컴퓨터구조 Project #4

(Cache Design)

수업명: 컴퓨터구조

교수님: 이성원 교수

수업 시간: 월 3교시, 수 4교시

소속: 컴퓨터정보공학부

학번: 2018202074

이름: 김상우

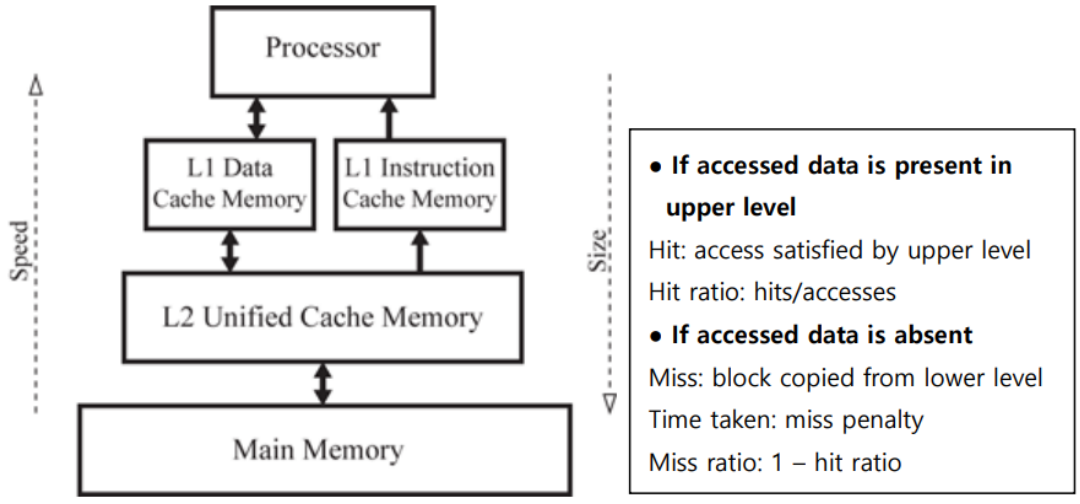
Introduction

Processor와 Main Memory는 Processor가 Main Memory로부터 값을 받아 동작을 하며 이 과정에서 Main Memory의 경우 방대한 instruction과 data에 대한 정보를 지니고 있다. 이 때문에 Processor가 이를 접근할 때마다 많은 시간이 소모되게 된다. 이를 해결하기 위해 Cache라는 개념이 존재한다.

Cache란 Processor와 Main Memory사이에 존재하며 instruction과 data에 대해 저장하는 기능을 수행한다. 위에서 언급한 것처럼 Main Memory의 경우 너무 방대한 크기를 가지기 때문에 한번 접근해 꺼내올 때마다 너무 긴 시간을 소모하게 되나 Cache의 경우 Main Memory보다 적은 양의 데이터를 저장하게 되어 비교적 빠르게 Processor로 값을 가져올 수 있게 되는 것이다.

Cache와 Main Memory 중 어디서 값을 받아올지는 Hit와 Miss를 판단하여 결정한다. 여기서 Hit란 Cache내 값이 있는 경우, Miss란 Cache 내에 data가 없는 경우를 의미한다. Hit가 발생할 경우, Processor는 Main Memory에 접근하지 않고 Cache로부터 값을 받아오기 때문에 Main Memory에 직접 접근하는 것에 비해 더 빠르게 결과를 처리할 수 있다. 그러나 Miss가 발생할 경우 Cache에 접근하여 Hit/Miss판단하는 데 소모된 시간과 Main Memory에 추가적으로 접근하는 시간까지 발생해 Cache가 없을 때 보다 더 많은 시간 손실이 발생하게 된다. 허나 평균적으로 Cache를 설치하여 시간 소모를 줄이는 경우가 더 많다.

L2 Cache의 경우 이미 추가된 하나의 캐쉬와 Main Memory사이에 Cache를 추가하는 것으로 이 경우 또한 위와 같이 Hit와 Miss를 통해 Processor에서 값을 가져갈 수 있다. 단, 해당 캐쉬의 경우 L1 Cache에 값이 없을 경우 접근되며 L2 Cache에 값이 있을 경우(Hit)다시 L1 cache를 통해 Processor에 전달되며 값이 없을 경우(Miss) Main Memory에 접근하여 값을 빼내 Cache가 없을 때 보다 더 오랜 시간이 걸린다는 점에서 L1 Cache와 같은 단점을 지니고 있다는 것을 알 수 있다.



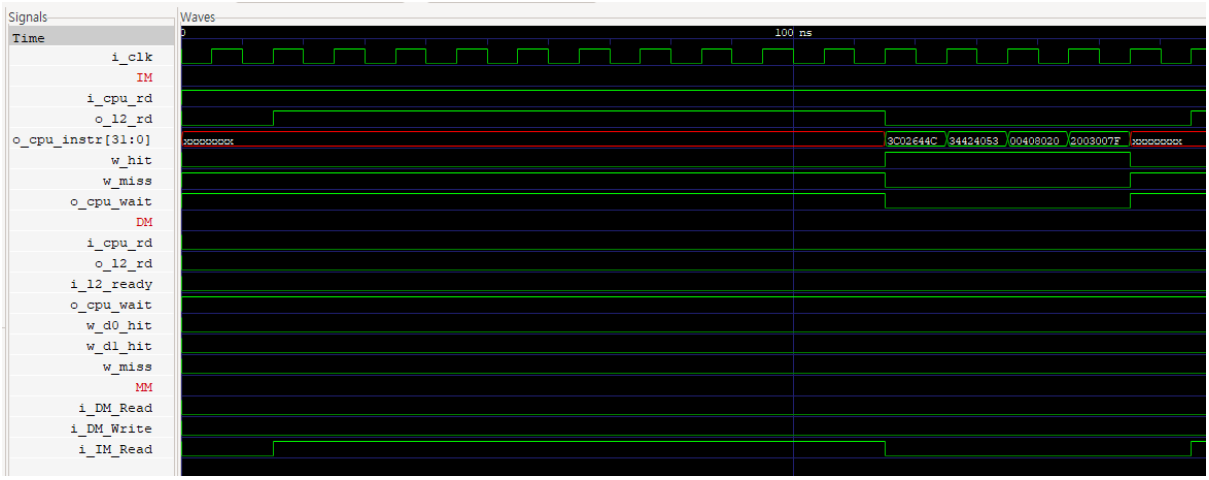
(제안서에 기재된 Cache의 모습)

-L1 hit와 miss에 대한 동작

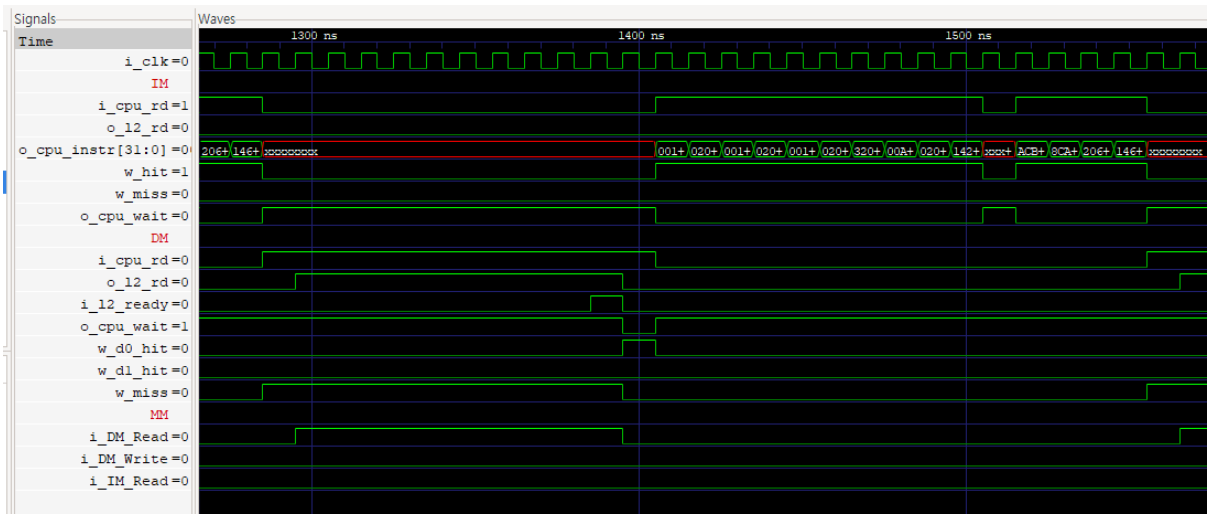
실험에 앞서 IM, DM(Instruction과 data)에 대한 동작과 MM(MainMemory)에 대한 접근을 파악하기 위해 아래와 같이 IM과 DM에 대하여 Hit와 Miss를 판별할 수 있는 변수를, MM에 대하여 Instruction이 읽혔는지 확인을 위한 i_IM_Read와 Data가 읽히고 써지는 지에 대한 i_DM_Read와 i_DM_Write를 추가해 판별할 수 있는 변수들을 세팅하여 결과를 확인하였다.

두 캐쉬 모두 Instruction에 대해 Miss/hit가 날 경우 아래와 같이 동작한다.

IM에 대해 캐쉬에 상관없이 두 경우 모두 i_cpu_rd가 활성화되고, Miss의 경우 o_12_rd또한 활성화 된다. 이를 바탕으로 Main Memory에 접근, Miss가 발생한 instruction에 대한 정보를 읽는다. 이후 값을 받아왔으니 hit가 활성화되며 1cycle마다 필요한 작업을 위해 o_cpu_instr에서 확인 가능하듯 동작을 수행하는 것을 확인할 수 있다. 반면 Miss없이 Hit가 발생한 경우 MainMemory에 접근없이 o_cpu_instr이 들어온 것을 확인할 수 있다.

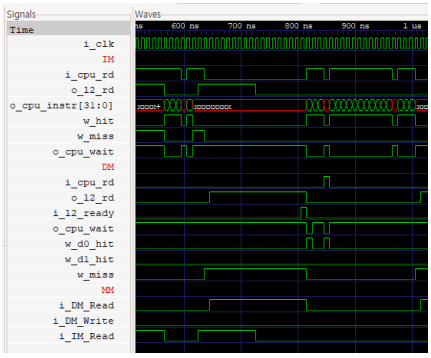


(Miss 발생의 경우)

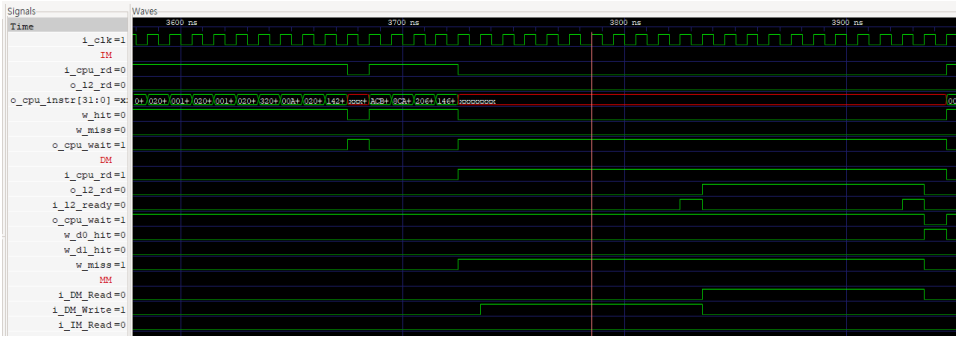


(Hit 발생의 경우)

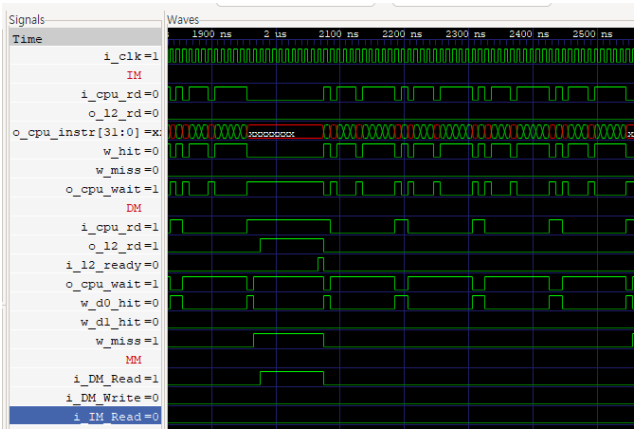
Data에 대해서는 아래와 같다. 기본적으로 2-ways set associative cache를 지원하여 w_d0_hit와 w_d1_hit라는 두가지 변수를 사용하여 hit여부를 확인하고 있다. 두 변수 모두 hit를 잡아내지 못한, 즉 Miss가 발생한 경우 o_12_rd가 활성화되어 i_DM_Read가 함께 활성화되는 모습을 보이고 있다. 이후 다시 읽어오며 hit가 활성화되는 모습을 보이고 있다.



경우에 따라서 MainMemory에 기술할 필요가 있을 경우 i_DM_Write를 활성화시켜 MainMemory에 값을 입력하고 이후 Read를 통해 읽어오는 모습또한 확인할 수 있었다.

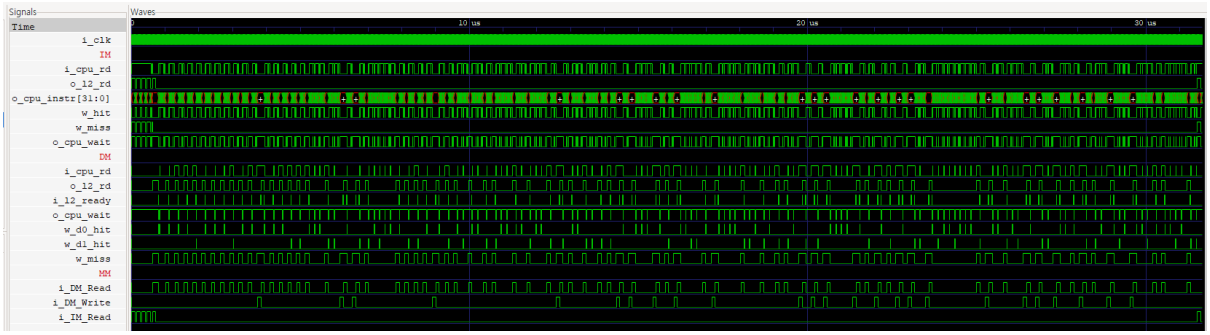


이후 Data를 w_miss는 활성화되지 않고 w_d0_hit 혹은 w_d1_hit가 활성화되어 MainMemory에서 read할 필요없는 경우(Hit)에는 아래와 같이 MainMemory에 대한 접근이 없는 모습을 확인할 수 있다.

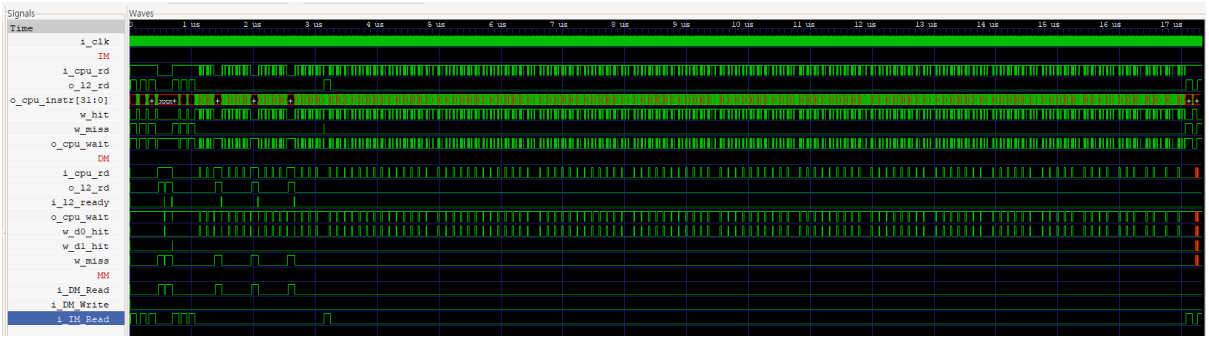


공통적으로 wait, 즉 o_cpu_wait들을 통해 받아오는 동안 cpu가 전송을 기다려주는 모습 또한 확인이 가능했다. 이는 instruction과 data에 대한 값이 없으면 실행이 불가능함을 위한 처리이다.

Random access



bubblesort



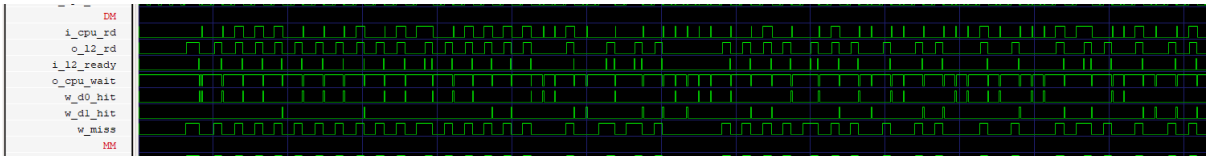
위는 각각 Random Access와 bubble sort를 진행했을 때의 모습이다.

IM에 대하여는 Random Access의 경우 처음에 명령어들이 일정기간동안 들어온 이후, 계속해 MISS를 유지하는 모습을 볼 수 있다. 이는 Random Access에 사용되는 Instruction이 저장가능한 Instruction의 최대 수 보다 적어서 일어나는 현상이다. Bubble sort 또한 이러한 경향을 보이는 것을 알 수 있다. 위 실험 결과에서는 Random Access가 먼저 Miss만 나오는 영역으로 들어감을 확인할 수 있었다.

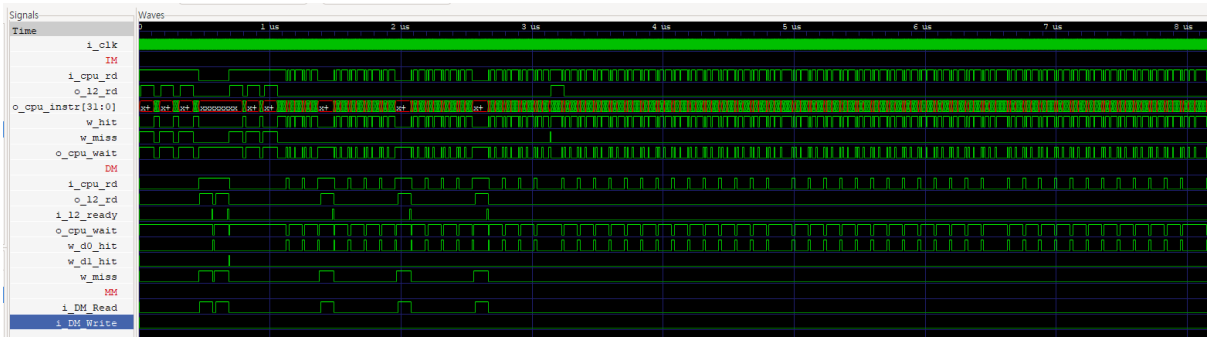
(상단이 Bubble sort, 하단이 Random Access이다.)

DM의 경우 Random Access의 경우 난수 생성인 만큼 Bubble sort에 비해 hit와 miss의 발생이 난잡한 것을 확인할 수 있다. 또한 난수 발생을 통해 값을 넣어 주기 때문에 충분한 Hit 이후에도 계속해서 Miss가 계속 발생하는 모습을 확인할 수 있다.

이 때문에 Miss가 발생하는 경우 Main Memory에 접근하게 되는 Cache에 특성에 기반하여 Random Access는 Miss가 계속해 발생하고, 이 때문에 Data Miss가 발생할 때마다 Main Memory에 접근하는 모습을 볼 수 있다.



반면 Bubble sort의 경우 아래와 같이 처음에 Cache 내의 저장을 위해 MISS가 몇 번 발생한 이후 같은 주기의 hit가 일정하게 나오며 Miss는 더 이상 발생하지 않는 모습을 볼 수 있다. 해당 주기는 계속해서 발생하는 데, 이는 Bubble sort의 같은 명령어와 Data를 반복해서 사용한다는 점과 다루는 Data가 점점 줄어든다는 점 때문에 발생한 것이다.



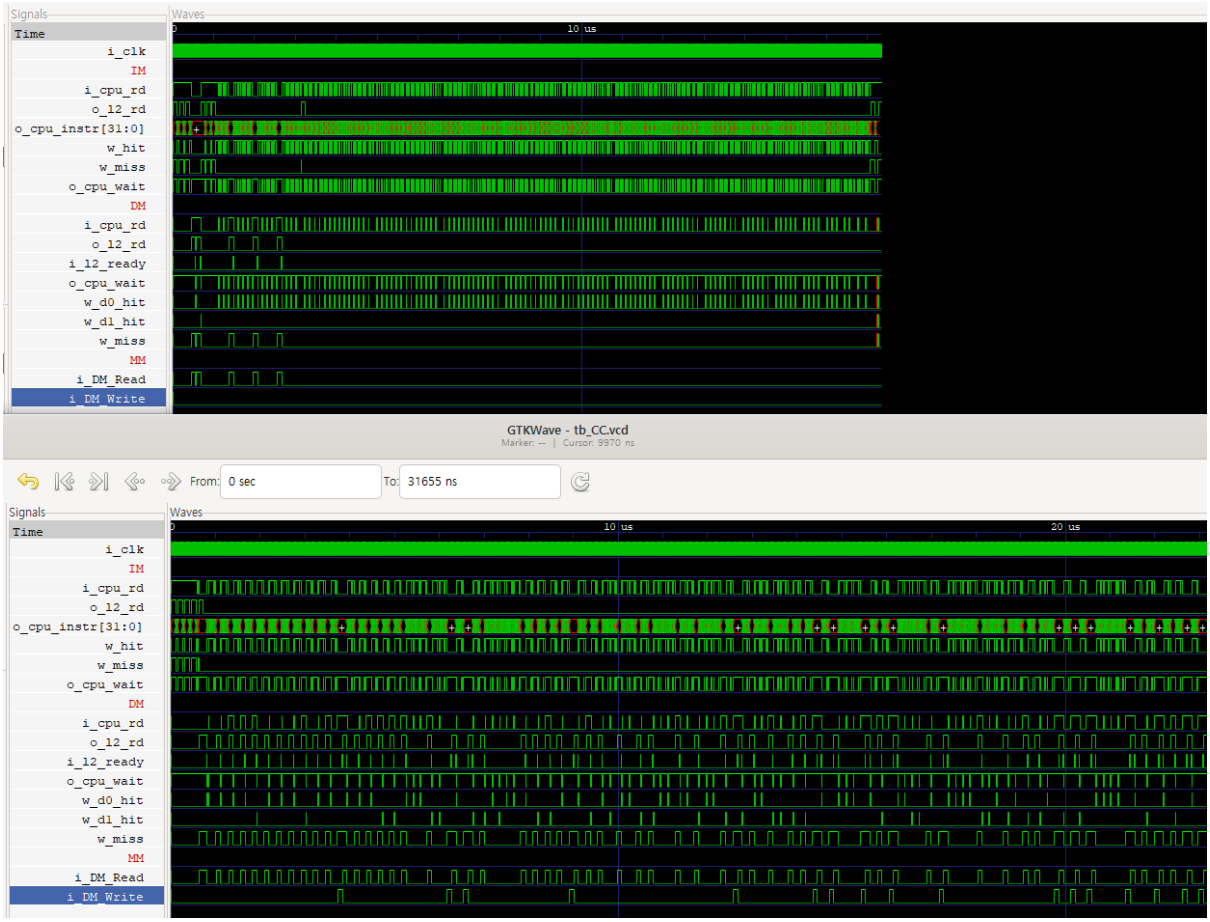
이 때문에 Miss가 발생하는 경우 Main Memory에 접근하게 되는 Cache에 특성에 의해 Bubble Sort의 경우 초반 Instruction과 Data가 충분히 저장되어 더 이상 Miss가 발생하지 않게 된 후에는 Main Memory에 접근하지 않는 모습이다.

결과적으로 Bubble Sort가 전반적으로 Random Access보다 좋은 결과를 내고 있는 모습이다.

그렇다면 여기서 data size를 키우면 어떻게 될까?

instruction과 data의 종류와 양을 늘려 이를 처리할 경우 Cache의 크기는 일정하기 때문에 cache가 담을 수 있는 양보다 instuction이나 data양이 더 적다면 위의 bubble sort와 마찬가지로 모든 값들을 Cache가 저장한 후에는 Miss가 발생하지 않을 것이며 Random access 또한 마찬가지로 모든 Data를 담고 나면 더 이상 Miss가 발생하지 않을 것이다.

다만 Cache의 용량보다 저장이 필요한 데이터나 instruction이 더 많을 경우 Miss양과 Hit양이 늘어나게 될 것이다. 특히 Instuction이 늘어날 경우 반복문에 경우 계속해 Miss가 발생하게 되어 치명적일 것이다. Data의 경우 Random Access의 경우 지금과 마찬가지로 Miss가 계속 발생할 것이나 종류가 늘어난 만큼 Hit가 발생하기 더 힘들어 질 것이다. Bubble Sort의 경우 일정만큼 진행되면 다뤄지는 Data수가 Cache용량보다 적어져 Miss가 없어지겠지만 이 구간까지 도달하는 것이 더 오래 걸리게 될 것이다.(Data의 format size를 늘리는 경우 Hit Miss판별에서 비교하는 등에서 처리하는 bit가 증가할 것이므로 근소하게 증가할 것이다.)



같은 시간대로 비교해 볼 경우, bubble sort가 더 일찍 종료되며 hit와 miss에 대해 규칙적인 결과를 낸다는 것을 확인할 수 있다.

Bench Mark비교를 수행하기에 앞서 다음과 같이 초기 설정을 진행하였다.

```
kw2018202074@ubuntu: ~/simplesim-3.0
kw2018202074@ubuntu:~$ cd simplesim-3.0/
kw2018202074@ubuntu:~/simplesim-3.0$ make config-pisa
rm -f config.h machine.h machine.c machine.def loader.c symbol.c syscall.c
ln -s target-pisa/config.h config.h
ln -s target-pisa/pisa.h machine.h
ln -s target-pisa/pisa.c machine.c
ln -s target-pisa/pisa.def machine.def
ln -s target-pisa/loader.c loader.c
ln -s target-pisa/symbol.c symbol.c
ln -s target-pisa/syscall.c syscall.c
rm -f tests
ln -s tests-pisa tests
kw2018202074@ubuntu:~/simplesim-3.0$ make
my work is done here...
kw2018202074@ubuntu:~/simplesim-3.0$ make sim-tests
cd tests ; \
make "MAKE=make" "RM=rm -f" "ENDIAN=`./sysprobe -s`" tests \
        "DIFF=diff" "SIM_DIR=.." "SIM_BIN=sim-fast" \
        "X=/" "CS=;" ; \
cd ..
make[1]: Entering directory '/home/kw2018202074/simplesim-3.0/tests-pisa'
```

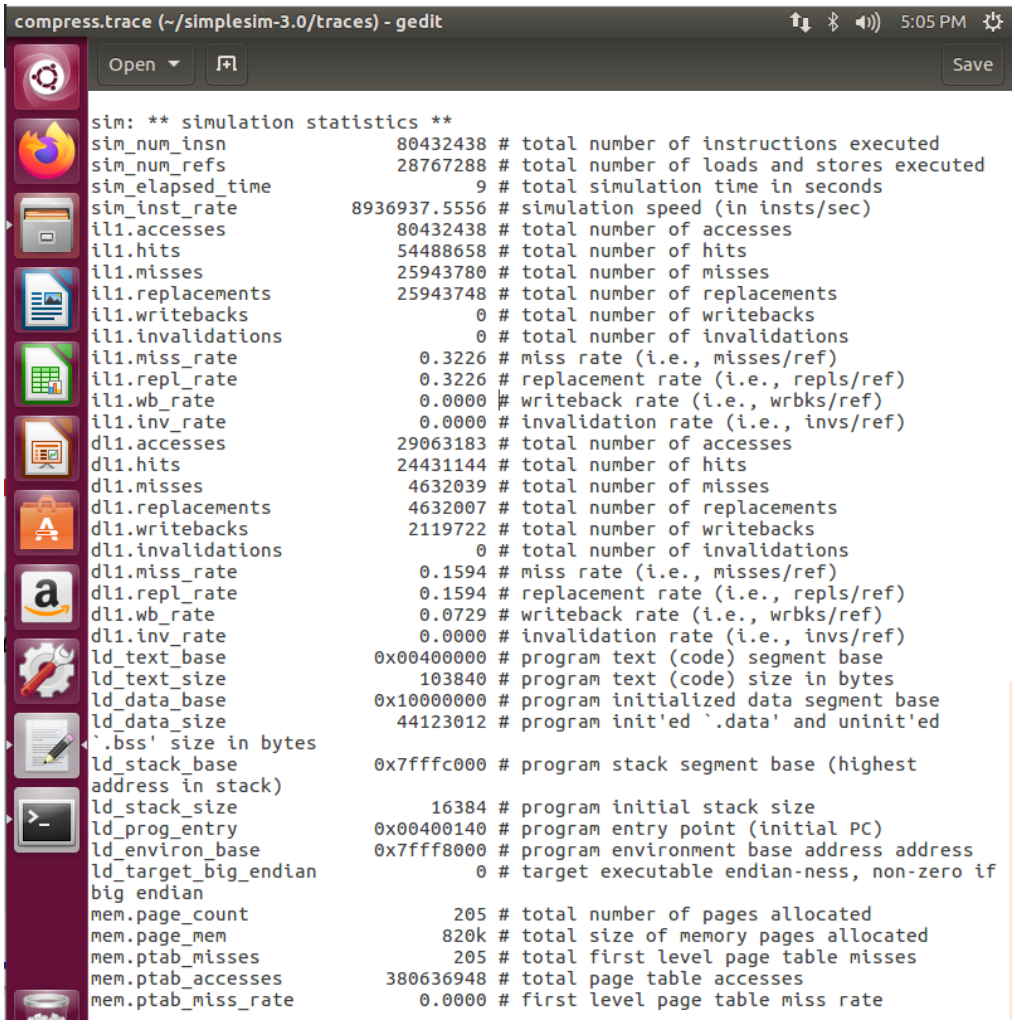
이후 아래 명령어를 통해 mycache.cfg에 접근, 해당 값을 변경하며 cache의 설정을 변경하였다.

```
kw2018202074@ubuntu:~/simplesim-3.0$ gedit ./config/mycache.cfg
```

```
Open
-cache:il1 il1:32:16:1:l
-cache:dl1 dl1:32:16:1:l
-cache:il2 none
-cache:dl2 none
-tlb:itlb none
-tlb:dtlb none
```

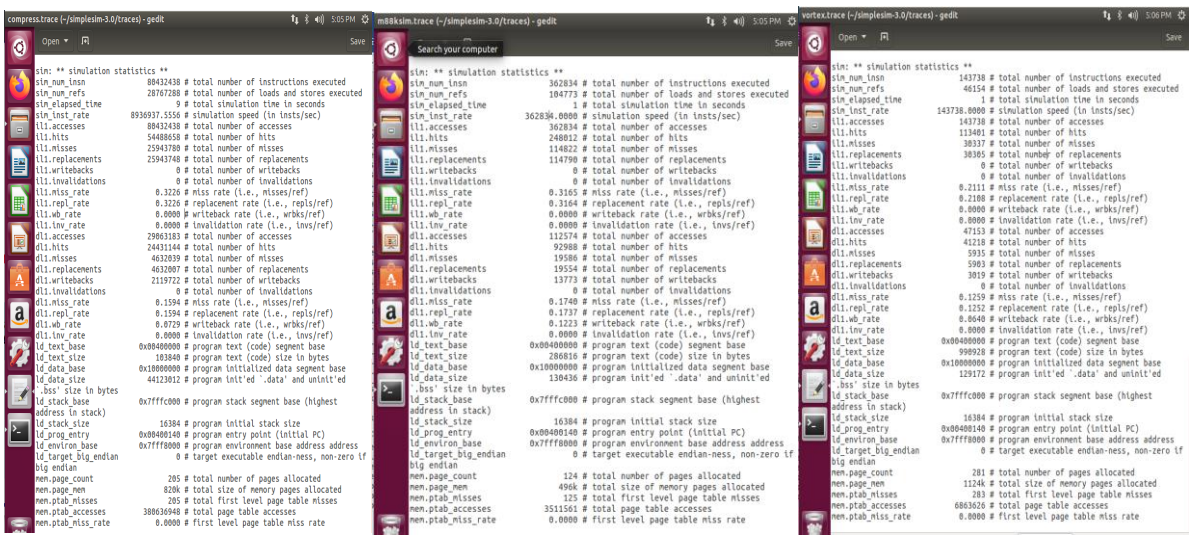
(split의 형태로 Cache를 L1만 사용한 input)

아래 결과들에서 L1 cache에 대한 Hit time은 1cycle, L2 cache에 대한 Hit time은 20cycle, 현재 위치에 관계없이 Main Memory에 접근할 경우 Hit time을 200cycle로 잡았다. 이를 바탕으로 AMAT는 L1만이 존재할시 L1(hittime=1)+L1(Missrate)*200, L2가 존재할시 L1(hittime=1)+L1(Missrate)*(L2(hittime=20)+L2(Missrate)*200)으로 계산하였다.



계산을 위한 Miss rate를 구하기 위해 위와 같은 결과들을 참고했으며 Miss rate의 계산은 (Miss된 instruction+Miss된 data)/(Instruction access수 + Data access수)로 계산하였다.

아래 기술된 내용들은 이러한 조건들을 바탕으로 서술된 것이며 나열된 순서는 compress95,m88ksim, vortex순으로 서술되어 있다.



Sim1

아래는 Cache 형태를 Instruction과 Data를 따로 접근할/하지않는 경우의 AMAT 계산 결과이다.

# of sets	Unified cache Miss rate	Unified cache AMAT	Split cache		Split cache AMAT
			Inst.Miss rate	Data Miss rate	
64	0.3070	62.4	0.3226	0.1594	56.84847818
128	0.1536	31.72	0.1991	0.1247	36.87398075
256	0.0965	20.3	0.0598	0.1050	15.35684446
512	0.0389	8.78	0.0392	0.0879	11.4273412

$$\frac{((80432438+29063183)-(54488658+24431144))}{(80432438+29063183)}\cdot 200+1 \qquad =56.84847818$$

$$\frac{((80432438+29063183)-(64415581+25439821))}{(80432438+29063183)}\cdot 200+1 \qquad =36.87398075$$

$$\frac{((80432438+29063183)-(75624386+26011177))}{(80432438+29063183)}\cdot 200+1 \qquad =15.35684446$$

$$\frac{((80432438+29063183)-(77279723+26507157))}{(80432438+29063183)}\cdot 200+1 \qquad =11.4273412$$

# of sets	Unified cache Miss rate	Unified cache AMAT	Split cache		Split cache AMAT
			Inst.Miss rate	Data Miss rate	
64	0.2960	60.2	0.3165	0.1740	57.54427355
128	0.2473	50.46	0.2786	0.1419	50.24023155
256	0.1763	36.26	0.2293	0.1300	48.35890014
512	0.1137	23.74	0.1599	0.1071	11.4273412

$$\frac{((362834+112574)-(248012+92988))}{(362834+112574)}\cdot 200+1 \qquad =57.54427355$$

$$\frac{((362834+112574)-(261763+96599))}{(362834+112574)}\cdot 200+1 \qquad =50.24023155$$

$$\frac{((362834+112574)-(279637+83197))}{(362834+112574)}\cdot 200+1 \qquad =48.35890014$$

$$\frac{((362834+112574)-(304821+100518))}{(362834+112574)}\cdot 200+1 \qquad =30.47741729$$

# of sets	Unified cache Miss rate	Unified cache AMAT	Split cache		Split cache AMAT
			Inst.Miss rate	Data Miss rate	
64	0.2059	42.18	0.2111	0.1259	39.00283932
128	0.1800	37	0.1892	0.0918	34.02827268
256	0.1430	29.6	0.1749	0.0456	29.59956729
512	0.1032	21.64	0.1322	0.0393	22.85540439

$$\frac{((143738+47153)-(113401+41218))}{(143738+47153)}\cdot 200+1 \qquad =39.00283932$$

$$\frac{((143738+47153)-(116543+42824))}{(143738+47153)}\cdot 200+1 \qquad =34.02827268$$

$$\frac{((143738+47153)-(118593+45001))}{(143738+47153)}\cdot 200+1 \qquad =29.59956729$$

$$\frac{((143738+47153)-(124733+45298))}{(143738+47153)}\cdot 200+1 \qquad =22.85540439$$

(*(Access수-Hit수)/(Access수)*(Miss Penalty)+Hit time 으로 계산)

위를 보면 알 수 있듯, set이 낮은 경우 전반적으로 AMAT가 낮은 쪽은 분리한 쪽이다. 허나 일정 수의 set 개수가 넘어가게 된 경우 분리한 경우의 AMAT가 더 커지는 것을 확인할 수 있다. 이는 Data와 Instuction에 대한 Cache의 사이즈를 Split Cache의 경우 나눠서 저장하기에 발생하는 현상으로, 통합하여 저장하는 Unified Cache의 경우 Instruction과 Data의 비율에 관계없이 이를 저장하여 더 많은 Hit를 위한 정보를 저장하고 있으나 Split cache의 경우 Instruction 혹은 Data중 하나의 Cache가 가득차게 되며 Cache내에서의 삭제가 이뤄지고 해당 값이 다시 입력되었을 때 Miss가 발생해 일어나게 된 것이다.

Instruction과 Data Cache비율이 1:1임을 바탕으로 Instruction과 data의 비율이 1:1과가장 가까운 것은 m88ksim임을 추측해볼 수 있다.

SIM2

위는 L2 Cache를 추가하고 Cache의 크기를 조절하며 AMAT를 확인한 경우이다.

L1I/L1D/L2U	Inst.Miss rate	Data.Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	0.4355	0.3685	0.0433	12.97087763
16/16/512	0.4035	0.2221	0.0910	14.57463029
32/32/256	0.3226	0.1594	0.2248	19.13958571
64/64/128	0.1991	0.1247	0.4225	19.74415494
128/128/0	0.0598	0.1050	(No Use L2)	15.35684446

$$1 + \frac{(35024917 + 10709859)}{(80432438 + 29063183)} \cdot (20 + 0.0433 \cdot 200) = 12.97087763$$

$$1 + \frac{(32454585 + 6455430)}{(80432438 + 29063183)} \cdot (20 + 0.0910 \cdot 200) = 14.57463029$$

$$1 + \frac{(25943780 + 4632039)}{(80432438 + 29063183)} \cdot (20 + 0.2248 \cdot 200) = 19.13958571$$

$$1 + \frac{(16016857 + 3623362)}{(80432438 + 29063183)} \cdot (20 + 0.4225 \cdot 200) = 19.74415494$$

$$\frac{((80432438 + 29063183) - (75624386 + 26011177))}{(80432438 + 29063183)} \cdot 200 + 1 = 15.35684446$$

L1I/L1D/L2U	Inst.Miss rate	Data.Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	0.3856	0.2995	0.1732	20.95556221
16/16/512	0.3350	0.2276	0.3201	27.0086479
32/32/256	0.3165	0.1740	0.5302	36.63420119
64/64/128	0.2786	0.1419	0.7931	44.9764508
128/128/0	0.2293	0.1300	No use	48.35890014

$$1 + \frac{(139917 + 33711)}{(362834 + 112574)} \cdot (20 + 0.1732 \cdot 200) = 20.95556221$$

$$1 + \frac{(121541 + 25623)}{(362834 + 112574)} \cdot (20 + 0.3201 \cdot 200) = 27.0086479$$

$$1 + \frac{(114822 + 19586)}{(362834 + 112574)} \cdot (20 + 0.5302 \cdot 200)| = 36.63420119$$

$$1 + \frac{(101071 + 15975)}{(362834 + 112574)} \cdot (20 + 0.7931 \cdot 200)| = 44.9764508$$

L1I/L1D/L2U	Inst.Miss rate	Data.Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	0.2785	0.2131	0.2027	16.88100277
16/16/512	0.2253	0.1595	0.4281	23.08109843
32/32/256	0.2111	0.1259	0.6404	29.13730233
64/64/128	0.1892	0.0918	0.8737	33.15962911
128/128/0	0.1749	0.0456	No use	29.59956729

$$1 + \frac{(40026 + 10049)}{(143738 + 47153)} \cdot (20 + 0.2027 \cdot 200)| = 16.88100277$$

$$1 + \frac{(32389 + 7519)}{(143738 + 47153)} \cdot (20 + 0.4281 \cdot 200)| = 23.08109843$$

$$1 + \frac{(30337 + 5935)}{(143738 + 47153)} \cdot (20 + 0.6404 \cdot 200)| = 29.13730223$$

$$1 + \frac{(27195 + 4329)}{(143738 + 47153)} \cdot (20 + 0.8737 \cdot 200)| = 33.15962911$$

L1 Hit time+ (L1 Miss수)/(L1 Access수)*(L1 Miss Penalty+(L2 Miss수)/(L2 Access수)*L2Misspenalty)

전반적으로 L2가 존재하는 경우, L2의 용량을 줄이고 L1Cache의 각 용량을 늘려줄 경우 L2의 Miss rate가 증가하는 경우를 볼 수 있으며 전반적으로 줄일 경우 L1에서 감소한 Miss rate에 비해 더 많이 증가되어 AMAT가 상승되는 모습을 보였다. 반면 L1Cache의 사이즈를 최소화하고 L2의 크기를 키운 경우, L1 Cache만을 사용한 경우보다 훨씬 낮은 Cycle을 소모하는 것을 확인할 수 있었다. 실제로 모든 경우 8/8/1024의 경우는 128/128/0을 사용한 경우보다 AMAT가 낮게 나오고 있다.

compress95,m88ksim, vortex순으로 access하는 수가 적어짐을 볼 때, L2의 용량을 늘리는 식의 방식은 access가 많아질수록 효과적임을 추측해볼 수 있다.

SIM3

해당 경우는 Associativity를 조절, way수를 변경하며 값을 확인한다.

# of Sets	Split Cache Miss rate/AMAT							
	1-way		2-way		4-way		8-way	
64	0.27924	56.84847	0.093761	19.75223	0.02346	5.69387	0.02083	5.16753
128	0.17936	36.87398	0.068648	14.72974	0.02113	5.22759	0.01802	4.60485
256	0.07178	15.35684	0.021479	5.295857	0.01811	4.62314	0.01475	3.95168
512	0.05213	11.42734	0.018295	4.659134	0.01485	3.97181	0.01196	3.39379
1024	0.02016	5.032080	0.015126	4.025264	0.01203	3.40748	0.00978	2.95762
2048	0.01712	4.424295	0.012438	3.487644	0.00991	2.98254	0.00787	2.575737

64 set AMAT

$$\frac{(25943780 + 4632039)}{(80432438 + 29063183)} \cdot 200 + 1| = 56.84847818$$

$$\frac{(7354582 + 2911855)}{(80432438 + 29063183)} \cdot 200 + 1| = 19.75223302$$

$$\frac{(59237 + 2510556)}{(80432438 + 29063183)} \cdot 200 + 1| = 5.693873557$$

$$\frac{(53064 + 2228569)}{(80432438 + 29063183)} \cdot 200 + 1| = 5.167532873$$

128 set AMAT

$$\frac{(16016857 + 3623362)}{(80432438 + 29063183)} \cdot 200 + 1| = 36.87398075$$

$$\frac{(4950520 + 2566213)}{(80432438 + 29063183)} \cdot 200 + 1| = 14.72974176$$

$$\frac{(51791 + 2262726)}{(80432438 + 29063183)} \cdot 200 + 1| = 5.227597376$$

$$\frac{(23810 + 1949771)}{(80432438 + 29063183)} \cdot 200 + 1| = 4.604858317$$

256 set AMAT

$$\frac{(4808052+3052006)}{(80432438+29063183)} \cdot 200+1| = 15.35684446$$

$$\frac{(49637+2302251)}{(80432438+29063183)} \cdot 200+1| = 5.29585764$$

$$\frac{(25194+1958398)}{(80432438+29063183)} \cdot 200+1| = 4.623143979$$

$$\frac{(2731+1613254)}{(80432438+29063183)} \cdot 200+1| = 3.951688817$$

512 set AMAT

$$\frac{(3152715+2556026)}{(80432438+29063183)} \cdot 200+1| = 11.4273412$$

$$\frac{(25637+1977659)}{(80432438+29063183)} \cdot 200+1| = 4.65913446$$

$$\frac{(4439+1622563)}{(80432438+29063183)} \cdot 200+1| = 3.971811996$$

$$\frac{(1750+1308801)}{(80432438+29063183)} \cdot 200+1| = 3.393796187$$

1024set AMAT

$$\frac{(28944+2178532)}{(80432438+29063183)} \cdot 200+1 = 5.032080881$$

$$\frac{(7680+1648586)}{(80432438+29063183)} \cdot 200+1| = 4.025264362$$

$$\frac{(1774+1316270)}{(80432438+29063183)} \cdot 200+1| = 3.407482579$$

$$\frac{(1749 + 1070010)}{(80432438 + 29063183)} \cdot 200 + 1| = 2.957628972$$

2048 set AMAT

$$\frac{(20552 + 1854175)}{(80432438 + 29063183)} \cdot 200 + 1| = 4.424295845$$

$$\frac{(4081 + 1357850)}{(80432438 + 29063183)} \cdot 200 + 1| = 3.487644689$$

$$\frac{(1749 + 1083649)}{(80432438 + 29063183)} \cdot 200 + 1| = 2.982541384$$

$$\frac{(1749 + 860933)}{(80432438 + 29063183)} \cdot 200 + 1| = 2.575737901$$

# of Sets	Split Cache Miss rate/AMAT							
	1-way		2-way		4-way		8-way	
64	0.28272	57.54427	0.241106	49.22131	0.20946	42.89243	0.1076801	22.536028
128	0.24620	50.24021	0.201256	41.25132	0.11251	23.50319	0.0458237	10.164759
256	0.20578	42.15748	0.121083	25.21667	0.04808	10.61742	0.0293810	6.8762157
512	0.14738	30.47741	0.068381	14.67625	0.03142	7.285548	0.0258577	6.1715579
1024	0.09324	19.64966	0.039690	8.938023	0.02618	6.376064	0.0256158	6.12317840
2048	0.06183	13.36790	0.029092	6.818581	0.02565	6.130330	0.0254181	6.08363342

64 set AMAT

$$\frac{(114822 + 19586)}{(362834 + 112574)} \cdot 200 + 1| = 57.54427355$$

$$\frac{(101253 + 13371)}{(362834 + 112574)} \cdot 200 + 1| = 49.22131727$$

$$\frac{(88564 + 11016)}{(362834 + 112574)} \cdot 200 + 1| = 42.89243765$$

$$\frac{(40703 + 10489)}{(362834 + 112574)} \cdot 200 + 1| = 22.536028$$

128 set AMAT

$$\frac{(101071 + 15975)}{(362834 + 112574)} \cdot 200 + 1| = 50.24023155$$

$$\frac{(84303 + 11376)}{(362834 + 112574)} \cdot 200 + 1| = 41.25132097$$

$$\frac{(43041 + 10450)}{(362834 + 112574)} \cdot 200 + 1| = 23.50319725$$

$$\frac{(12039 + 9746)}{(362834 + 112574)} \cdot 200 + 1| = 10.16475953$$

256 set AMAT

$$\frac{(83197 + 14636)}{(362834 + 112574)} \cdot 200 + 1| = 42.15748999$$

$$\frac{(46805 + 10759)}{(362834 + 112574)} \cdot 200 + 1| = 25.21667284$$

$$\frac{(13108 + 9753)}{(362834 + 112574)} \cdot 200 + 1| = 10.61742335$$

$$\frac{(4383 + 9585)}{(362834 + 112574)} \cdot 200 + 1| = 6.876215798$$

512 set AMAT

$$\frac{(58013 + 12056)}{(362834 + 112574)} \cdot 200 + 1| = 30.47741729$$

$$\frac{(22653 + 9856)}{(362834 + 112574)} \cdot 200 + 1| = 14.67625282$$

$$\frac{(5346 + 9595)}{(362834 + 112574)} \cdot 200 + 1| = 7.285548413$$

$$\frac{(2711+9582)}{(362834+112574)} \cdot 200 + 1| = 6.171557904$$

1024 set AMAT

$$\frac{(34057+10274)}{(362834+112574)} \cdot 200 + 1| = 19.64966513$$

$$\frac{(9252+9617)}{(362834+112574)} \cdot 200 + 1| = 8.938023761$$

$$\frac{(2868+9582)}{(362834+112574)} \cdot 200 + 1| = 6.237606435$$

$$\frac{(2629+9549)}{(362834+112574)} \cdot 200 + 1| = 6.123178407$$

2048 set AMAT

$$\frac{(19358+10041)}{(362834+112574)} \cdot 200 + 1| = 13.36790294$$

$$\frac{(4247+9584)}{(362834+112574)} \cdot 200 + 1| = 6.818581092$$

$$\frac{(2653+9542)}{(362834+112574)} \cdot 200 + 1| = 6.130330159$$

$$\frac{(2625+9459)}{(362834+112574)} \cdot 200 + 1| = 6.083633426$$

# of Sets	Split Cache Miss rate/AMAT							
	1-way		2-way		4-way		8-way	
64	0.19001	39.00283	0.159913	32.98264	0.13765	28.53089	0.1202937	25.05875
128	0.16514	34.02827	0.140467	29.09351	0.11247	23.49451	0.4404607	9.809215
256	0.14299	29.59956	0.109182	22.83654	0.04990	10.98161	0.0182041	4.640821
512	0.10927	22.85540	0.061050	13.21010	0.01915	4.830458	0.0171354	4.427086
1024	0.07342	15.68586	0.027554	6.510998	0.01759	4.519285	0.0169677	4.393559
2048	0.03956	8.913416	0.019330	4.866080	0.01696	4.393559	0.0169677	4.393559

64 set AMAT

$$\frac{(30337+5935)}{(143738+47153)} \cdot 200 + 1| = 39.00283932$$

$$\frac{(27399+3127)}{(143738+47153)} \cdot 200 + 1| = 32.98264978$$

$$\frac{(24691+1586)}{(143738+47153)} \cdot 200 + 1| = 28.53089459$$

$$\frac{(21734+1229)}{(143738+47153)} \cdot 200 + 1| = 25.05875604$$

128 set AMAT

$$\frac{(27195+4329)}{(143738+47153)} \cdot 200 + 1| = 34.02827268$$

$$\frac{(24958+1856)}{(143738+47153)} \cdot 200 + 1| = 29.09351934$$

$$\frac{(20236+1234)}{(143738+47153)} \cdot 200 + 1| = 23.49451258$$

$$\frac{(7198+1210)}{(143738+47153)} \cdot 200 + 1| = 9.80921573$$

256 set AMAT

$$\frac{(25145+2152)}{(143738+47153)} \cdot 200 + 1| = 29.59956729$$

$$\frac{(19561+1281)}{(143738+47153)}\cdot 200+1| = 22.83654546$$

$$\frac{(8312+1215)}{(143738+47153)}\cdot 200+1| = 10.98161254$$

$$\frac{(2294+1181)}{(143738+47153)}\cdot 200+1| = 4.640821202$$

512 set AMAT

$$\frac{(19005+1855)}{(143738+47153)}\cdot 200+1| = 22.85540439$$

$$\frac{(10436+1218)}{(143738+47153)}\cdot 200+1| = 13.21010943$$

$$\frac{(2474+1182)}{(143738+47153)}\cdot 200+1| = 4.83045822$$

$$\frac{(2102+1169)}{(143738+47153)}\cdot 200+1| = 4.427086662$$

1024 set AMAT

$$\frac{(12522+1495)}{(143738+47153)}\cdot 200+1| = 15.68586785$$

$$\frac{(4072+1188)}{(143738+47153)}\cdot 200+1| = 6.510998423$$

$$\frac{(2189+1170)}{(143738+47153)}\cdot 200+1| = 4.519285875$$

$$\frac{(2070+1169)}{143738+47153}\cdot 200+1| = 4.393559675$$

2048 set AMAT

$$\frac{(6252+1301)}{(143738+47153)} \cdot 200 + 1| = 8.913416557$$

$$\frac{(2518+1172)}{(143738+47153)} \cdot 200 + 1| = 4.866080643$$

$$\frac{(2070+1169)}{(143738+47153)} \cdot 200 + 1| = 4.393559675$$

$$\frac{(2070+1169)}{(143738+47153)} \cdot 200 + 1| = 4.393559675$$

위의 결과를 보면 알 수 있듯, Way를 늘릴수록, AMAT가 감소하는 모습을 볼 수 있다. 다만 way와 set이 일정이상 올라갈 경우 AMAT가 감소하는 량이 점점 줄어드는 것을 확인할 수 있으며 AMAT에 변화가 없는 경우도 확인할 수 있었다.

Sim4

Block size에 따른 Miss rate와 AMAT의 변화를 측정하였다.

Block size	Unified cache Miss rate	AMAT
16	0.0389	8.78
64	0.0126	3.52
128	0.0111	3.22
256	0.0042	1.84
512	0.0032	1.64

$$1 + 0.0389 \cdot 200 = 8.78$$

$$1 + 0.0126 \cdot 200 = 3.52$$

$$1 + 0.0111 \cdot 200 = 3.22$$

$$1 + 0.0042 \cdot 200 = 1.84$$

$$1 + 0.0032 \cdot 200 = 1.64$$

Block size	Unified cache Miss rate	AMAT
16	0.1137	23.74
64	0.0256	6.12
128	0.0093	2.86
256	0.0056	2.12
512	0.0053	2.06

$$1 + 0.1137 \cdot 200 = 23.74$$

$$1 + 0.0256 \cdot 200 = 6.12$$

$$1 + 0.0093 \cdot 200 = 2.86$$

$$1 + 0.0056 \cdot 200 = 2.12$$

$$1 + 0.0053 \cdot 200 = 2.06$$

Block size	Unified cache Miss rate	AMAT
16	0.1023	21.46
64	0.0245	5.9
128	0.0096	2.92
256	0.0066	2.32
512	0.0020	1.4

$$1 + 0.1023 \cdot 200 = 21.46$$

$$1 + 0.0245 \cdot 200 = 5.9$$

$$1 + 0.0096 \cdot 200 = 2.92$$

$$1 + 0.0066 \cdot 200 = 2.32$$

$$1 + 0.0020 \cdot 200 = 1.4$$

Block size가 커질수록 Miss rate는 감소하며 그에 따라 AMAT가 낮아지는 것을 확인할 수 있었다.

결과적으로 Block size가 클수록, Way가 많을수록, L2와 분리하여 L2의 크기가 클수록, Set이 적을 경우 Instruction과 data를 분해해서, 많을수록 합쳐서 저장할수록 빠르다는 결과를 위의 실험결과에 한해서 알아낼 수 있었다.

SIM1: compress95와 m88ksim의 경우 data와 instruction의 비율이 비슷해보이나, vortex의 경우 극단적으로 AMAT 감소량이 적은 것을 보아 vortex는 instruction과 data의 비율이 극단적일 가능성이 커보인다.

Sim2: compress95의 경우 64/64/128에서 급격하게 instruction Miss rate가 감소했다. 즉, compress95의 경우 이 수치 근처의 instruction을 반복해서 사용함을 추측할 수 있다.

m88ksim, vortex의 경우 극단적으로 감소하는 구간은 크게 없으며 비교적 규칙적으로 L2 cache의 Miss rate가 내려감을 볼 때 특정 범위에서 지속적으로 instruction이나 data를 사용하는 성향이 적어보인다.

Sim3: m88ksim, vortex의 경우 2way로 변경했을 때 일정 set 수 이상에서 극단적으로 줄어드는 것을 확인할 수 있다. 이 때문에 이 두 benchmark는 2개 단위로 Temporal Locality가 높을 가능성이 커보인다.

Sim4:모든 경우에서 block size를 늘렸을 때 4배 증가할 때마다 계속해서 AMAT가 크게 감소하는 모습을 볼 수 있다. 이를 통해 Data들은 Spartial Locality가 높은 편이라는 것을 알 수 있다.

위의 내용들을 바탕으로 추측하면 다음과 같다.

compress95:Instruction과 Data의 비율이 비교적 일정하며(약 1:2) 특정 instruction들을 반복해서 사용하며 비교적 temporal locality가 높은 편은 아니다. 또한 Sparitial Locality가 높은 편이다.

실제로는 in-memory형식의 data buffer를 생성하고 이를 압축해 다른 in memory buffer로 옮기고 이를 해제하는 방식으로 진행된다. 이 때문에 buffer에 맞는 형태의 access가 필요하며(단순하게는 1차원 배열도 고려해볼 수 있을 것이다.) 압축 및 해제가 필요하기에 타 benchmark에 비해 많은 access가 발생한다.

m88ksim: Instruction과 Data의 비율이 비교적 일정하며(약 2:1) 비교적 temporal locality가 높은 편이다. 또한 Sparitial Locality가 높은 편이다.

실제로는 실행파일을 나타내는 Binary image를 디스크 파일로부터 읽어오는 알고리즘이다. 이 때문에 n*m형태의 2차원 배열을 가지게 되며 이에 따라 2d array access를 하게 될 것이다.

vortex: Instruction과 Data의 비율이 비교적 일정하지 못하며 비교적 temporal locality가 높은 편이다. 또한 Sparitial Locality가 높은 편이다.

실제로는 공유 메모리 기반의 처리를 테스트하는 알고리즘을 지닌다. 압축 등의 복잡한 알고리즘이 비교적 적기 때문에 access수가 적으며 공유 메모리가 기반이기 때문에 기존의 상태가 비교적 잘 유지되기 때문에 단순한 access가 이뤄지게 될 것이다.

또한 모든 경우에서 지금은 Data Size가 크지 않아 결과가 극명하지만 Data Size가 커질수록 순서대로 대부분의 data를 확인하는 Bubble sort와 확률적으로 data를 찾아오는 Random Access가 비슷한 효율을 가져 결과적으로 둘이 비슷한 값을 가지게 될 것이다.

-Perfomance 측정

위의 내용을 결과로 각 BenchMark에 대해 적합한 Cache를 선택하고자 한다.

우리는 cache size가 2배 상승할 때마다 4%, associativity가 2배 상승할 때마다 2%씩 상승한다. 또한 Block size의 경우 HIT time이 늘어난 Blocksize만큼 증가한다고 가정하고 performance를 구한다.

현재 cache size가 2배될때마다 4%씩 증가한다. 이때 제시된 결과들 중에서는 number of set이 증가할 수록 올라가는 비율이 cycle 비율에 비해 크며 이 때문에 최적의 number of set은 제시된 값 중 512가 된다.

이는 Sim2에도 적용된다. L1I, L1D, L2U또한 비율을 변경하며 총 cache량을 반에 가까이 줄여도 AMAT감소량이 (1.04)^n을 넘는 모양이며(12.97*1.04=13.48<14.57) L1만 사용한 128/128/0과 비교해도 (12.97)*(1.04)*(1.04)<15.35... 와 같이 L1만 사용한 것 보다 큰 모습이다. 즉 모든 경우에서 8/8/1024가 가장 최적의 형태임을 알 수 있다.(전체에 1.04를 곱한 것이 기본 cycle에 1.04를 곱한

것 보다 큰 결과를 낸다. 그럼에도 크게 나온 것을 확인할 수 있다.)

Sim3에서 way의 경우 2배 증가마다 1.02만큼 상승함을 이용, 현재 사용 예정인 8/8/1024를 기준으로 number of set이 1024라고 가정하고 볼 시 각각 모두 8way일 때 최적이 된다.

$$\frac{(2070+1169)}{143738+47153}\cdot 200\cdot 1.02+1\cdot 1.02 \qquad = 4.481430869$$

가장 비슷한 vortex의 1024의 4way 8way비교도 위와 같이 8way가 더 적게 나온다.

끝으로 sim4에서 Block size를 2배 늘릴 경우, 기본 cycle도 1.04배 는다고 가정하면 compress95의 경우 512(1.84>1.64*1.04), m88ksim에서는 256(2.12<2.06*1.04), vortex에서는 512(2.32>1.4*1.04)이다.

결과적으로 각 BenchMark에 대하여 최적의 결과는 L1I/L1D/L2U가 8/8/1024인 상태로 8way이며 Block size가 compress95, vortex의 경우 512, m88ksim의 경우 256일 경우 최적의 캐시가 된다.

단, 이 경우는 상승하는 Memory size에 비해 증가하는 perfomance가 적기 때문에 위와 같이 거의 높을수록 좋은 결과가 나오게 되는 것이며, 실제 캐쉬에서는 이렇지 않을 것이다.

Cost의 경우 메모리 사이즈가 커질수록 price가 커진다고 되어있다. 따로 비율이 언급되어 있지 않으므로 Cost가 Performance에 대해 1:1의 비율을 가진다고 가정한다면, (Memory Size가 2배가 되면 Cost도 2배가 되며 Performance도 이에 따라 2배가 된다고 가정) Block Size를 제외한 모든 경우에서 Number of set을 2배로 올릴 경우 AMAT가 2배 이상 차이나는 경우는 존재하지 않는다. 만약 Block Size를 2배로 올릴 때마다 Memory Size도 2배 상승시킨다고 가정한다면 각 경우 21.46/4<5.9, (21.46/8=2.6825)<2.92...로 가장 작은 Block Size인 16을 가질 때 가장 효율적이게 된다.

고찰

위에서 사용한 방법은 위에서 측정된 결과만을 바탕으로 최적의 캐쉬를 찾아낸 것이며 이 때문에 해당 결과는 최적의 캐쉬가 아닐 수 있다. 또한 Performance에 대해 감소하는 AMAT 대비 performance penalty가 크지 않다는 점도 크게 작용하여 실제로 적용하기에 적합하지 않은 결과가 나온 것 같다. 추후 이러한 작업을 진행하여야 한다면 단순히 많은 결과를 돌려 최적의 값을 찾는 것보다 Performance와 각 요소간 AMAT변화에 대한 식을 찾아내어 최적의 방향성을 모색하는 것이 효율이 좋아 보인다. 추가적으로 실제로는 L3 혹은 그 이상의 Cache를 실 컴퓨터에 적용하는 경우가 많은 것으로 알고 있는데 이러한 경우의 AMAT 개선량 또한 관심이 생겼다.