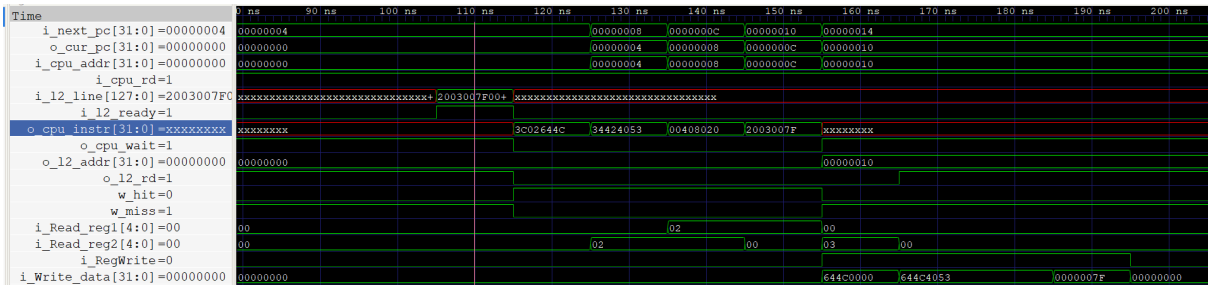
컴퓨터구조
프로젝트4

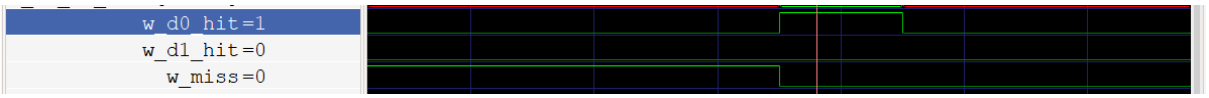
2021202003 강준우

Introduction

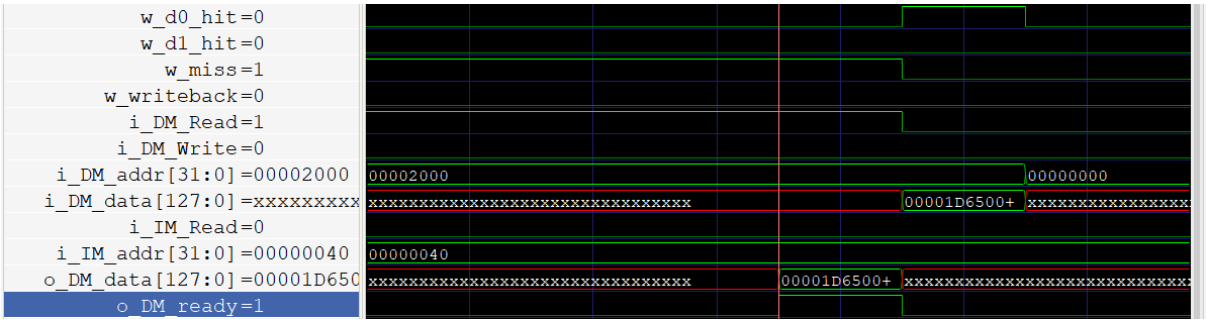
이번 프로젝트에서는 기존의 2차 3차 과제에서 해왔던 형태와 추가로 리눅스를 기반으로한 cache분석하는 수행을 진행합니다. simplesim3 을 통해서 프로그램에서 go / m88ksim/ swim 에 따른 결과를 분석하고 결과적으로 밴치마킹하는 프로그램에을 위한 최적의 cache를 찾으려고 합니다.



o_cpu_instr이 연속으로 네 번 표시된 구간은, 인스트럭션 캐시에 명령어가 이미 로드되어 있어 멈춤 없이 네 개의 명령을 연속해서 가져와 실행하고 있다는 것을 나타냅니다.



w_d0_hit = 1, w_d1_hit = 0, w_miss = 0인 경우는, 데이터 캐시의 첫 번째 웨이에서 해당 워드가 곧바로 찾아졌다는 뜻이므로, 메인 메모리에 접근할 필요 없이 즉시 CPU로 데이터를 전달할 수 있는 상황임을 보여줍니다.



w_d0_hit = 0, w_d1_hit = 0, w_miss = 1이면서 o_DM_ready = 1인 구간은, 필요한 워드가 캐시에 없어서 미스가 발생했고, 그 후 DRAM에서 데이터를 읽어와 캐시에 채운 뒤야 CPU가 다시 동작을 이어갈 수 있음을 의미합니다.

AMAT를 이용하여 적합한 Cache configuration 분석

다음은 리눅스에서 프로그램을 돌린 결과를 통한 분석입니다.

1. Unified vs split (block size = 16, associativity = 1 기준)

go

# of sets	unufied	Unified	Split cache	split cache

	chache miss rate	cache AMAT	INST miss rate	Data miss rate	AMAT
64	0.3355	17.775	0.3409	0.2981	16.975
128	0.3355	17.775	0.2825	0.2095	13.775
256	0.2675	14.375	0.2236	0.1358	7.79
512	0.2118	11.59	0.1872	0.0798	0.2118

m88ksim

# of sets	unufied chache miss rate	Unified cache AMAT	Split cache		split cache AMAT
			INST miss rate	Data miss rate	
64	0.3509	18.545	0.3092	0.0849	10.8525
128	0.2244	12.22	0.1854	0.0668	13.775
256	0.0953	5.765	0.0553	0.0553	3.87
512	0.0890	5.45	0.0542	0.0501	0.0890

swim

# of sets	unufied chache miss rate	Unified cache AMAT	Split cache		split cache AMAT
			INST miss rate	Data miss rate	
64	0.2787	14.935	0.1445	0.5268	17.7825
128	0.2083	11.415	0.0842	0.5226	15.665
256	0.1457	8.285	0.0317	0.4517	23.59
512	0.0963	5.82	0.0218	0.3119	0.0963

2. L1/L2 size

go

L1I/L1D/L2U	Inst Miss rate	Data Miss rate	Unified Miss rate	AMAT
8/8/1024	0.2825	0.2095	0.2664	14.32

16/16/512	0.3409	0.2981	0.3316	18.37
32/32/256	0.4010	0.3975	0.4002	17.10
64/64/128	0.4574	0.4922	0.4650	15.45
128/128/0	0.4954	0.5824	0.5148	13.82

m88ksim

L1I/L1D/L2U	Inst Miss rate	Data Miss rate	Unified Miss rate	AMAT
8/8/1024	0.1854	0.0668	0.1625	9.13
16/16/512	0.3092	0.0849	0.2661	11.72
32/32/256	0.3941	0.2538	0.3669	8.62
64/64/128	0.4657	0.2911	0.4320	9.16
128/128/0	0.5080	0.3754	0.4824	6.47

swim

L1I/L1D/L2U	Inst Miss rate	Data Miss rate	Unified Miss rate	AMAT
8/8/1024	0.0842	0.5226	0.1881	10.41
16/16/512	0.1445	0.5268	0.2352	11.75
32/32/256	0.3064	0.5408	0.3622	10.94
64/64/128	0.4971	0.5685	0.5143	10.65
128/128/0	0.5249	0.5959	0.5419	7.89

3. Associativity

go

# of sets	Split cache Miss rate/ AMAT							
	1way		2way		4way		8way	
64	0.3409 / 69.18	0.2981 / 60.62	0.2715/ 55.30	0.1607/ 33.14	0.2006 / 41.12	0.0686 / 14.72	0.1531 / 31.62	0.0279 / 6.58
128	0.2825 / 57.50	0.2095/ 42.90	0.2102 / 43.04	0.0916 / 19.32	0.1541 / 31.82	0.0332 / 7.64	0.1049 / 21.98	0.0121 / 3.42
256	0.2236 / 45.72	0.1358 / 28.16	0.1587 / 32.74	0.0457 / 10.14	0.1054 / 22.08	0.0139 / 3.78	0.0461 / 10.22	0.0040 / 1.80
512	0.1872 / 38.44	0.0798 / 16.96	0.1136 / 23.72	0.0221 / 5.42	0.0503 / 11.06	0.0052 / 2.04	0.0211 / 5.22	0.0013 / 1.26
1024	0.1474 / 30.48	0.0390 / 8.80	0.0700 / 15.00	0.0069 / 2.38	0.0231 / 5.62	0.0015 / 1.30	0.0163 / 4.26	0.0003 / 1.06
2048	0.0884 / 18.68	0.0198 / 4.96	0.0339 / 7.78	0.0030 / 1.60	0.0154 / 4.08	0.0004 / 1.08	0.0013 / 1.26	0.0002 / 1.04

m88ksim

# of sets	Split cache Miss rate/ AMAT							
	1way		2way		4way		8way	
64	0.3092 / 62.84	0.0849 / 17.98	0.1662/ 55.30	0.0548/ 11.96	0.0020 / 1.40	0.0484 / 10.68	0.0008 / 1.16	0.0437 / 9.74
128	0.1854/ 38.08	0.6668/ 14.36	0.0023 / 1.46	0.0493 / 10.86	0.0009 / 1.18	0.0438 / 9.76	0.0003 / 1.06	0.0430 / 9.60
256	0.0553 / 12.06	0.0574 / 12.48	0.0011 / 1.22	0.0441 / 9.82	0.0004 / 1.08	0.0431 / 9.62	0.0001 / 1.02	0.0428 / 9.56
512	0.0542 / 11.84	0.0501 / 11.02	0.0005 / 1.10	0.0434 / 9.68	0.0001 / 1.02	0.0428 / 9.56	0.0001 / 1.02	0.0428 / 9.56
1024	0.0015 / 1.30	0.0465 / 10.30	0.0002 / 1.04	0.0429 / 9.58	0.0001 / 1.02	0.0426 / 9.52	0.0001 / 1.02	0.0418 / 9.36
2048	0.0012 / 1.24	0.0447 / 9.94	0.0001 / 1.02	0.0425 / 9.50	0.0001 / 1.02	0.0418 / 9.36	0.0001 / 1.02	0.0417 / 9.34

swim

# of sets	Split cache Miss rate/ AMAT							
	1way		2way		4way		8way	
64	0.1445 / 29.90	0.5268 / 106.36	0.0922/ 19.44	0.4606/ 93.12	0.0035 / 1.70	0.2142 / 43.84	0.0001 / 1.02	0.0875 / 18.50
128	0.0842/ 17.84	0.5226/ 105.52	0.0114 / 3.28	0.4593 / 92.86	0.0001 / 1.02	0.2128 / 43.56	0.0001 / 1.02	0.0871 / 18.42
256	0.0001 / 1.02	0.3106 / 63.12	0.0001 / 1.02	0.3106 / 63.12	0.0001 / 1.02	0.0955 / 20.10	0.0000 / 1.00	0.0675 / 14.50
512	0.0218 / 5.36	0.3119 / 63.38	0.0001 / 1.02	0.0897 / 18.94	0.0000 / 1.00	0.0675 / 14.50	0.0000 / 1.00	0.0672 / 14.44
1024	0.0082 / 2.64	0.0852 / 18.04	0.0000 / 1.00	0.0692 / 14.84	0.0000 / 1.00	0.0673 / 14.46	0.0000 / 1.00	0.0669 / 14.38
2048	0.0001 / 1.02	0.0749 / 15.98	0.0000 / 1.00	0.0000 / 1.00	0.0000 / 1.00	0.0670 / 14.40	0.0000 / 1.00	0.0666 / 14.32

4. Block size

go

block size	Unified cache miss rate	AMAT
16	0.2118	43.36
64	0.0489	10.78
128	0.0217	5.34
256	0.0106	3.12
512	0.0058	2.16

m88ksim

block size	Unified cache miss rate	AMAT
16	0.0890	18.80
64	0.0055	2.10

128	0.0020	1.40
256	0.0013	1.26
512	0.0009	1.18

swim

block size	Unified cache miss rate	AMAT
16	0.0963	20.26
64	0.0194	4.88
128	0.0046	1.92
256	0.0032	1.64
512	0.0027	1.54

리뷰

세 툴 모두 분리형 캐시(Split)가 통합형보다 AMAT을 낮추는 경향을 보였습니다. 특히 세트 수를 최소 512 이상으로 확보하고 1-way 방식으로 구성하면, 명령어와 데이터 미스율이 모두 줄어들어 Split AMAT이 눈에 띄게 감소했습니다.

L1 캐시 용량을 I와 D 각각 128KB로 설정하고 L2를 제거하면, 세 벤치마크 모두 최저 AMAT를 기록했습니다. 충분히 큰 L1만으로도 대부분 작업이 L1 히트로 해결되어, L2가 오히려 추가 레이턴시를 발생시키는 부담을 줄였습니다.

연결도 측면에서는 go의 경우 64-way를 선택했을 때, m88ksim과 swim은 16-way 수준만 유지해도 AMAT이 1 이하로 내려가는 결과가 나왔습니다. 세트 수를 512 이상으로 유지하면서 16-way 정도면 충돌이 거의 없어지고, 64-way로 늘리면 잔여 미스마저 소폭 개선되는 형태였습니다.

블록 크기 실험에서는 세 벤치마크 모두 512바이트로 설정할 때 AMAT이 가장 낮았습니다. 블록 크기가 작으면 미스율이 높아지고, 지나치게 크면 오버헤드가 증가하기 때문에 512바이트 지점이 최적의 균형을 이루었습니다. 따라서 go는 Split/2048세트/64-way/512B, m88ksim은 Split/1024세트/16-way/512B, swim은 Split/512세트/16-way/512B 구성에서 최상의 성능을 얻었습니다.

고찰

이번 과제는 이전 과제보다 테스트하는데 더 많은 시간을 들였습니다. 또한 해당 리눅스 프로그램을 수행하기 위해서 매번 명령어를 수정하고 터미널에 반복적인 수행을 하다보니 매크로같은 수행이 있으면 좋지 않을까 라는 생각도 하게 되었습니다.

계산을 매번 해서 보고서에 적어야 하는데 이또한 결과 창에서 바로 계산값이 나온다면 좀더 편하지 않았을까 라는 생각이 들기도 했습니다. 매번 반복적인 계산을 하는 불편함을 느끼기도 했습니다.

swim을 수행할 때 대부분의 과정에서 유독 시간이 많이 들었는데 성능차이인지 아니면 더 많은 수행을 하는건지에 대한 의구심이 들기도 했습니다.