Yonathan Bettan

302279138

yonibettan@gmail.com

Alon kwart

201025228

Alon.kwart@gmail.com

תיאור הפתרון:

כנדרש בתרגיל מימשנו 2 פונקציות:

- Simple_parallel_walsh()
- Fast_parallel_walsh()

הפונקציה simple_parallel_walsh() מומשה בצורה הסטנדרטית של הכפלת ווקטור במטריצה נתונה, אשר מחושבת תוך כדי ריצה מפאת גודלה.

בדרך זו אנו יוצרים עותק של הווקטור כך שנוכל לכתוב את התוצאה לווקטור הקלט. לאחר שלב זה מחלקים את התכנית לחוטים כך שכל חוט אחראי על הכפל שורה יחידה במטריצה בווקטור העמודה ולכתוב את התוצאה למקום המתאים בווקטור התוצאה (שהיה בעבר ווקטור הקלט)

מכיוון שכל החוטים מבצעים קריאה בלבד מווקטור הקלט אין צורך באמצעי סנכרון כלל על ווקטור זה.

ברגע שכל החוטים סיימו הווקטור שנשלח ככקלט כעת מכיל את ווקטור התוצאה. משחררים את הזיכרון שהוקצה ומסיימים.

הפונקציה fast_parallel_walsh() הינה מתוחכמת יותר ומחולקת לשלבים.

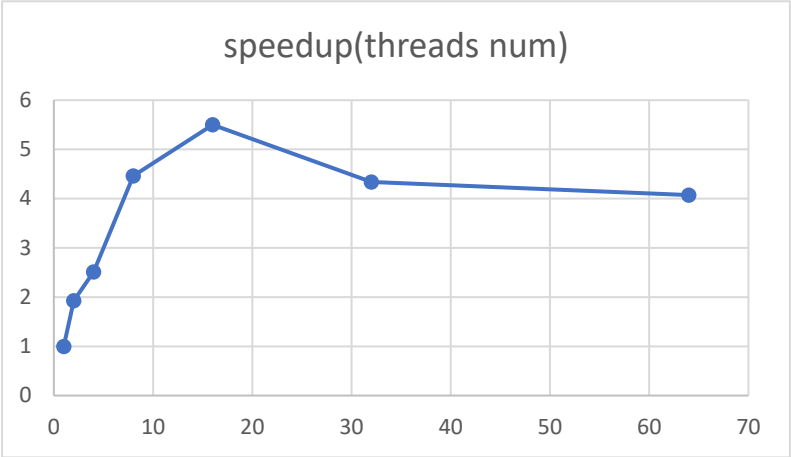
השלב הראשון של הפונקציה נקרא reorganize_input_vector() ותפקידו לבצע את FFT בצורה מקבילית. החיסרון של פונקציה זו שהיא איטית עבור חלוקה לווקטורים קטנים מדי אך יעילה למדי עבור ווקטורים גדולים, ולכן תנאי העצירה של פונקציה זו היא כאשר חילקנו את הווקטור למספר ווקטורים המתאים למספר החוטים שהוקצו לנו. לדוגמא עבור 16 חוטים הפונקציה תבצע 4 איטראציות שבסופם יהיו "16 תתי וקטורים" שעברו את 4 השלבים הראשונים בלבד.

השלב הבסיסי ביותר הוא שלב סריאלי לחלוטין בשם fast_serial_walsh(), התפקיד הלוגי של שלב זה הוא פעולה סריאלית של חוט יחיד ע"י FFT, כלומר חלוקת הווקטור ל-2 בכל שלב כך שכל איבר בווקטור מקבל את הסכום\ההפרש של 2 האיברים המתאימים לו בחצאי הווקטור הרלוונטים.

לסיכום הפונקציה הראשית מפעילה את reorganize_input_vector() עבור השלבים הראשונים של האלגוריתם וברגע שהגענו לגדלים המתאימים לניצול מקסימלי של החוטים שיש לנו נפעיל עבור כל ווקטור בצורה מקבילה את הפונקציה fast_serial_walsh().

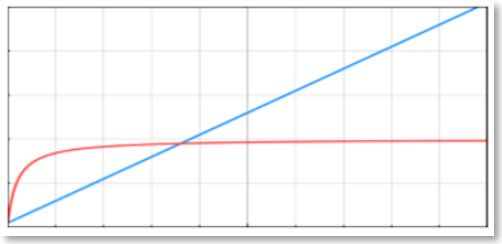
שאלות תיאורטיות:

.1



a. ניתן לראות שה- speedup המקסימלי מתקבל עבור 16 חוטים, ולאחר פגיעה קטנה בביצועים עבור 32 חוטים "מתאזן" ל- speedup מסוים.
מכיוון שיש לנו 32 מעבדים שהינם hyper threading (כלומר 16 מעבדים שלכל אחד 2 hw threads) הביצועים הטובים ביותר מתקבלים עבור ניצול מלא של כל המעבדים בשרת.
אמנם היה ניתן לחשוב שעבור שימוש בכל ה-32 חוטים ישופרו ביצועים עוד יותר אך נזכור שעבור כל החלפת הקשר בין 2 חוטים חומרה במעבד יחיד יתכנו פסילות במטמון ולכן בסופו של דבר החלפת חוט במעבד כתוצאה מ-miss במטמון קרוב אמנם החליפה חוט ש"תמסך" את זמן הגישה לזיכרון אך הנ"ל יצר עוד misses במטמון עקב capacity\conflict miss ולכן במצטבר פגע בביצועים.
עבור 64 חוטים המצב המתואר רק מחמיר ולבסוף מתאזן כפי שניתן להבחין.

b. התוצאות שקיבלנו אכן מתאימות לחוק אמדל כפי שנלמד בהרצאה (העקום האדום) וזאת מכיוון שבדקנו את הביצועים עבור ווקטור בגודל מסוים ולכן חוק אמדל טוען שישנו גבול לשיפור שכן בשלב כלשהו ה- overhead גובר על המקביליות **כאשר גודל הבעיה קבוע**



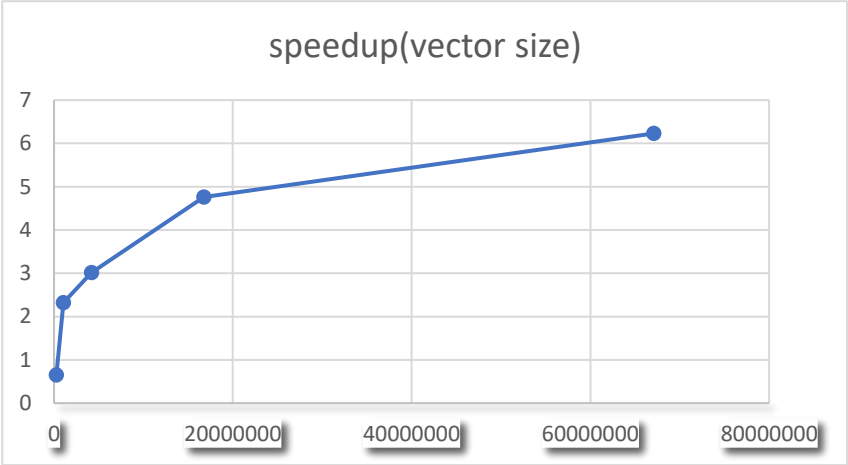
c. הגרף אינו לינארי שכן שישנו overhead של יצירת חוטים, סינכרון ע"י barriers ואילוצים נוספים כמו איכות הפתרון – יתכן כי קיים פתרון יעיל יותר שינצל את המטמונים בצורה טובה יותר מאשר המימוש שלנו

d. כפי שהסברנו בחלק הנראשון יתכנו מספר סיבות לכך:

- ישנם רק 16 מעבדים בשרת (לכל אחד מהם 2 חוטי חומרה) ולכן עבור יותר מחוט יחיד לכל מעבד עלול להגדיל את ה- missRate של המטמונים ובכך לפגוע בביצועים
- החלפת הקשר הינה פעולה יקרה מאוד וגוררת פסילת ה-TLB, פסילת המטמונים, שמירת ההקשר הישן וכ"ו ולכן עבור ריבוי בהחלפות הקשר נשלם overhead גבוהה
- גם ללא החלפות הקשר מיותרות עבור חלוקה לבעיות קטנות מדיי ה- overhead הבסיסי של החלפת הקשר עלול להיות יקר מדי בהשוואה לגודל הבעיה עבור חוט יחיד, לדוגמא עבור חישוב סכום של 10 מספרים לא נרצה לייצר 10 חוטים שכן התקורה תגבר על הרווח המקבילי

- עבור הבעיה שלנו קיימת הפרדה ברורה בין המידע שכל חוט ניגש אליו ולכן עבור ניהול נכון של מטמונים ו- prefetch ניתן היה לא להגיע לכל החסרונות האלו... לצערנו אנחנו לא הצלחנו להגיע לביצועים מושלמים אלו.

.2



a. ניתן לראות שככל שאנו מגדילים את הבעיה כך ה- speedup שלנו גדל **אך לא בצורה לינארית כפי שאולי היינו מצפים** בעקבות חוק גוסטפסון

b. התוצאה תואמת לחוק גוסטפסון שכן חוק גוסטפסון טוען שעבור הגדלת הבעיה **כך שהחלק הסדרתי אינו משתנה** הוספת יותר ליבות תגרום לשיפור לינארי בביצועים.
במימוש שלנו אנחנו מוגבלים למספר ליבות מסוים והגרף חושב עבור 16 ליבות בלבד ולכן הגדלת הבעיה בשלב כלשהו גוררת את הגדלת החלק הסדרתי.
לא עמדנו בתנאי משפט גוסטפסון ולכן אין סיבה שנקבל את התוצאות הצפויות ממשפט זה (למרות שהדבר יתכן).
לא סתרנו את משפט גוסטפסון והתוצאות הינן כמצופה.

c. כפי שהוסבר, אם היה לנו מספר בלתי מוגבל של מעבדים אכן היינו מקבלים גרף לינארי ב-speedup כפונקציה של גודל הבעיה.
במידה והמימוש שלנו כולל חלק סדרתי משמעותי הגדלת הבעיה תשפר רק את החלק המקבילי אך תגדיל את החלק הסדרתי עד אשר הוא יהפוך לדומיננטי. במקרה זה נצפה לפגיעה ממש בביצועים

.3

a. החלק המקבילי הינו שילוב של הפונקציות reorganize_input_vector() ו- fast_serial_walsh() כך שביחד עומק הרקורסיה הינה $\log_2 N$ ובכל שלב אנו עוברים על $\frac{N}{2}$ איברים ומבצעים עליהם 7 פעולות אריתמטיות. סה"כ נקבל $3.5 \cdot N \log_2 N$
עבור החלק הסדרתי יש לנו $3+(1+2+4+8+16)=34$

b. אנחנו מאמינים שהכוונה הייתה לגדלים $2^{18}, 2^{20}, 2^{22}$

החלק הסדרתי הינו $A(n)=\frac{3.5n\log n}{3.5n\log n+34}$ לכן נקבל

$$A(2^{18})=0.9999979$$
$$A(2^{20})=0.9999995$$
$$A(2^{22})=0.99999989$$

לפי הנוסחאות

$$speedup_{amdal}(\#CPUs)=\frac{1}{\frac{A}{\#CPUs}+1-A}$$
$$speedup_{gustafson}(\#CPUs)=\#CPUs+(\#CPUs-1)(1-A)$$

נקבל

$$speedup_{amdal}\left(A=2^{18},\#CPUs=64\right)=63.9915$$
$$speedup_{amdal}\left(A=2^{20},\#CPUs=64\right)=63.9979$$
$$speedup_{amdal}\left(A=2^{22},\#CPUs=64\right)=63.9995$$
$$speedup_{gustafson}\left(A=2^{18},\#CPUs=64\right)=64.00013$$
$$speedup_{gustafson}\left(A=2^{20},\#CPUs=64\right)=64.000031$$
$$speedup_{gustafson}\left(A=2^{22},\#CPUs=64\right)=64.000000693$$

c. התוצאות שקיבלנו

$$speedup\left(2^{18}\right)=4.33$$
$$speedup\left(2^{20}\right)=6.44$$
$$speedup\left(2^{22}\right)=5.39$$

ניתן לראות כי אנחנו לא קרובים לתוצאות האופטימליות.

d. הסיבות שלא קיבלנו את אותן תוצאות הינן גם מכיוון שאנחנו לא מריצים על 64 ליבות אלא על 16 כמו שנתבקשנו בתרגיל (ובהתאם להגבלות השרת) וגם כי הרצנו 8 חוטים למרות שיש לנו 16 ליבות וגם כי המימוש שלנו אינו אופטימלי וגם מכיוון שהגדלים הנתונים הינם גדולים מאוד ביחס למטמונים של המעבדים בשרת

.4

a. מכיוון שמדובר בקורס בו מניחים שרוב המשימות ירוצו במקביל הגיוני לצפות שנרצה שכל ליבה תהיה כמה שיותר עצמאית על מנת לטפל בחלקים הסדרתיים ביעילות. נרצה שליבות אלו יכילו חזאי קפיצות. מטמונים ומערכת OOO לשיפור ביצועים

b. עפ"י הנוסחה בתרגולים

$$speedup_{adymetric}\left(A,n,r\right)=\frac{T_{BCE-serial}=1}{\frac{1-A}{perf\left(r\right)}+\frac{A}{perf\left(r\right)+n-r}}\underset{perf(r)\approx\sqrt{r}}{=}\frac{1}{\frac{0.1}{\sqrt{r}}+\frac{0.9}{\sqrt{r}+32-r}}$$

המקסימום מתקבל עבור $r=11.28$ ולכן נבחר ערך שלם המוביל למקסימום ונקבל $speedup\left(r=11\right)=14.9$

c. עפ"י הנוסחה בתרגולים $speedup_{dynamic}\left(A,n,r\right)=\frac{T_{BCE-serial}=1}{\frac{1-A}{perf\left(r\right)}+\frac{A}{n}}$ מדובר בפונקציה מונוטונית עולה ממש

כפונקציה של r ולכן המקסימום המתקבל $speedup\left(r=32\right)=\frac{1}{\frac{0.1}{\sqrt{32}}+\frac{0.9}{32}}=21.8$