציות בין כל המספרים בטווח שלו .המשתנה min_number_max_counter מתעדכן במספר שדרש את האיטרציות המקסימליות.

התוכנית מאזנת אוטומטית את עומס העבודה בין תהליכי העבודה על ידי חלוקת המספרים לנתחים .עם זאת, לביצועים מיטביים, ייתכן שיהיה עליך להתאים את chunk_size בהתבסס על הארכיטקטורה של המערכת והמשאבים הזמינים .הדיוק של קירוב השורש הריבועי תלוי בערך האפסילון .ערכים קטנים יותר של אפסילון יובילו לתוצאות מדויקות יותר, אך עשויים לדרוש יותר איטרציות וכתוצאה מכך, זמן חישוב רב יותר .לעומת זאת, ערכים גדולים יותר של אפסילון יביאו לחישובים מהירים יותר אך עם דיוק מופחת .התוכנית תוכננה עבור מספרים ממשיים לא שליליים .ייתכן שהוא לא יפיק תוצאות משמעותיות עבור מספרים שליליים.

לנוחות יש MAKEFILE לתוכניות כולן עם תכונה של לנקות את כל האובייקטים שנוצרו

תוצאות לדוגמה (ssqrt – סדרתית , dsqrt – דינאמי , nsqrt – סטטי )

שימו לב שב nsqrt אין תמיכה למספר תהליכים כי מדובר על תהליך אחד בלבד

גרסה שהתוכנית הסטטית עובדת יותר מהר (ssqrt) :

N = 10000 , epsilon = 0.01 chunk size = 10 num_processes = 5

```
(base) linuxu@ParallelC23-21:~/c$ mpiexec -n 5 ./ssqrt 10000

number requiring max number of iterations : 4592 (number of iterations : 7)
sequential time: 0.003278 secs
(base) linuxu@ParallelC23-21:~/c$ mpiexec -n 5 ./dsqrt 10000 0.01 10

number requiring max number of iterations : 4592 (number of iterations : 7)
sequential time: 0.050739 secs
(base) linuxu@ParallelC23-21:~/c$ mpiexec ./nsqrt 10000

number requiring max number of iterations : 4592 (number of iterations : 7)
sequential time: 0.000635 secs
(base) linuxu@ParallelC23-21:~/c$
```

גרסה שהתוכנית הדינאמית עובדת יותר מהר (dsqrt) :

N = 100000 , epsilon = 0.8 chunk size = 30 num_processes = 6

```
(base) linuxu@ParallelC23-10:~/yam$ mpiexec -n 6 ./dsqrt 100000 0.8 3000

number requiring max number of iterations : 42136 (number of iterations : 8)
sequential time: 0.009318 secs
(base) linuxu@ParallelC23-10:~/yam$ mpiexec -n 6 ./sqrt 100000 0.8

number requiring max number of iterations : 42136 (number of iterations : 8)
sequential time: 0.001416 secs
(base) linuxu@ParallelC23-10:~/yam$ mpiexec ./ssqrt 100000 0.8

number requiring max number of iterations : 42136 (number of iterations : 8)
sequential time: 0.006057 secs
```