הרצת DoctorantMemroy עם CPU2017

1. הורד את הפרויקט DoctorantMemory מ- <https://github.com/dbiton/DoctorantMemory>
2. הורד את DynamoRIO-Linux-10 מ- <https://dynamorio.org/page_releases.html>
3. הורד את cpu2017-1.1.9.iso מ- <https://www.spec.org/cpu2017/>, או ממקור כלשהו אחר
4. מקם את DynamoRIO-Linux-10 ואת SPEC בתיקייה של DoctorantMemory, באותה התיקייה כמו doctorant\_memory.py.
5. התחברות ל-catalina1.cs.technion.ac.il באמצעות MobaXterm והעתק את התיקייה של DoctorantMemory.
6. העתק את אותה התיקייה ל- tiran1.cs.technion.ac.ilבאמצעות scp מתוך catalina1, בצורה הבאה:

scp DoctorantMemory user@tiran1.cs.technion.ac.il: DoctorantMemory

1. התחבר ל-tiran1 באמצעות ssh בצורה הבאה: ssh user@tiran1.cs.technion.ac.il
2. חלץ את DynamoRIO-Linux-10.tar.gz באמצעות הפקודה הבאה:

tar -xvf DynamoRIO-Linux-10.0.0.tar.gz

1. הרצנו את הפקודה הבאה בשביל לאפשר הרצה של DoctorantMemory:

chmod +x doctorant\_memory.py

1. במקרה שלי, הייתה חסרה ספרייה נדרשת שקשורה לכיווץ קבצים, ולכן הייתי צריך להריץ:

sudo apt-get install libsnappy-dev

כאשר הופיע מסך ורוד ששאל האם אפשר לאתחל תהליכים מסוימים, בחרתי באופציה שהאתחול יבוצע במועד מאוחר יותר, והתוכנה הותקנה.

1. בצע את הפקודה הבאה בשביל לטעון את SPEC

sudo mount -t iso9660 -o ro,exec,loop cpu2017-1.1.9.iso /mnt

1. התקן את התוכנה באמצעות:

cd /mnt

ואז

./install.sh

כאשר תשאל איפה להתקין את התוכנה, בחר

/home/user

1. נכנסתי לתיקייה שבה הותקנה התוכנה

cd /home/user

ואז הרצתי

source shrc

1. נכנסתי לתיקייה config ושם הרצתי:

cp Example-gcc-linux-x86.cfg DoctorantConfig.cfg

אחרי זה נכנסתי לקובץ DoctorantConfig.cfg ושם הוספתי בשורה 152 את השורה:

% define gcc\_dir “/usr”

זה צריך להיות לפני השורה

% ifndef %{gcc\_dir}

מאחר שהמיקום ברירת המחדל עבור המהדר אינו נכון עבור השרת שלנו.

1. בניתי את ה-suite שמעניין אותנו, integer rate, באמצעות הפקודה הבאה:

runcpu --config=DoctorantConfig --action=build intrate

1. בחרתי להריץ את ה-benchmark בשם leela\_r. ה-benchmark הזה משחק גו, וניצח בתחרות ב-2008. הפקודה הבאה מביאה אותנו לתיקייה הרלוונטית:

go leela\_r exe

ועל מנת להריץ את התוכנה מהתיקייה הנ"ל אנחנו משתמשים בפקודה:

./leela \_r\_peak.mytest-m64 ../data/test/input/test.sgf

יש 3 סטים של נתונים שאפשר לרוץ איתם – test, train, refrate, בתיקיות תואמות בתוך התיקייה data. test מסיים לרוץ לאחר 5 שניות, ובחרנו בו.

על מנת לייצר את ה-trace עבור ה-benchmark הנ"ל, חזרנו ל-~, שם העתקנו את DoctorantMemory, והרצנו את הפקודה הבאה, אשר דרשה 2 דקות על מנת לסיים לרוץ:

python3 doctorant\_memory.py -operation generate -app\_path ~/ben chspec/CPU/541.leela \_r/exe/leela\_r\_peak.mytest-m64 -app\_args ~/benchspec /CPU/541.leela\_r/data/test/input/test.sgf

1. כעת כשיש לנו טרייס כנדרש, אנחנו יכולים להריץ עליו את הכלים שלנו, למשל, נריץ את סימולטור הקאש בצורה הבאה:

python3 doctorant\_memory.py -operation parse -trace\_path TRACEPATH -parse\_tool\_name cache\_simulator

שים לב שיש להחליף את שם תיקיית הטרייס בשם שנוצר עבור הטרייס שלך. מאחר שהשם מבוסס על חתימת זמן, השם יהיה שונה.

מאחר שהטסטים ב-CPU2017 יוצרים המון גישות לזיכרון, נרצה להגביל את מספר הפקודות שאנו מעבדים כאשר אנחנו מריצים את memory\_accesses או drcachesim\_memory\_accesses. כאשר הרצתי את memory\_accesses ללא הגבלה על הטרייס הנ"ל, נוצר קובץ בגודל 300 ג'יגהבייט עד שהתמלא כל האחסון בשרת. נוכל לעשות זאת, למשל, בצורה הבאה:

python3 doctorant\_memory.py -operation generate -app\_path ~/benchspec/CPU/541.leela\_r/exe/leela\_r\_peak.mytest-m64 -app\_args ~/benchspec/CPU/541.leela\_r/data/test/input/test.sgf -additional\_options "-exit\_after\_tracing 1000000"

על מנת לייצר טרייס באורך מיליון פקודות.

דבר נוסף שמומלץ לעשות הוא לבחור ערך סביר עבור ה-alignment של כתובות זיכרון אשר מחושבות בשביל hot\_addresses, לדוגמה 64. המשמעות היא שאנו מעגלים את 64 הביטים הנמוכים ביותר כאשר אנחנו ממיינים את הכתובות הפופולריות, כדי לצמצם את מספר הכתובות שאנחנו עוברים עליהם.

פקודה כנ"ל לדוגמה תראה כך:

python3 doctorant\_memory.py -operation parse -trace\_path TRACEPATH -parse\_tool\_name memory\_accesses -parse\_alignment\_size 32

אם נתקלים בבעיות מוזרות, נסו להגביל את מספר הפקודות ש-parse מעבד עם exit\_after\_tracing.