מקרה בדיקה 2: בדיקת sanity

עבור המקרה מטה בחרנו שורה של צעדים שיגרמו לקומבינציות שונות של hit / miss בכל אחד מרמות הcache. וידאנו שבעת פגיעה מתעדכן המקום המתאים בcache ע"פ הלוגיקה שלו (Direct mapped לL1, 2-way associative לL2). כמו כן וידאנו את המצבים בהם כל אחת מרמות הcache מבצעת flush לרמה הבאה או אל הזכרון הראשי.

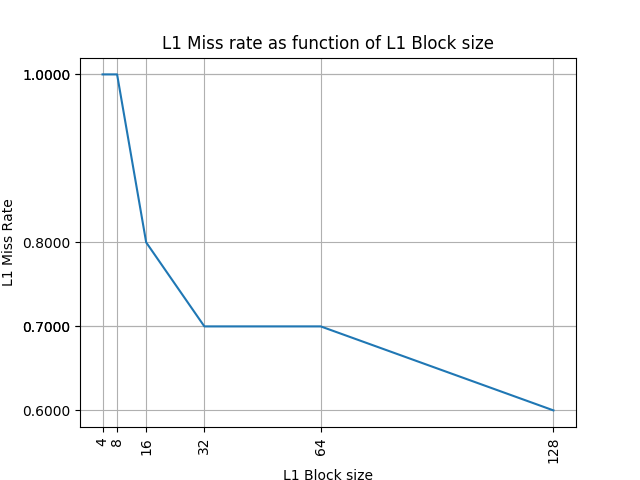
בעמודת הHit / Miss מטה ניתן לראות מה מצופה בדיוק מכל שלב בתוכנית לבדוק.

השתמשנו בגדלי בלוקים של 128 עבור L1 ו512 עבור L2 כיוון שהם גדולים יותר מרוחב הbus ולכן נדרשים מחזורים נוספים בכדי להעביר את המידע בין הזכרון ושתי רמות הזכרון. באופן זה ניתן לבדוק את המנגנון אשר מונה כמה מחזורים לקחה התוכנית.

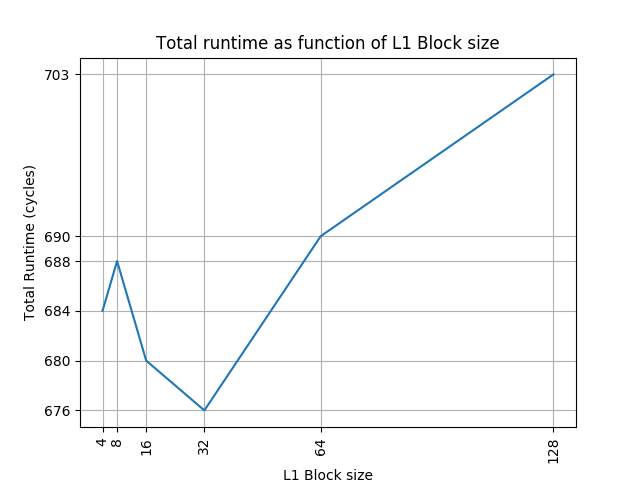
השתמשנו גם בoffsetים שונים בפקודות בתוכנית. על אף שהם לא משפיעים על הtag שיתאחסן שבכל אחד מהתאים בcache, הם מאפשרים לנו לבצע מעקב ולוודא שאכן הdata נכתב בצורה נכונה לקבצי הפלט.

כל אחד משלבי הסימולציה מפורט מטה (באופן בינארי, כדי להקל על המעקב אחר כל אחד מחלקי הכתובת); ביצענו אותם ידנית תחילה ווידאנו שהסימולטור אכן עובד כמצופה (ואכן הנתונים בקבצי הפלט של הסימולטור תואמים את התוצאות שלנו מהחישוב הידני).

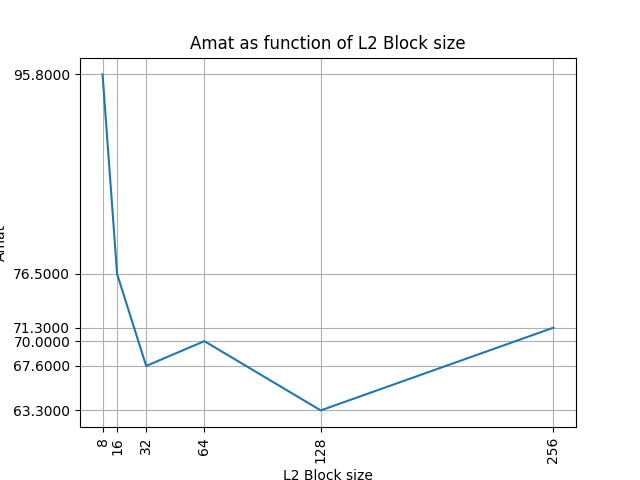
**הגרפים אותם קיבלנו:**



התוכנית פונה לכתובות הנמצאים במרווחים קטנים וגדולים (תכננו אותה כך שתתאים למקרים בהם הבלוק של L1 הוא בגודל 128 בתים), ולכן עבור גדלי בלוק גדולים יותר נקבל ניצול טוב יותר של תכונת הlocality במקום של התכנית, אותה גדלי בלוקים קטנים יותר לא יוכלו לנצל במקרה זה. מכאן שהגיוני שקיבלנו גרף שיורד עם גודל הבלוק.



גודל בלוק L2 הוא 128 בתים. במקרה זה ניתן לראות שגודל בלוק 32 הוא האופטימלי ביותר לL1 מבחינת זמן הריצה של התכנית. ככל שגודל הבלוק עולה, כך מספר הבלוקים בL1 קטן (פחות ביטים בindex שהולכים לטובת הoffset). לכן ניתן להניח שעבור גדלי בלוק גדולים 64, 128 בתים, התכנית תגרום לconflict misses שיגרמו לflush – פעולה שלוקחת זמן רב ומקפיצה את אורך התוכנית ומכאן נקבל את העליה בחלק הימני של הגרף. את החלק השמאלי של הגרף ניתן להסביר בכך שגדלי בלוקים קטנים מדי (4,8) יגרמו למס' רב של L1 write / read compulsory misses עליהם נצטרך לשלם בזמני מחזור נוספים עד שהמידע יגיע מ L2 ומכאן העליה המתונה לעומת גודל בלוק של 32 בתים אשר מוצא איזון נכון בין מס' flushים למס' הmisses.



גודל בלוק של L1 הוא 8 בתים, כלומר קטן ולכן נצפה למס' רב של L1 compulsory misses אותם הL2 ינסה לבלום. ה-average memory access time גבוה כאשר גודל הבלוק של L2 נמוך. במקרה של גודל בלוק 8 יתכן כי הבלוק קטן מדי בכדי לנצל את תכונת הלוקאליות במקום של התכנית. עבור גודל בלוק של 256 נצטרך לשלם במחזורים נוספים על כל L2 Miss שכן רוחב הbus לזכרון הראשי הוא רק 64 בתים ולכן נראה עליה בקצה הימני של הגרף. גם גודל בלוק של 128 דורש מחזורים נוספים על כל L2 miss, אך כאן כיוון שהוא נותן את זמן הגישה הממוצע הטוב ביותר לזכרון ניתן להניח שהוא מצליח להגן על missים נוספים אשר אותם גדלי בלוקים נמוכים יותר לא מצליחים לבלום.

**הסימולציה הידנית עבור מקרה הבדיקה אותו אנו מגישים מצורפת מטה.**

|  |  |
| --- | --- |
| Direct mapped  Block transfer time [L2 to L1]: 7 cycles  (Bus size = 256)  Block transfer time [L1 to CPU]: 1 cycle  (Bus size = 32, always 4 bytes) | 2-Way-Set Associative  Block transfer time [MM to L2]: 163 cycles  (Bus size = 64) |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Command** | **Hit / Miss** | **L1 Index** | **L2 Index** | **L1 Tag** | **L2 Tag** | **Dirty?** | **Cycles** | **Type** |
| 1 LD 000000000000 00000 0000100 | L1 R-Miss L2 R-Miss | 00000 | 00000 | 000000000000 | 0000000000 | L1: No L2: No | 1+163+7+1 | Compulsory |
| 2 LD 000000000000 00000 0001000 | L1 R-hit | 00000 | - | 000000000000 | - | L1: No | 2+1 |  |
| 3 ST 000000000001 00010 0001000 A1A2A3A4 | L1 W-miss  L2 R-miss | 00010 | 01000 | 000000000001 | 0000000000 | L1: Yes  L2: No | 3+163+7+1 | Compulsory |
| 4 ST 000000000001 00010 0000100 B1B2B3B4 | L1 W-hit | 00010 | - | 000000000001 | - | L1: Yes | 4+1 |  |
| 5 LD 000000100100 00010 0010000 | L1 R-miss  L2 R-miss  L1 Flush  L2-W hit | 00010 | 00000 | 000000100100 | 0000001001 | L1: No L2: No, Yes | 5+163+7+7+1 | Conflict+  Compulsory |
| 6 ST 000000100100 00010 0000000 E1E2E3E4 | L1 W-hit | 00010 |  | 000000100100 |  | L1: Yes | 6+1 |  |
| 7 LD 000000000001 00010 0000000 | L1 R-miss  L2 R-hit  L1 Flush  L2 W-hit | 00010 | 01000 | 000000000001 | 0000000000 | L1: No  L2: No, Yes | 7+7+7+1 | Conflict |
| 8 ST 000000000000 00000 1111100 F1F2F3F4 | L1 W-hit | 00000 |  | 000000000000 |  | L1: Yes | 8+1 |  |
| 9 ST 100000000000 00000 1000000 C1C2C3C4 | L1 W-miss  L2 R-miss  L1 Flush  L2 W-miss  L2 Flush | 00000 | 00000 | 100000000000 | 1000000000 | L1: Yes  L2: No, Yes | 9+163+7+7+163+1+163 | Conflict |
| 10 ST 000000000101 00011 1010101 D1D2D3D4 | L1 W-miss  L2 R-miss | 00011 | 01000 | 000000000101 | 0000000001 | L1: Yes  L2: No | 10+163+7+1 | Compulsory + Conflict |

**Cache state:**

* Dirty marked in red

Steps 1, 2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **L1 Tag** | **L2 – Way 0 Tag** | **L2 – Way 1 Tag** | **L2 – LRU** |
| 00000 | 000000000000 | 0000000000 |  | 1 |
| 00001 |  |  |  | 0 |
| 00010 |  |  |  | 0 |
| 00011 |  |  |  | 0 |
| 00100 |  |  |  | 0 |
| 00101 |  |  |  | 0 |
| 00110 |  |  |  | 0 |
| 00111 |  |  |  | 0 |
| 01000 |  |  |  | 0 |
| … | | | | |

Steps 3, 4:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **L1 Tag** | **L2 – Way 0 Tag** | **L2 – Way 1 Tag** | **L2 – LRU** |
| 00000 | 000000000000 | 0000000000 |  | 1 |
| 00001 |  |  |  | 0 |
| 00010 | 000000000001 |  |  | 0 |
| 00011 |  |  |  | 0 |
| 00100 |  |  |  | 0 |
| 00101 |  |  |  | 0 |
| 00110 |  |  |  | 0 |
| 00111 |  |  |  | 0 |
| 01000 |  | 0000000000 |  | 1 |
| … | | | | |

Step 5:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **L1 Tag** | **L2 – Way 0 Tag** | **L2 – Way 1 Tag** | **L2 – LRU** |
| 00000 | 000000000000 | 0000000000 | 0000001001 | 0 |
| 00001 |  |  |  | 0 |
| 00010 | 000000100100 |  |  | 0 |
| 00011 |  |  |  | 0 |
| 00100 |  |  |  | 0 |
| 00101 |  |  |  | 0 |
| 00110 |  |  |  | 0 |
| 00111 |  |  |  | 0 |
| 01000 |  | 0000000000 |  | 1 |
| … | | | | |

Step 6:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **L1 Tag** | **L2 – Way 0 Tag** | **L2 – Way 1 Tag** | **L2 – LRU** |
| 00000 | 000000000000 | 0000000000 | 0000001001 | 0 |
| 00001 |  |  |  | 0 |
| 00010 | 000000100100 |  |  | 0 |
| 00011 |  |  |  | 0 |
| 00100 |  |  |  | 0 |
| 00101 |  |  |  | 0 |
| 00110 |  |  |  | 0 |
| 00111 |  |  |  | 0 |
| 01000 |  | 0000000000 |  | 1 |
| … | | | | |

Step 7:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **L1 Tag** | **L2 – Way 0 Tag** | **L2 – Way 1 Tag** | **L2 – LRU** |
| 00000 | 000000000000 | 0000000000 | 0000001001 | 0 |
| 00001 |  |  |  | 0 |
| 00010 | 000000000001 |  |  | 0 |
| 00011 |  |  |  | 0 |
| 00100 |  |  |  | 0 |
| 00101 |  |  |  | 0 |
| 00110 |  |  |  | 0 |
| 00111 |  |  |  | 0 |
| 01000 |  | 0000000000 |  | 1 |
| … | | | | |

Step 8:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **L1 Tag** | **L2 – Way 0 Tag** | **L2 – Way 1 Tag** | **L2 – LRU** |
| 00000 | 000000000000 | 0000000000 | 0000001001 | 0 |
| 00001 |  |  |  | 0 |
| 00010 | 000000000001 |  |  | 0 |
| 00011 |  |  |  | 0 |
| 00100 |  |  |  | 0 |
| 00101 |  |  |  | 0 |
| 00110 |  |  |  | 0 |
| 00111 |  |  |  | 0 |
| 01000 |  | 0000000000 |  | 1 |
| … | | | | |

Step 9:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **L1 Tag** | **L2 – Way 0 Tag** | **L2 – Way 1 Tag** | **L2 – LRU** |
| 00000 | 100000000000 | 1000000000 | 0000000000 | 0 |
| 00001 |  |  |  | 0 |
| 00010 | 000000000001 |  |  | 0 |
| 00011 |  |  |  | 0 |
| 00100 |  |  |  | 0 |
| 00101 |  |  |  | 0 |
| 00110 |  |  |  | 0 |
| 00111 |  |  |  | 0 |
| 01000 |  | 0000000000 |  | 1 |
| … | | | | |

Step 10 [Final state of the cache]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **L1 Tag** | **L2 – Way 0 Tag** | **L2 – Way 1 Tag** | **L2 – LRU** |
| 00000 | 100000000000 | 1000000000 | 0000000000 | 1 |
| 00001 |  |  |  | 0 |
| 00010 | 000000000001 |  |  | 0 |
| 00011 | 000000000101 |  |  | 0 |
| 00100 |  |  |  | 0 |
| 00101 |  |  |  | 0 |
| 00110 |  |  |  | 0 |
| 00111 |  |  |  | 0 |
| 01000 |  | 0000000000 | 0000000001 | 1 |
| … | | | | |

**Statistics:**

Total cycles: 1269;

Mem access cycles: 1214

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Read hits: | Read misses: | Write hits: | Write misses: | Miss Rate: |
| L1: | 1 | 3 | 3 | 3 |  |
| L2: | 1 | 5 | 2 | 1 |  |

Global miss rate = L1 miss-rate \* L2 miss rate =

AMAT: