

컴퓨터구조 project4

교수님 : 이성원 교수님

이름 : 이준휘

학번 : 2018202046

분반 : 월3, 수4

A. Basic

L1 cache는 Processor와 가장 가까운 cache이다. Harvard Arch.는 해당 cache를 Instruction과 Data를 따로 분리하는 방식을 사용하여 Pipeline processor에서의 동작이 더욱 원활하게 진행되도록 돕고 있다. Cache의 동작은 다음과 같다. Memory address를 전달받으면 cache 내에 해당 데이터와 일치하는 Set에 접근하여 tag들을 확인한다. 만약 동일한 tag를 발견한 경우 valid bit를 확인하여 해당 데이터가 쓰여진 데이터가 맞는 경우 이를 HIT 상태로 반환하게 된다. 그리고 해당 way에 저장된 data를 processor로 전달하게 된다. 해당 과정은 매우 적은 cycle(1~2)에서 동작이 가능하다. 하지만 해당 과정에서 tag가 틀리거나 valid bit가 꺼진 경우 MISS가 발생한다. L1의 Cache는 MISS가 발생하였을 때 아래 계층(memory or L2 Cache)로 접근하게 된다. 해당 계층은 상위 계층보다 메모리가 크지만 느리기 때문에 더욱 많은 cycle(L2 : 20 cycle, Memory : 200 cycle)을 소비하게 된다.

기본적으로 데이터는 크기가 커지면 하나의 word에 보관하지 못하기 때문에 이를 나누어 담게 된다. 보통 나누어 담은 데이터는 바로 옆에 인접해 있기 때문에 우리는 이러한 인접한 데이터의 집합을 Block이라 한다.

B. Observation for Program Behaviors

1. Bubble sort

해당 Bubble sort는 이전 project3에서 수행한 과제의 결과를 바탕으로 작성하였다.



```
C:\WINDOWS\system32\cmd.exe
WARNING: Memory_F.v:121: $readmemb(M_DATA_SEG.txt): Not enough words in the file for the requested range [0:1023].

-----
| H020-3-1647-01: Computer Architecture |
| CE.KW.AC.KR                         |
-----

FST info: dumpfile tb_PC.vcd opened for output.

Break signal: 1, # of Cycles: 2637

tb_PipelinedCPU_P.v:85: $finish called at 26485000 (1ps)

E:\학교\3학년 1학기\컴퓨터구조\prj3_PCPU_2022\2018202046_이준휘_Assignment#3>FC /L mem_dump_BS.txt mem_dump.txt
파일을 비교합니다: mem_dump_BS.txt - MEM_DUMP.TXT
FC: 다른 점이 없습니다.

E:\학교\3학년 1학기\컴퓨터구조\prj3_PCPU_2022\2018202046_이준휘_Assignment#3>FC /L reg_dump_BS.txt reg_dump.txt
파일을 비교합니다: reg_dump_BS.txt - REG_DUMP.TXT
FC: 다른 점이 없습니다.

E:\학교\3학년 1학기\컴퓨터구조\prj3_PCPU_2022\2018202046_이준휘_Assignment#3>gtkwave tb_PC.vcd

GTKWave Analyzer v3.3.108 (w)1999-2020 BSI

FSTLOAD | Processing 80 facts.
FSTLOAD | Built 79 signals and 1 alias.
FSTLOAD | Building facility hierarchy tree.
FSTLOAD | Sorting facility hierarchy tree.
```

위의 코드는 bubble sort의 assembly code와 실행 결과를 나타낸 것이다. 해당 bubblesort가 데이터에 접근하는 방식을 자세히 보자. Bubble sort는 바로 옆에 있는 인자와 값을 비교해 나가면서 정렬을 시도한다. 즉 바로 옆에 있는 인자 또한 cache에 있으면 hit의 확률이 증가하게 된다. 이를 위해서는 Block size를 키우거나, Set을 키움으로써 이 과정에서 AMAT를 줄일 수 있다.

2. Random access

```

#include <stdio.h>

void main() {

    int i;
    unsigned int x,y,a,b;

    x = 0x644c4053;
    y = 0x510fbab4;

    for (i=0;i<100;i++) {
        a = (x << 21);
        b = (x >> 11);
        x ^= a;
        y ^= ((y << 21) | b);
        if (y>0x80000000) {
            x ^= (y >> 3);
            y ^= 0;
        }
        else {
            x ^= ~(~y >> 3);
            y ^= 0xffffffff;
        }
        a = (x << 4);
        b = (x >> 28);
        x ^= a;
        y ^= ((y << 4) | b);
        printf("%x %x\n",x,y);
    }

    for (i=0;i<256;i++) {
        x ^= (x << 11);
        if (x>0x80000000) {
            x ^= (x >> 17);
        }
        else {
            x ^= ~(~x >> 17);
        }
        x ^= (x << 2);
        printf("%x\n",x);
    }

    for (i=0;i<256;i++) {
        printf("%x\n",i);
    }
}

```

```

C:\WINDOWS\system32\cmd.exe
C:\Users\#82109\Desktop\prj4_CACHE_-_2022>vvp tb_CC.o -fst
WARNING: Memory_C.v:351: $readmemb(M_TEXT_SEG.txt): Not enough words in the file for the requested range [0:1023].

-----
| H020-3-1647-01: Computer Architecture |
|                               CE.KW.AC.KR |
-----
FST info: dumpfile tb_CC.vcd opened for output.

Break signal: 1,   # of Cycles:      3154
-----
tb_PipelinedCPU_C.v:102: $finish called at 31655000 (1ps)
C:\Users\#82109\Desktop\prj4_CACHE_-_2022>gtkwave tb_CC.vcd

GTKWave Analyzer v3.3.108 (w)1999-2020 BSI

FSTLOAD | Processing 53 facts.
FSTLOAD | Built 48 signals and 5 aliases.
FSTLOAD | Building facility hierarchy tree.
FSTLOAD | Sorting facility hierarchy tree.

```

다음 코드와 실행 결과는 위 와 같이 나와있다. 해당 코드는 프로그램에 의해 무작위의 부분을 읽기 때문에 이로 인한 miss는 기본적으로 많은 코드다. 이를 Verilog를 통해 확인하면 더욱 쉽게 확인할 수 있다.

64	0.296	$1.04+0.296*20$ 0 = 60.04	0.316	0.173	57.2
128	0.247	$1.08+0.247*20$ 0 = 50.48	0.278	0.141	50.1
256	0.176	$1.12+0.176*20$ 0 = 36.32	0.229	0.130	42.15
512	0.113	$1.16+0.113*20$ 0 = 23.76	0.160	0.107	30.5

Compress95

# of Sets	Unified cache Miss rate	Unified cache AMAT	Split cache		Split cache AMAT
			Inst. Miss rate	Data Miss rate	
64	0.307	$1.04+0.307*20$ 0 = 62.44	0.322	0.159	56.6
128	0.153	$1.08+0.153*20$ 0 = 31.68	0.199	0.124	36.35
256	0.096	$1.12+0.096*20$ 0 = 20.32	0.059	0.105	15.32
512	0.038	$1.16+0.038*20$ 0 = 8.76	0.039	0.087	11.4

Vortex

# of Sets	Unified cache Miss rate	Unified cache AMAT	Split cache		Split cache AMAT
			Inst. Miss rate	Data Miss rate	
64	0.205	$1.04+0.205*20$ 0 = 42.04	0.211	0.125	38.9
128	0.179	$1.08+0.179*20$ 0 = 36.88	0.189	0.091	33.9
256	0.142	$1.12+0.142*20$ 0 = 29.52	0.174	0.045	29.4
512	0.102	$1.16+0.102*20$ 0 = 21.58	0.132	0.039	22.8

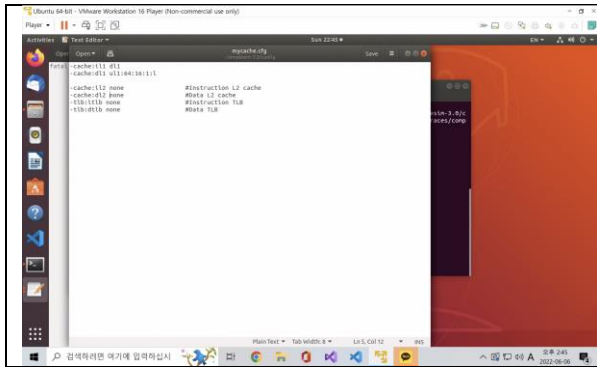
Sim1의 경우 Unified cache와 Split cache의 차이, 그리고 Set의 크기에 따른 차이를 볼 수 있다. Set은 이론적으로 커지면 커질수록 기존의 데이터를 지울 확률이 줄어들기 때문에 miss가 줄어들게 된다. 기본적으로 두 경우에서 모두 Set이 커질수록 AMAT가 작아지는 양상을 보인다. Unified cache와 Split cache의 차이는 기본적으로는 Set의 크기가 작을 때 Split cache의 AMAT가 더 작지만, Set의 크기가 커질수록 Unified cache랑 비슷하거나 역전당하는 상황이 보인다.

위의 표를 확인하였을 때 Mk88ksim에 가장 잘 어울리는 Set은 256 Set의 Unified Cache가 AMAT, cache size에 따른 cost를 따졌을 때 가장 좋은 효율을 낼 것으로 예상된다.

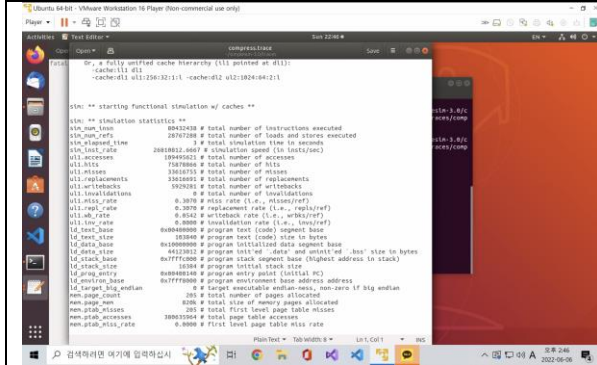
Compress95의 경우에는 같은 256 Sets이더라도 Split cache를 사용한 경우 더욱 가성비가 좋을 것으로 예상된다.

Vortex는 기본적으로 Set의 향상에 따른 성능 개선이 비교적 적기 때문에 128 Split cache를 사용하면 가성비가 비교적 양호해질 것이다.

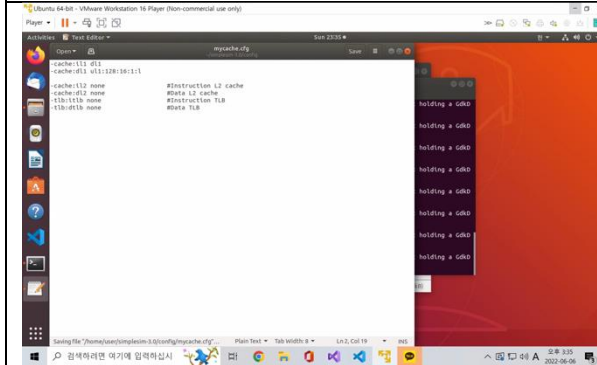
Unified cache 64 Sets	m88ksim
-----------------------	---------



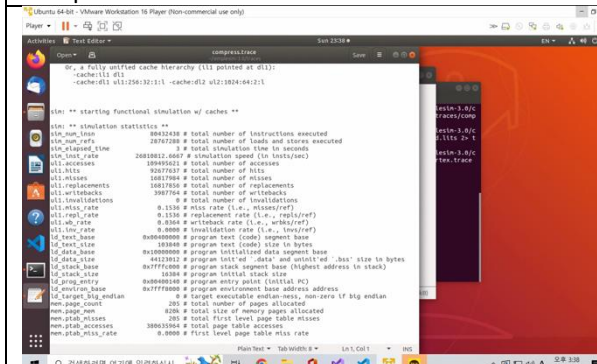
Compress95



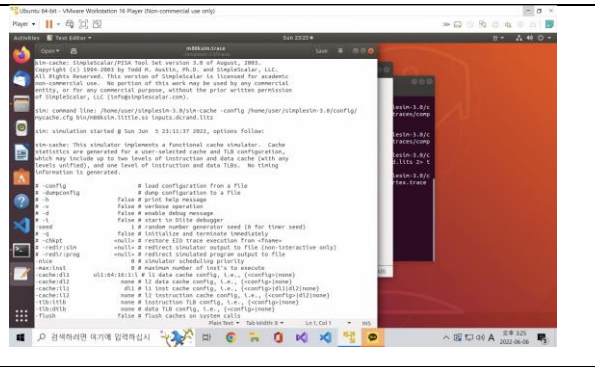
Unified cache 128 Sets



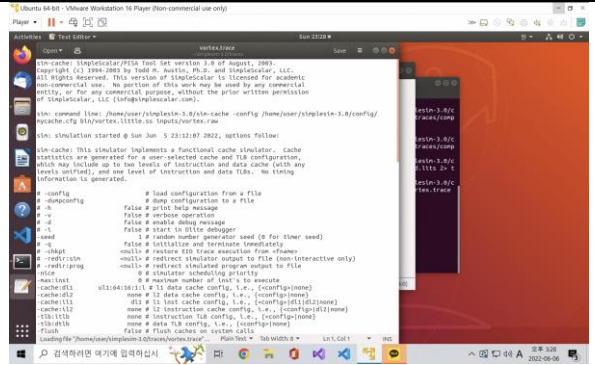
Compress95



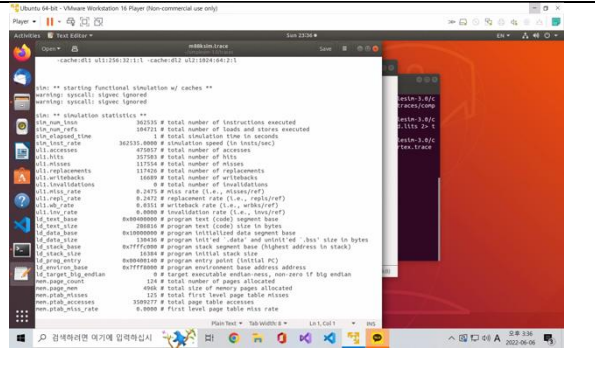
Unified cache 256 Sets



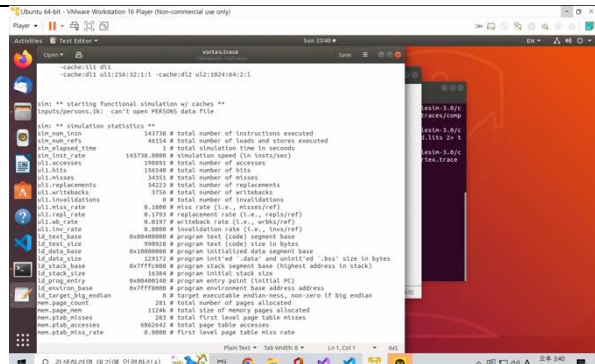
vertex



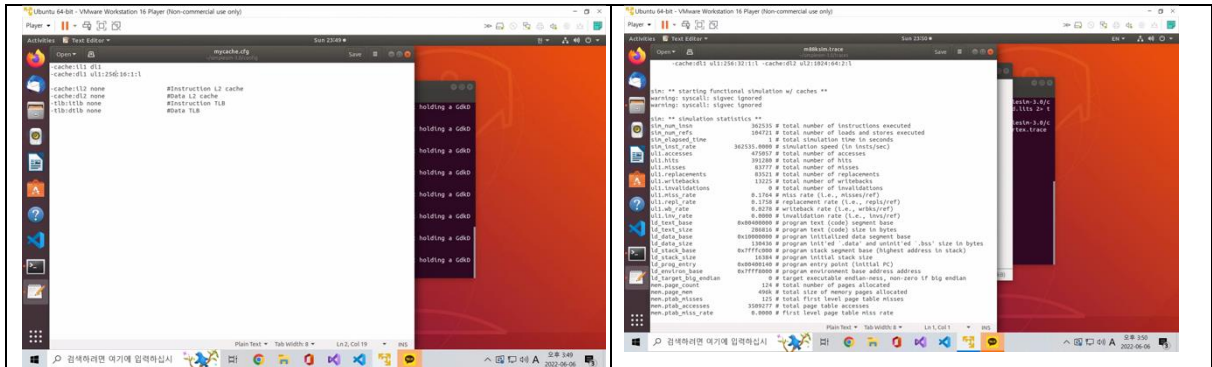
m88ksim



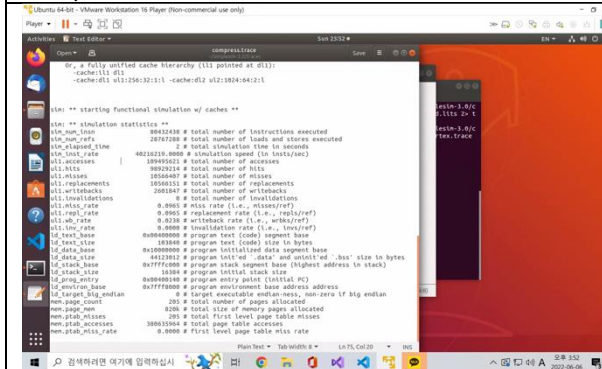
vertex



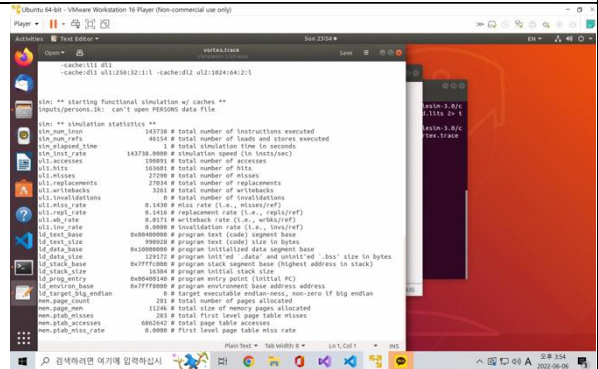
m88ksim



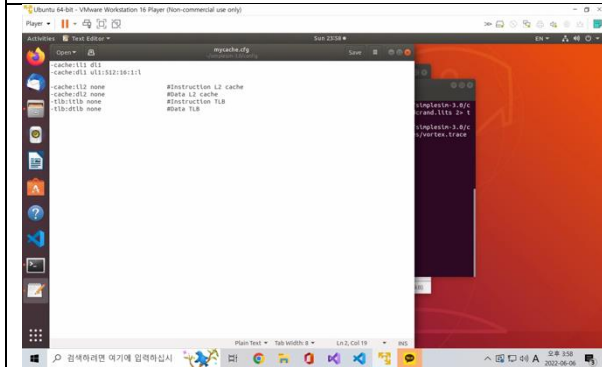
Compress95



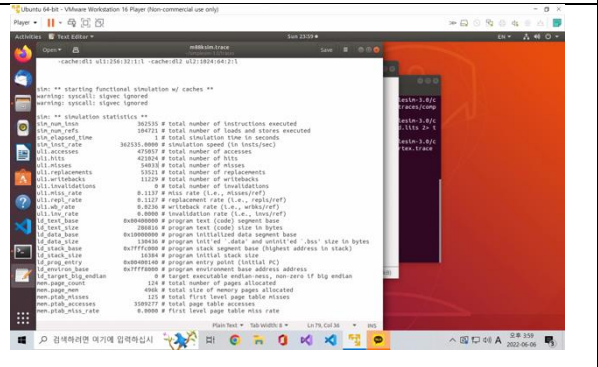
vertex



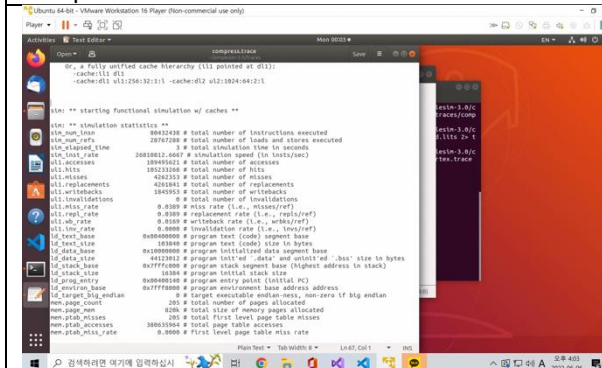
Unified cache 512 Sets

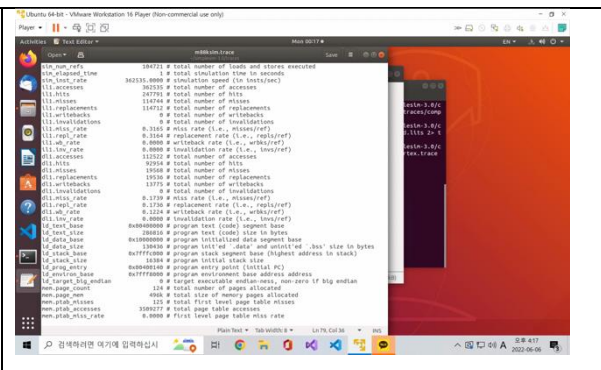


m88ksim

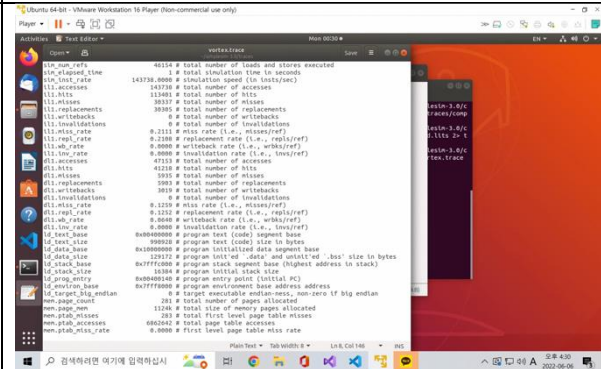


Compress95

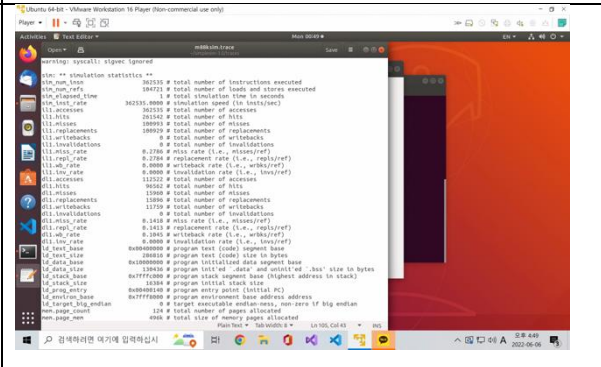




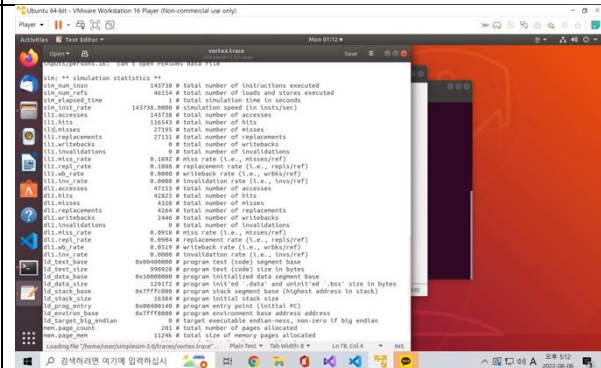
vertex



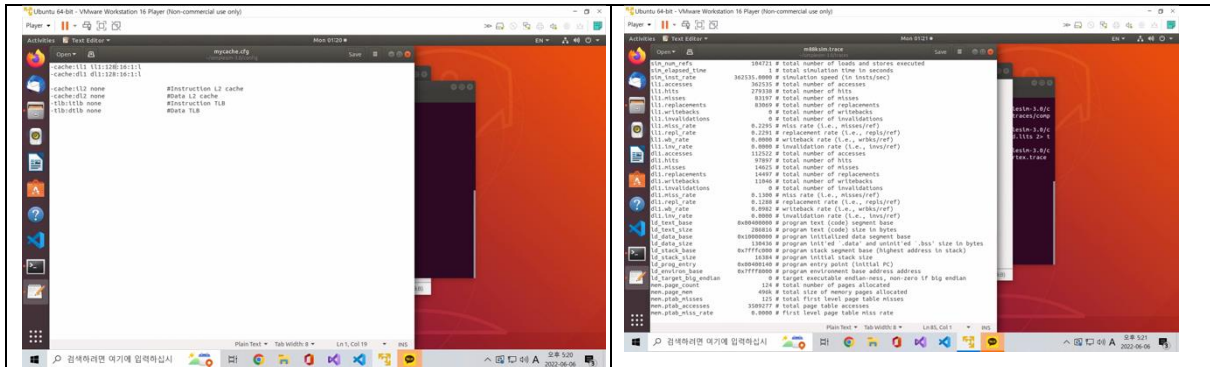
m88ksim



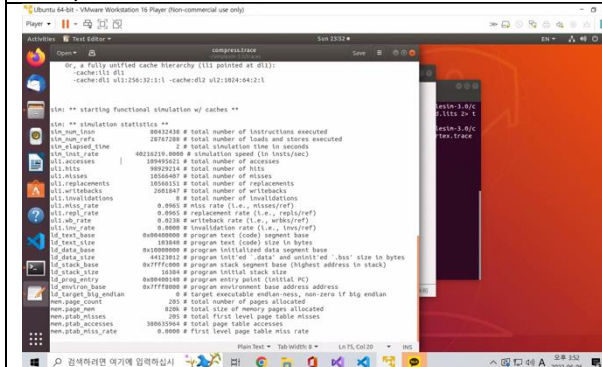
vertex



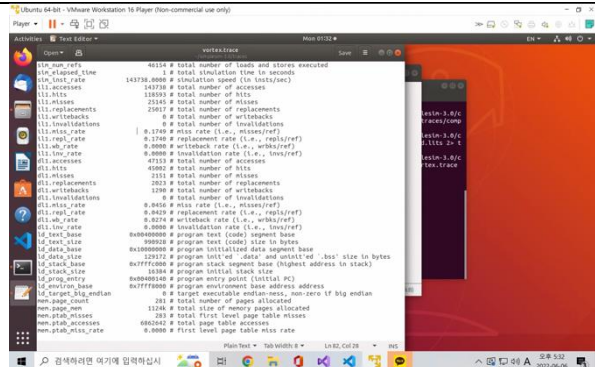
m88ksim



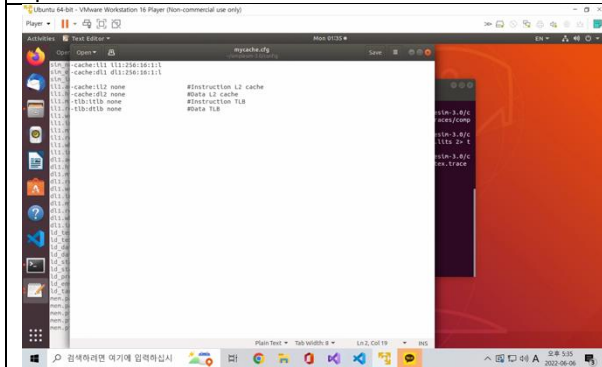
Compress95



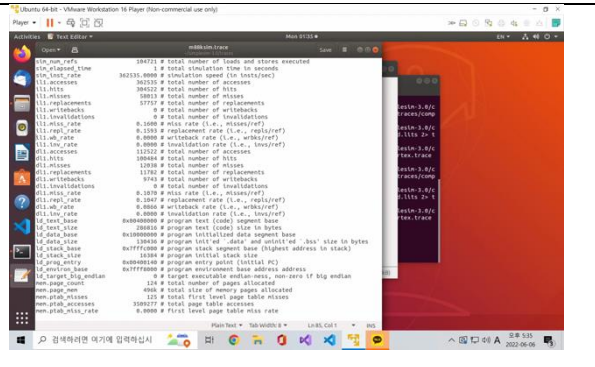
vertex



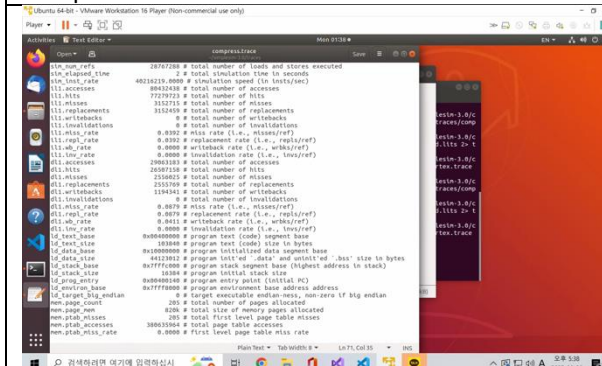
Split cache 512 Sets



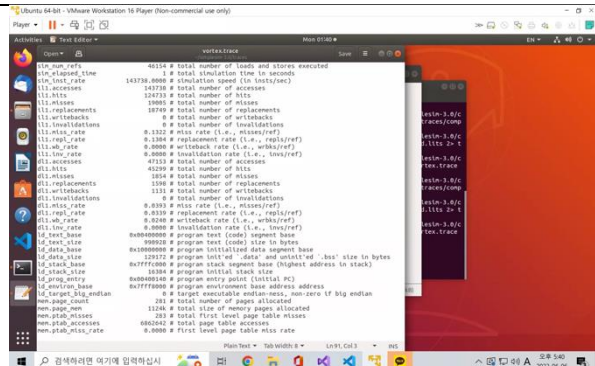
m88ksim



Compress95



vertex



3. Sim2

M88ksim

L1/L1D/L2U	Inst. Miss rate	Data Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	0.385	0.299	0.173	20.8

16/16/512	0.335	0.227	0.320	26.9
32/32/256	0.316	0.173	0.530	36.5
64/64/128	0.278	0.141	0.793	44.8
128/128/0	0.229	0.130	1	53.2

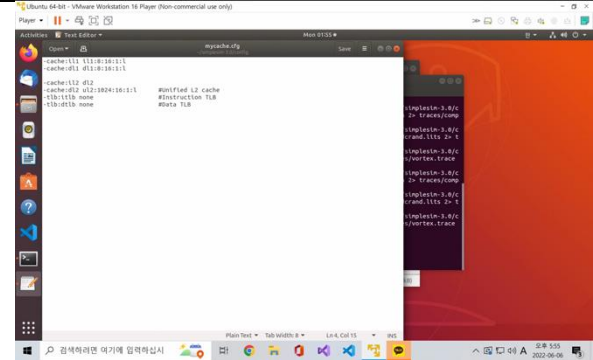
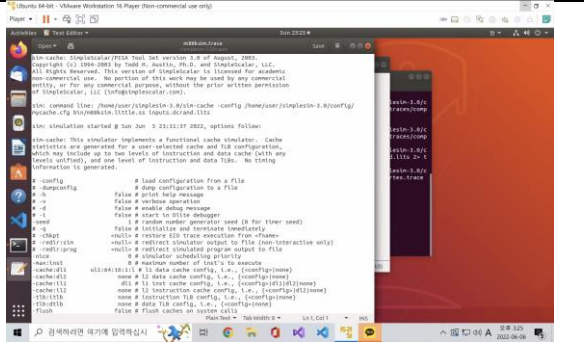
Compress95

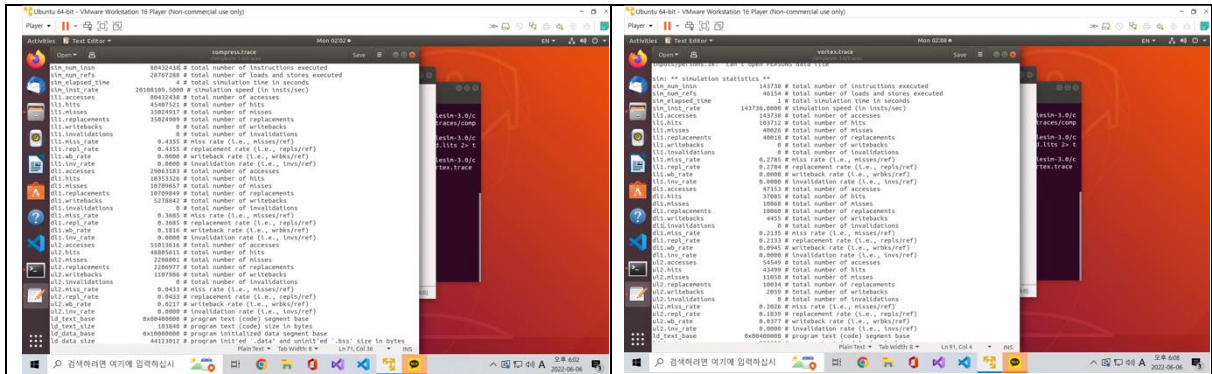
L1/L1D/L2U	Inst. Miss rate	Data Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	0.435	0.368	0.043	12.8
16/16/512	0.403	0.222	0.091	14.5
32/32/256	0.322	0.159	0.224	19.0
64/64/128	0.199	0.124	0.422	19.7
128/128/0	0.059	0.105	1	19.9

Vortex

L1/L1D/L2U	Inst. Miss rate	Data Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	0.278	0.213	0.202	16.7
16/16/512	0.225	0.159	0.428	22.9
32/32/256	0.211	0.125	0.640	29.0
64/64/128	0.189	0.091	0.873	33.0
128/128/0	0.174	0.045	1	36.6

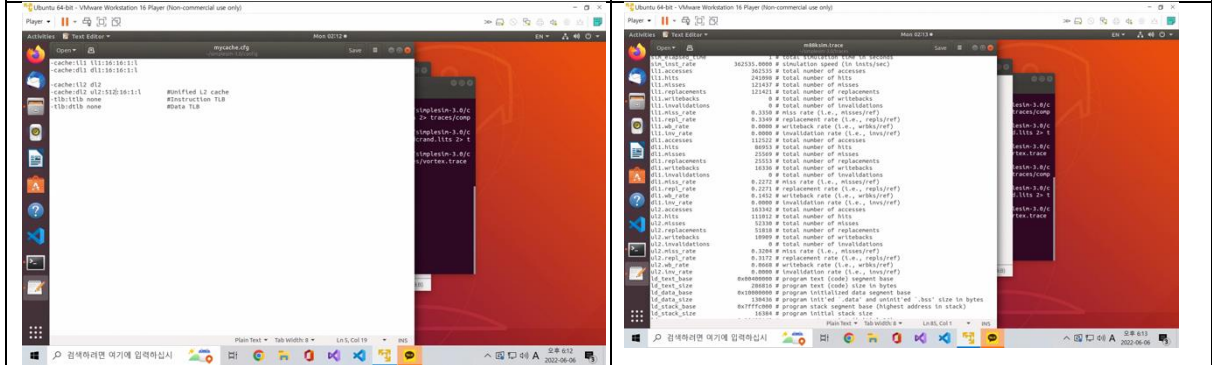
sim2의 경우 L2의 크기와 L1/L2간의 크기 비율에 따른 성능의 차이를 확인할 수 있다. L1의 크기가 작더라도 L2의 크기가 클 경우 L1의 Miss rate는 높아지지만 L2의 Miss rate는 낮아지기 때문에 200 cycle의 동작을 최대한 피할 수 있다. 단 이러한 방향으로 제작할 경우 L2U가 너무 커지게 되면 가격적인 문제가 발생할 수 있다. 때문에 M88ksim은 32/32/256을, Compress95는 16/16/512를, Vortex도 16/16/256을 사용하는 것이 가격과 AMAT를 고려하였을 때 가장 좋을 것으로 예상된다.

8/8/1024 cache 	m88ksim 
Compress95	vertex



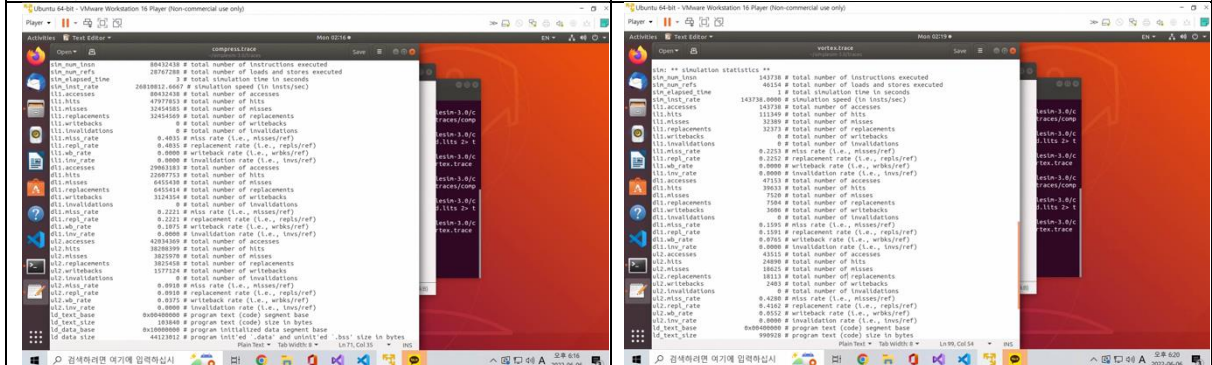
16/16/512 cache

m88ksim



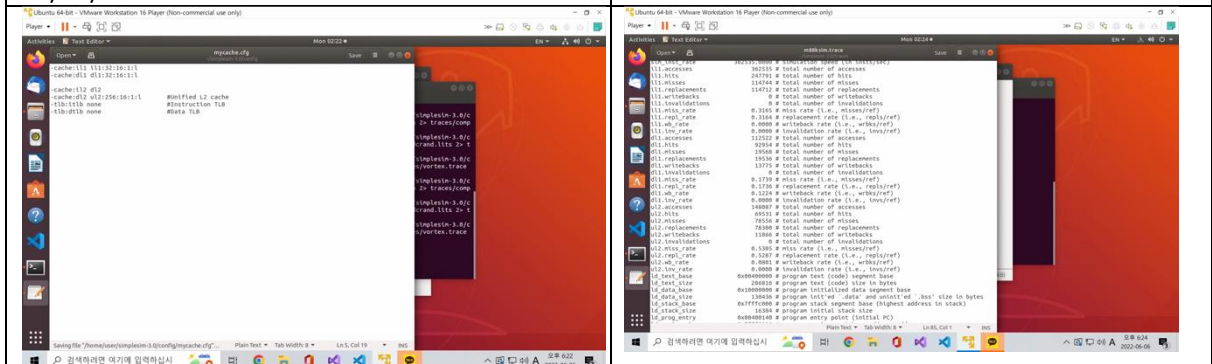
Compress95

vertex



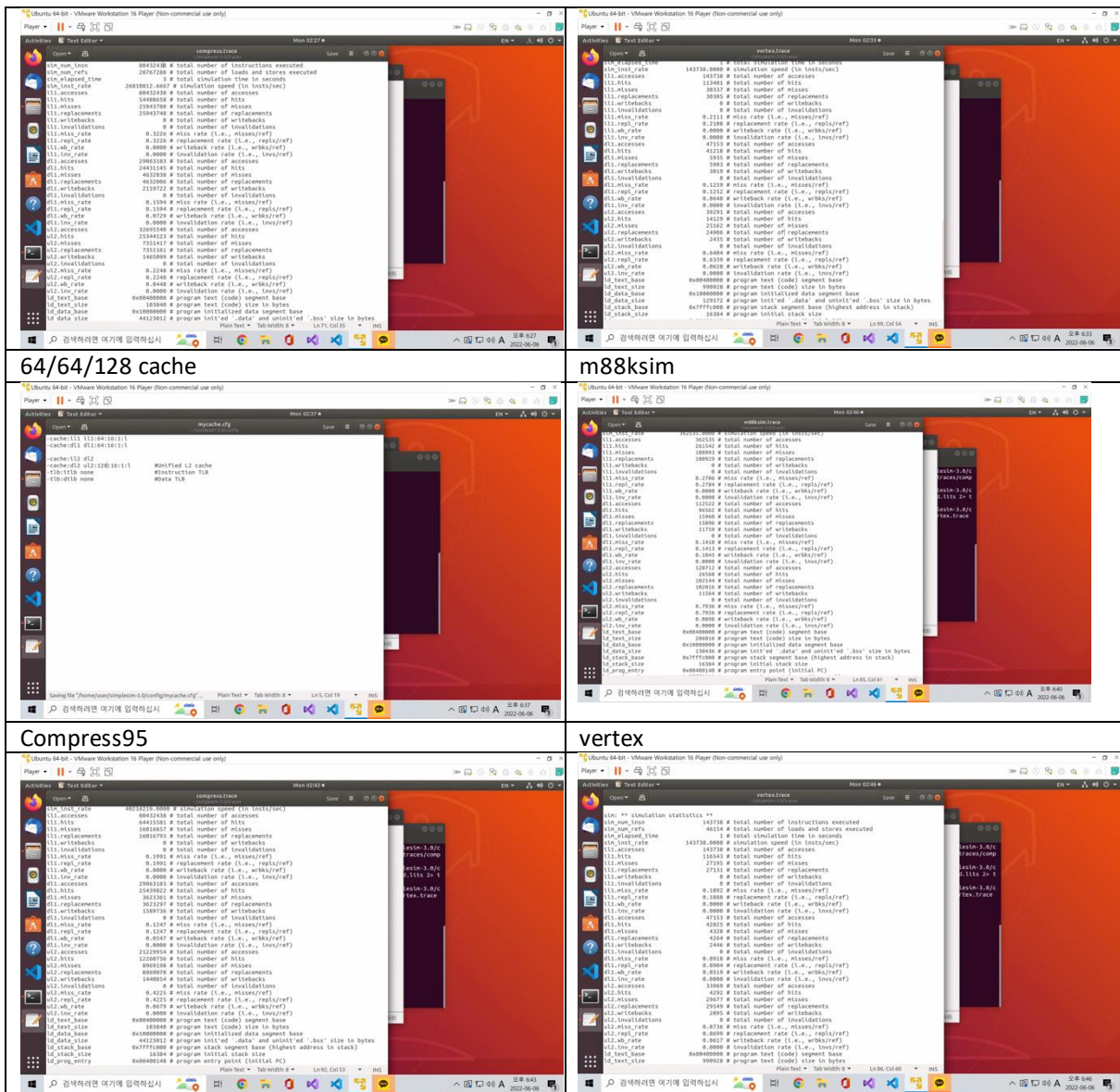
32/32/256 cache

m88ksim



Compress95

vertex



4. Sim3

M88ksim

#of Sets	Split Cache Miss rate/AMAT							
	1-way		2-way		4-way		8-way	
64	0.281	57.2	0.242	49.7	0.208	42.9	0.104	22.6
128	0.245	50.1	0.200	41.2	0.139	23.5	0.045	10.3
256	0.210	42.15	0.120	25.3	0.047	10.7	0.029	7.1
512	0.146	30.5	0.067	14.7	0.030	7.4	0.018	6.3

Compress95

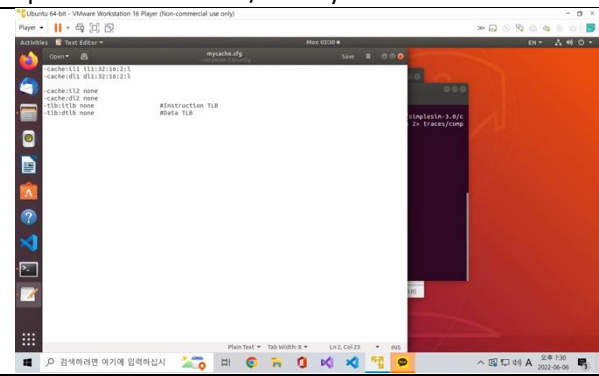
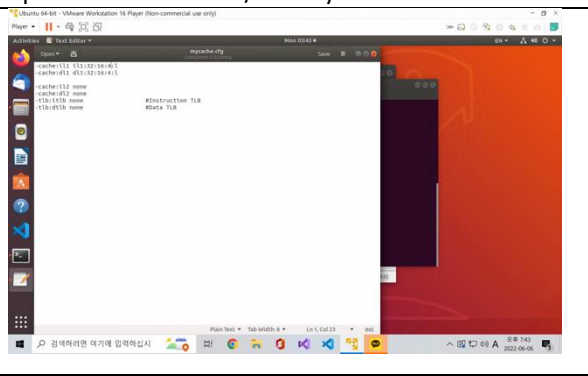
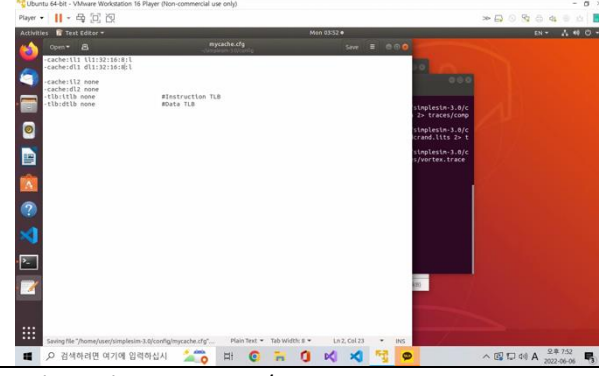
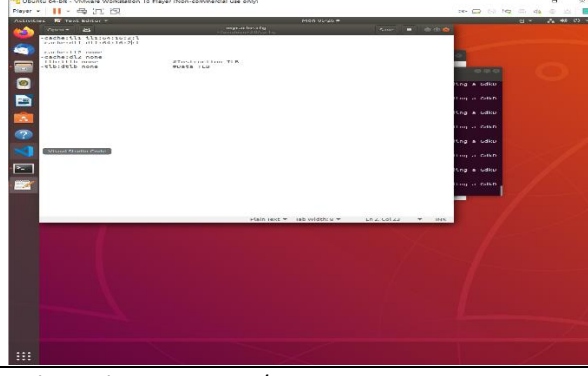


#of Sets	Split Cache Miss rate/AMAT							
	1-way		2-way		4-way		8-way	
64	0.278	56.6	0.097	19.6	0.023	5.7	0.020	5.3
128	0.176	36.35	0.068	14.7	0.020	5.3	0.017	4.8
256	0.071	15.32	0.021	5.43	0.018	4.8	0.014	4.1

512	0.051	11.4	0.018	4.8	0.014	4.1	0.011	3.6
-----	-------	------	-------	-----	-------	-----	-------	-----

Vortex

#of Sets	Split Cache Miss rate/AMAT							
	1-way		2-way		4-way		8-way	
64	0.189	38.9	0.158	32.8	0.136	28.4	0.119	25.1
128	0.164	33.9	0.139	29.0	0.111	23.5	0.043	9.9
256	0.141	29.4	0.108	22.9	0.049	11.0	0.018	4.9
512	0.108	22.8	0.060	13.2	0.018	5.0	0.016	4.5

Sim3의 경우 Associativity에 따른 AMAT의 차이를 확인할 수 있다. Associativity가 커질 경우 같은 Set의 정보가 들어왔을 때 이를 삭제하지 않고 보관할 수 있음으로써 hit의 경우가 늘어나게 된다. 특히 이는 여러 위치를 옮겨가는 프로그램에서 더욱 크게 힘을 발휘한다. 단 무작정 associativity를 늘릴 경우에도 cache의 크기가 커지고 cache가 복잡해진다. 때문에 이에 따른 가격과의 비교를 통해 합의점을 찾아야 한다. M88ksim은 256 Set에 4-way가 가장 성능 대비 가격이 좋을 것으로 예상되며 Compress95는 64 Set에 4-way가, Vortex는 128에 8-way가 가장 좋을 것으로 예상된다.

Split cache 64 Sets / 2-way 	Split cache 64 Sets / 4-way 
Split cache 64 Sets / 8-way 	Split cache 128 Sets / 2-way 
Split cache 128 Sets / 4-way 	Split cache 128 Sets / 8-way 



5. Sim4

M88ksim

Block size	Unified cache Miss rate	AMAT
16	0.113	23.92
32	0.046	10.38

64	0.025	6.2
128	0.009	3.0
256	0.005	2.26

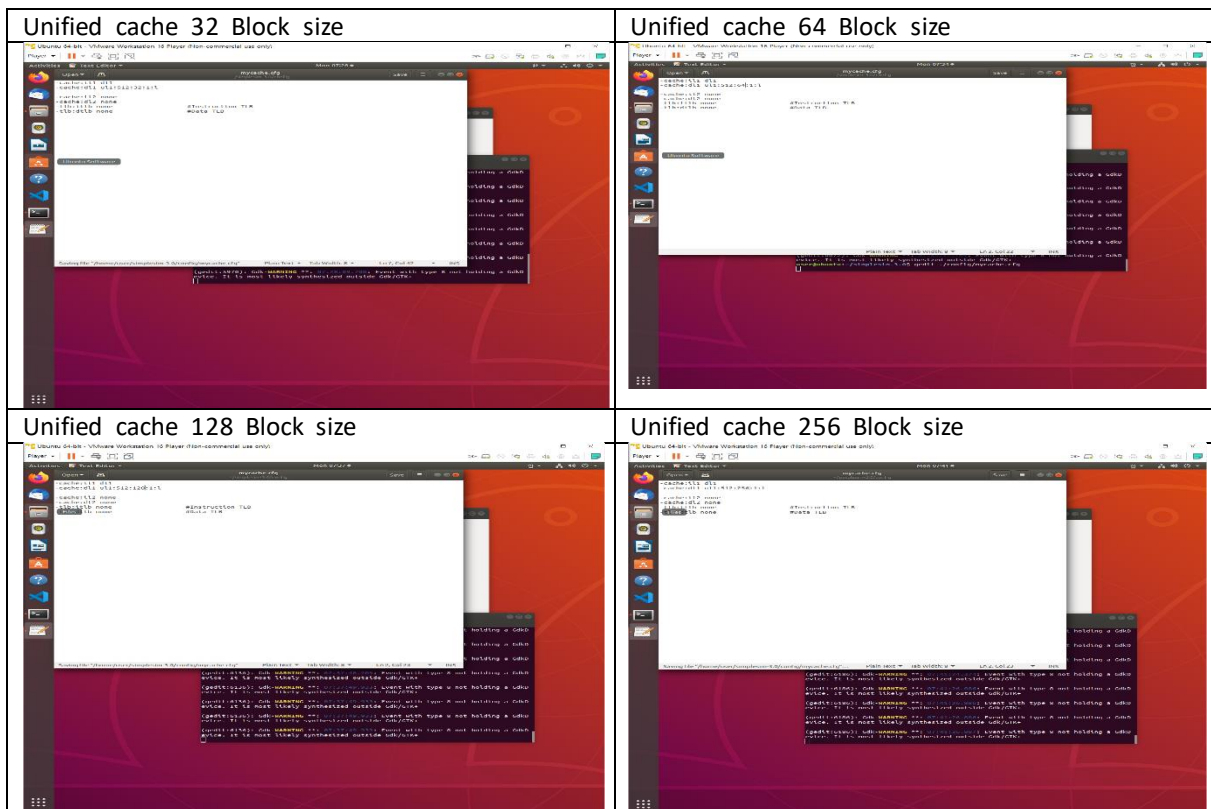
Compress95

Block size	Unified cache Miss rate	AMAT
16	0.038	8.92
32	0.018	4.7
64	0.012	3.6
128	0.011	3.4
256	0.004	2.06

Vortex

Block size	Unified cache Miss rate	AMAT
16	0.102	21.72
32	0.041	9.3
64	0.024	6.0
128	0.009	3.0
256	0.006	2.46

다음 Sim4는 Block size의 차이에 따른 성능의 차이를 확인할 수 있다. 기본적으로 프로그램을 내의 데이터의 크기가 클수록 인접한 memory의 위치할 확률이 커진다. 이는 Block size가 클 때 유리하게 작동할 수 있다. 또한 Instruction의 경우 대부분 바로 다음의 코드를 읽기 때문에 이러한 부분에서도 Block이 클수록 Instruction cache의 Miss rate가 줄어든다. Block size 또한 커질수록 cache가 커지기 때문에 적절한 타협이 필요하다. M88ksim의 경우 64 size가, Compression95는 32 size, Vortex도 32 size가 가격과 성능을 모두 잡을 것으로 예상된다.



위와 같은 Simulate를 종합하여 적절한 cache의 spec을 예상하였을 때 M88ksim은 256 set, 64 block, 4-way, Unified L1 cache와 1024 L2 cache 성능과 가격 측면에서 가장 좋을 것으로 예상된다. Compress95는 256 set, 32 block, 2-way, Split L1 cache와 1024 L2 cache가 가장 좋을 것으로 예상된다. Vortex는 128 set, 32 block, 8-way, Split L1 cache와 512 L2 cache가 좋을 것이다.

D. Consideration

위의 결과를 토대로 Set, associativity, Block size가 클수록 성능이 좋으며, L2 cache가 존재하고 L2의 크기가 클수록 성능이 좋아진다는 것을 확인할 수 있었다. 단 이러한 성능의 상승폭은 점점 작아지며 이에 따른 가격 또한 올라가기 때문에 가격과 성능의 중간점에서 타협을 봐야 한다는 점을 느낄 수 있었다. M88ksim 프로그램의 경우 Block의 크기가 큰 것을 보았을 때 연속적인 값을 읽는 경우가 많을 것으로 예상되며, Compress95는 반복문이 적고 불러들여야 할 Instruction의 수가 많을 것으로 예상된다. Vortex는 associativity가 큰 것을 미루어 보았을 때 memory의 주소의 이동이 잦을 것으로 예상된다. 때문에 기존에 주어진 Bubble sort와 Random access 프로그램은 각각 M88ksim, Vortex의 cache와 비슷한 양상으로 사용할 경우 성능이 좋을 것으로 예상된다.

E. Reference

M88ksim: <https://www.spec.org/cpu95/CINT95/124.m88ksim/>

Compress95: <https://courses.cs.washington.edu/courses/cse471/06sp/hw/benchmark-guide.pdf>

Vortex: <https://www.spec.org/cpu95/CINT95/147.vortex/>