컴퓨터구조실험 보고서

Project #4 – Cache Design

과 목 컴퓨터구조실험

담당교수 이성원교수님

학 과 컴퓨터정보공학부

학 번 2021202058

이 름 송채영

1. Introduction

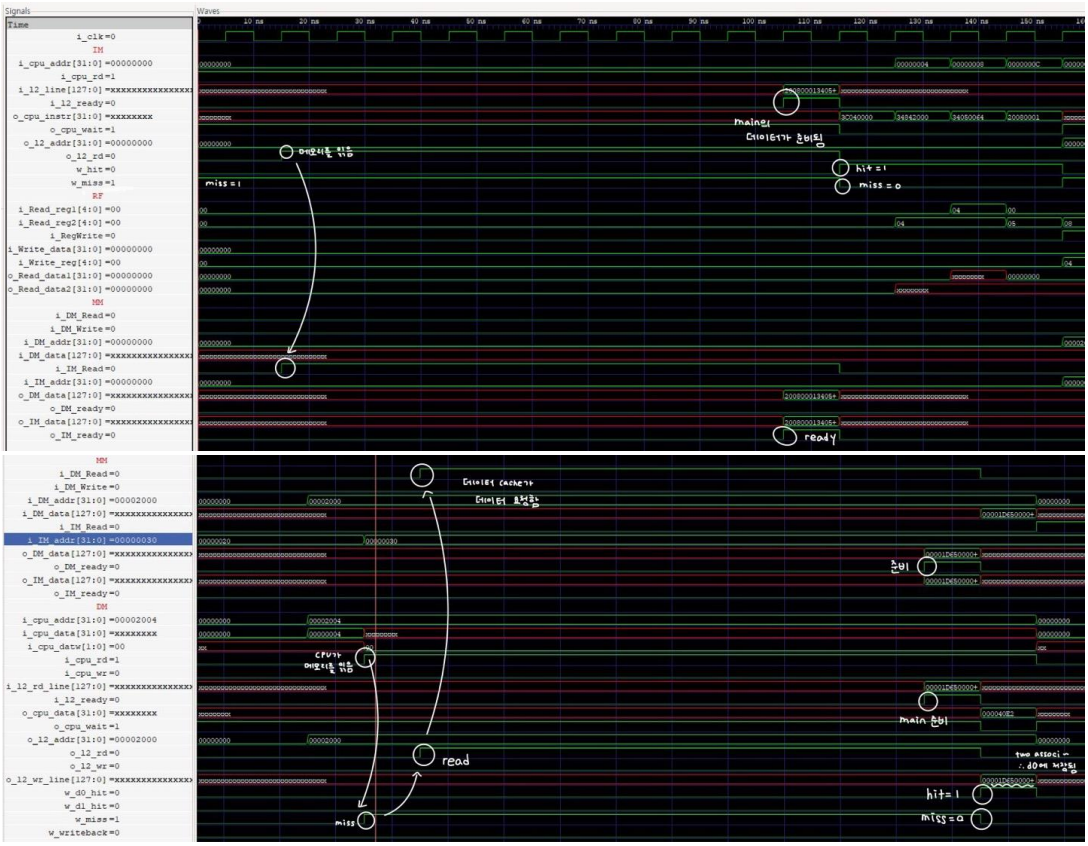
Cache 는 작고 빠른 메모리로, 사용자가 필요할 때 main memory 에 접근하지 않고 데이터를 빠르게 얻을 수 있도록 한다. 이번 프로젝트에서는 waveform 을 통해 Insertion sort 와 Random access 중 더 좋은 성능을 가지고 있는 것을 찾아본다. Insertion Sort 는 간단하고 직관적인 정렬 알고리즘으로, 배열이나 리스트를 정렬하는 데 사용된다. 이 알고리즘은 초기에 정렬된 부분이 비어 있는 상태에서 시작하여 순서대로 정렬되는 부분을 점진적으로 구축하는 방식으로 동작한다. Random Access 는 메모리 시스템에서 순차적이거나 연속적이지 않은 순서로 데이터 요소에 접근하는 능력을 의미한다. 순차적인 액세스와 달리, Random access 는 이전 요소를 거치지 않고 원하는 데이터 요소를 직접적으로 검색하고 접근할 수 있다. 두 프로그램 수행 중 instruction 을 가져올 때 cache miss 와 hit 이 어떻게 발생하는지, 그리고 miss 가 발생한다면 memory 에서 instruction 을 어떻게 가져오는지에 대해 확인한다. Data 에 대해서도 instruction 과 같은 방식으로 수행한다. 두 번째로 과제에서 주어진 3 개의 벤치마크(cc1, ilpeg, perl)를 통해 cache access 의 결과를 비교하고 각 벤치마크에 가장 적합한 cache 가 무엇인지 찾아본다.
벤치마크 Benchmark 는 컴퓨터 시스템의 성능을 평가하거나 비교하기 위해 사용되는 표준화된 테스트 또는 측정 도구를 말한다. 벤치마크는 일련의 프로그램 또는 작업으로 구성되어 시스템의 성능, 처리 속도, 메모리 사용량 등을 측정하고 평가할 수 있다.
벤치마크를 통한 cache access 의 성능을 비교하기 위해서는 AMAT 를 사용하는데, AMAT 는 Average Memory Access Time 을 의미하며 cache 에서 필요한 데이터를 가져오는데 걸리는 평균 시간을 뜻한다. 즉 AMAT 이 작을수록 data 를 더 빠르게 가져올 수 있어 더 좋은 성능을 가졌다는 것을 알 수 있다. 3 개의 벤치마크에 대해 간단하게 설명해보면 다음과 같다.
cc1 은 GNU C 컴파일러 버전 2.5.3 을 기반으로 한 컴파일러로, C 언어 소스 코드를 기계어로 변환하는 작업을 담당한다. 다음으로 ijpeg 유틸리티는 메모리상의 이미지에 대한 압축 및 압축 해제 작업을 수행하는 도구를 말한다. 주로 JPEG 이미지 형식에 대한 압축 및 해제 작업을 처리하는데 사용된다. 마지막으로 Perl 인터프리터는 perl 언어의 실행을 담당하는 인터프리터로, 문자열 조작, 정규 표현식, 파일 입출력, 네트워크 통신 등을 지원한다.

2. Assignment

2.1 Observation for Program Behaviors

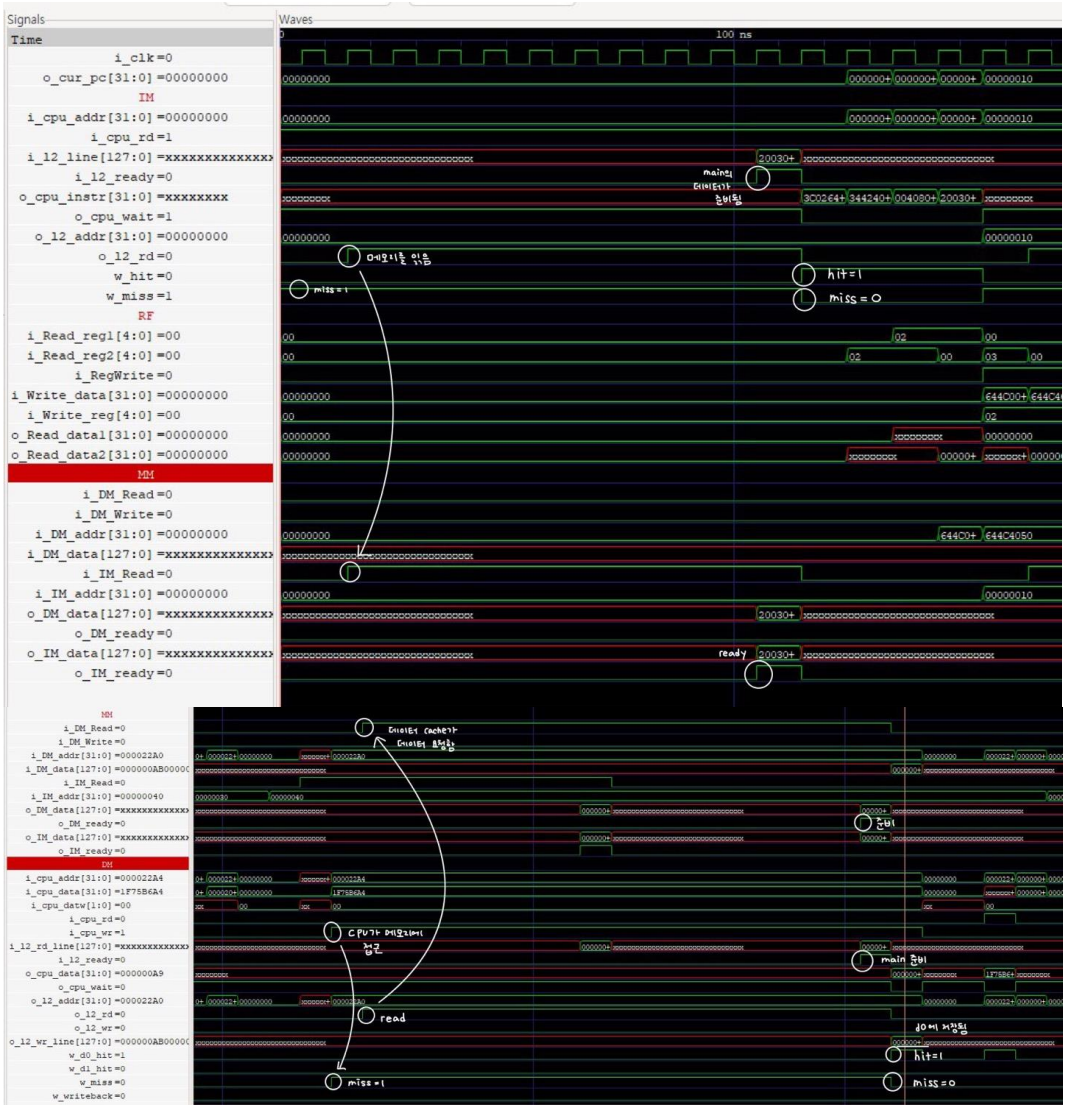
첫 번째 phase 로 Insertion sort 와 Random access 의 타이밍을 확인하고, 과제에서 요구하는 I/D L1 이 hit 할 때와 miss 할 때 어떻게 동작하는지, main memory 인 L2 가 언제 어떻게 동작하는지, Insertion sort 와 Random access 의 동작 측면에서의 차이점이 무엇인지, 데이터 크기가 증가하면 어떻게 되는지에 대해 알아보면 아래와 같다.

<Insertion sort>



우선 instruction Memory 에서 w_miss 가 1, o_l2_rd 가 1 이므로 메모리를 읽고 mm(main memory)에서도 i_IM_Read 를 1 로 설정된 것을 볼 수 있다. i_l2_ready 가 1 로 설정되어 main 의 데이터가 준비됨을 알 수 있고 o_IM_ready 도 1 로 설정된 것을 확인할 수 있다. w_hit 과 w_miss 가 각각 1, 0 이 된 것을 확인할 수 있으며 Instruction cache 의 hit 과 miss 동작방식에 대해 설명해보았다. 다음으로 Data Memory 에서 i_cpu_rd 가 1 로 설정되어 cpu 가 메모리를 읽어 miss 가 1 인 것을 확인한 후 o_l2_rd 를 1 로 설정하고 Main memory 에서는 i_DM_read 이 1 이 되어 l1 cache 가 데이터를 필요로 하는 것을 알 수 있다. 데이터가 준비가 되면 o_DM_ready 가 1 이 되어 i_l2_ready 가 1 이 됨을 보고 l1 cache 가 데이터를 가져가 d0 에 저장되는 것을 확인할 수 있다. 이때 2-ways set associative 방식에 의해 d0 가 선택됐다는 것을 알 수 있다. 또한 w_d0_hit 과 w_d0_miss 가 각각 1, 0 이 된 것을 확인할 수 있으며 Data cache 의 hit 과 miss 동작방식에 대해 설명해보았다.

<Random access>



다음으로 Random access 의 동작 방식에 대해 설명해 보겠다. Random access 는 insertion sort 의 동작 방식과 동일하다. 우선 instruction Memory 에서 w_miss 가 1, o_l2_rd 가 1 이므로 메모리를 읽고 mm(main memory)에서도 i_IM_Read 를 1 로 설정된 것을 볼 수 있다. i_l2_ready 가 1 로 설정되어 main 의 데이터가 준비됨을 알 수 있고 o_IM_ready 도 1 로 설정된 것을 확인할 수 있다. w_hit 과 w_miss 가 각각 1, 0 이 된 것을 확인할 수 있으며 Instruction cache 의 hit 과 miss 동작방식에 대해 설명해보았다. 다음으로 Data Memory 에서 i_cpu_rd 가 1 로 설정되어 cpu 가 메모리에 접근해 miss 가 1 인 것을 확인한 후 o_l2_rd 를 1 로 설정하고 Main memory 에서는 i_DM_read 이 1 이 되어 l1 cache 가 데이터를 필요로 하는 것을 알 수 있다. 데이터가 준비가 되면 o_DM_ready 가 1 이 되어 i_l2_ready 가 1 이 됨을 보고 l1 cache 가 데이터를 가져가 d0 에 저장되는 것을 확인할 수 있다. 이때 2-ways set associative 방식에 의해 d0 가 선택됐다는 것을 알 수 있다. 또한 w_d0_hit 과 w_d0_miss 가 각각 1, 0 이 된 것을 확인할 수 있으며 Data cache 의 hit 과 miss 동작방식에 대해 설명해보았다.

<Insertion sort and Random access 의 차이>

Insertion sort 와 Random access 는 Instruction L1 cache 의 동작은 동일하지만 Data L1 cache 의 동작은 다르다. Insertion sort 는 Main memory 에 값을 차례대로 정렬을 해서 주소를 가져와 인접한 주소들을 block 단위로 Data L1 Cache 에 저장하지만, Random access 는 값을 random 으로 가져와 저장하기 때문에 Main memory 에서 값을 가져와 다른 주소에 접근할 수 있어 Miss 발생율이 높다.

추가적으로 data size 가 증가한다고 가정했을 때에 대해 서술하겠다. 데이터 크기가 32 비트에서 64 비트로 증가하면 정해진 블록 크기는 4 워드인데, 이로 인해 저장되는 명령어 개수는 4 개에서 2 개로 줄어들게 된다. 이로 인해 Miss 비율은 증가하고, Hit 비율은 감소하게 된다. 또한, 데이터 크기를 증가시킴으로써 더 많은 데이터를 더 빠른 속도로 프로세서에 제공할 수 있어 성능이 향상될 수 있지만, L1 캐시와 같이 데이터를 가장 빠르게 전달하는 캐시는 크기가 작아야 한다. 따라서, 데이터 크기가 증가하면 캐시 내부에 저장할 수 있는 데이터의 양이 줄어들기 때문에 Miss 발생률이 상대적으로 높아질 수 있다.

2.2 Cache Simulation

두 번째 phase 로 주어진 벤치마크 프로그램에 가장 적합한 cache 를 찾고 차이점을 비교해본다.

우선 각각의 벤치마크 프로그램의 original purpose 에 대해 말해보겠다. Introduction 에서 간단하게 설명한 것과 같다. 자세하게 설명해보겠다. cc1 은 CNU C 컴파일러(GNU Compiler Collection, GCC)의 일부로 C 언어 소스 코드를 기계어로 변환하는 역할을 담당한다. 이 프로그램은 초기에 C 언어로 작성된 소스 코드를 분석하고, 중간 코드를 생성하며 최적화를 수행하여 실행 가능한 프로그램으로 변환한다. cc1 은 컴파일러 최적화와 코드 생성 기술의 연구 및 개발을 위한 도구로 널리 사용된다. 이를 통해 컴파일러의 성능, 최적화 효과, 메모리 사용량 등을 평가하고 개선되는데 활용된다. SPEC CINT95 벤치마크 스위트의 일부로도 사용되며, 다양한 컴파일러 및 컴파일러 설정의 비교 평가에도 활용된다. 다음으로 ijpeg 는 메모리 상의 이미지에 대한 압축 및 압축 해제 작업을 수행하는 유틸리티로 JPEG(Joint Photographic Experts Group) 이미지 형식에 대한 압축 및 해제 작업을 처리하는데 사용된다. ijpeg 는 이미지 데이터를 메모리에 load 하고 압축 알고리즘을 적용하거나 압축을 해제하여 원래 이미지를 복원한다. 이 프로그램은 이미지 처리 및 압축 기술의 성능을 평가하기 위해 사용된다. 압축 속도, 압축률, 압축 해제 속도 등을 측정하여 다양한 압축 알고리즘과의 성능 비교 및 최적화에 활용된다. SPEC CINT95 벤치마크 스위트의 일부로서 이미지 처리 및 압축 기술에 대한 연구 및 평가에도 활용된다. 마지막으로 perl 은 강력한 스크립트 프로그래밍 언어로 알려져 있으며, perl 인터프리터는 perl 언어의 실행을 담당한다. perl 인터프리터는 perl 스크립트를 해석하고 실행해 원하는 동작을 수행한다. 문자열 조작, 정규 표현식, 파일 입출력, 네트워크 통신 등 다양한 작업을 지원하며 특히 시스템 관리, 웹

개발, 데이터 처리 등의 영역에서 널리 사용된다.

다음으로 알고리즘 측면에서 벤치마크 프로그램의 memory access pattern 에 대해 말해보겠다. 우선 cc1 은 컴파일러이기 때문에 주로 메모리의 읽기 및 쓰기 작업이 발생한다. 위에서 말한 것처럼, cc1 은 C 언어 소스 코드를 분석하고, 중간 코드를 생성하며 최적화를 수행하여 기계어로 변환한다. 이 과정에서 소스 코드 및 중간 코드의 변수, 배열, 구조체 등의 데이터에 대한 액세스가 필요하다. 메모리 액세스 패턴은 주로 변수 또는 배열 요소의 읽기와 쓰기 그리고 포인터 연산 등으로 구성된다. 또한 최적화 과정에서 임시 변수 및 최적화에 필요한 데이터 구조에 대한 메모리 액세스도 발생할 수 있다. 다음으로 ijpeg 는 이미지 압축 및 압축 해제를 수행하는 유틸리티이기 때문에 메모리 액세스 패턴은 이미지 데이터에 집중된다. 이미지 압축은 주로 이미지 데이터의 읽기 작업을 수반하며, 압축 알고리즘을 적용하여 데이터를 반환한다. 이후 압축 해제 단계에서는 압축된 데이터를 읽고, 압축을 해제해 원래 이미지 데이터로 복원한다. 따라서 메모리 액세스 패턴은 주로 이미지 데이터의 일기와 쓰기 작업으로 구성된다. 마지막으로 perl 인터프리터는 perl 스크립트를 실행하는 역할이므로, 메모리 액세스 패턴은 스크립트에 포함된 작업에 따라 다양하다. perl 은 다양한 작업을 지원하기 때문에 메모리 액세스 패턴은 문자열 처리, 배열 조작, 파일 입출력 등에 관련된 작업에 초점을 맞출 수 있다. 따라서 perl 인터프리터의 메모리 액세스 패턴은 스크립트에 포함된 작업 특성에 따라 다양하게 변할 수 있다.

아래는 여러 캐시 구성으로 실험을 수행한 결과를 표로 나타낸 것이다. Simulation1 부터 Simulation4 까지 있으며, 각각에 대해 설명해보면 다음과 같다. unified 와 split 두 방식과 set 의 수에 따른 cache size 를 바탕으로 cache access 의 성능을 구해보고, L1 과 L2 를 사용하여 두 cache 의 size 에 따른 cache access 의 성능을 구해본다. 다음으로 set 수에 따른 cache size 와 associativity 수를 통해 cache access 의 성능을 구해보고 마지막으로 block size 에 따라 cache access 의 성능을 구해 비교해본다.

Introduction 에서 설명했듯이, cache access 성능을 비교하기 위해 AMAT(Average Memory Access Time), cache 에서 필요한 데이터를 가져오는데 걸리는 평균 시간을 구해야 하는데 AMAT 를 구하는 공식과 그 외에 실험에 필요한 식에 대해 알아보면 다음과 같다.

* split cache AMAT = %instr X (instr hit time + instr miss rate X instr miss penalty) + %data X (data hit time t data miss rate X data miss penalty
* unified cache AMAT = hit time + miss rate x miss penalty(miss penalty = main memory access time)
* main memory access time = 200 cycle
* miss rate = miss 개수 / 전체 access 수

simulation 1,2 는 block size =16, associativity = 1 로, simulation 3 은 block size = 16 으로, simulation 4 는 number of sets = 512, associativity = 1 로 고정해서 실험을 진행하였다. (아래에서 해당 정보는 생략함)

또한 simulation2 의 경우 127/128/0 대신 none 을 넣어 실험을 진행하였고, miss rate 는 1 로 고정하였다.

<cc1>

Simulation 1. unified vs splits

# of Sets	Unified cache Miss rate	Unified Cache AMAT	Split cache		Split cache AMAT
			Inst. Miss rate	Data Miss rate	
64	$\frac{132616235}{388757301} = 0.3411$	69.5015	$\frac{103268229}{273922583} = 0.3697$	$\frac{24596013}{109434718} = 0.2248$	68.7822
128	$\frac{106716828}{388757301} = 0.2203$	56.2328	$\frac{87087199}{279322583} = 0.3118$	$\frac{18101718}{109434718} = 0.1654$	55.1577
256	$\frac{86538258}{388757301} = 0.2203$	45.4461	$\frac{71443781}{279322583} = 0.2558$	$\frac{12631324}{109434718} = 0.1154$	44.3371
512	$\frac{64804288}{388757031} = 0.1667$	34.78155	$\frac{59537149}{279322583} = 0.2131$	$\frac{8673082}{109434718} = 0.0793$	36.2120

Simulation 2. L1/L2 size

L1/L1D/L2U	Inst. Miss rate	Data. Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	$\frac{134825214}{279322583} = 0.4827$	$\frac{41153391}{109434718} = 0.3761$	$\frac{46669296}{190471162} = 0.2450$	32.2358
16/16/512	$\frac{119379511}{279322583} = 0.4274$	$\frac{32326592}{109434718} = 0.2954$	$\frac{63698577}{163585371} = 0.3910$	39.0616
32/32/256	$\frac{103268229}{279322583} = 0.3697$	$\frac{24596013}{109434718} = 0.2248$	$\frac{81823479}{137115368} = 0.5967$	46.4157
64/64/128	$\frac{87087199}{388757031} = 0.3118$	$\frac{18101718}{109434718} = 0.1654$	$\frac{92620079}{111916799} = 0.8267$	50.6750
128/128/0	$\frac{71443781}{388757031} = 0.2558$	$\frac{12631324}{109434718} = 0.1154$	1	44.4254

Simulation 3. Associativity

# of Sets	Split Cache Miss rate / AMAT			
	1-way	2-way	4-way	8-way

	Inst	data	AMAT	Inst	Data	AMAT	Inst	Data	AMAT	Inst	Data	AMAT
64	0.3697	0.2248	66.7822	0.2965	0.1320	51.0587	0.2403	0.0778	39.9517	0.1926	0.0456	31.3051
128	0.3118	0.1654	55.1577	0.2456	0.0881	41.3136	0.1949	0.0483	31.8085	0.1378	0.0298	22.5833
256	0.2558	0.1154	44.3371	0.1986	0.0564	32.8174	0.1408	0.0313	23.1205	0.0858	0.0192	14.5582
512	0.2131	0.0793	36.3120	0.1499	0.0362	24.7261	0.0901	0.0200	15.2437	0.0383	0.0122	7.3843
1024	0.1617	0.0524	27.3563	0.0990	0.0228	16.7032	0.0440	0.0126	8.2493	0.0145	0.0062	3.6742
2048	0.1172	0.0338	19.9613	0.0563	0.0140	10.1185	0.0182	0.0064	4.2415	0.0082	0.0021	2.5877

Simulation 4. Block size

Block size	Unified Cache Miss rate	AMAT
16	$\frac{64804288}{388757301} = 0.1667$	34.34
64	$\frac{15952784}{388757301} = 0.0410$	9.2816
128	$\frac{80186814}{388757301} = 0.0206$	5.2449
256	$\frac{4598981}{388757301} = 0.0118$	3.5299
512	$\frac{2503457}{388757301} = 0.0064$	2.4967

cc1 벤치마크의 경우 먼저 simul1의 경우 miss rate는 cache size가 커질수록 줄어들며 unified 512 set의 경우가 가장 성능이 좋다. simul2의 경우 L1(inst,data)의 miss rate는 줄어들지만 L2(unified)의 miss rate는 증가한다. 이 경우 8/8/1024의 경우가 가장 성능이 좋다. 이어서 simul3의 경우 cache size와 associativity가 증가할수록 성능이 좋으므로, 8-way, 2048 set의 경우가 가장 성능이 좋다. 마지막으로 simul4의 경우 block size가 커질수록 miss rate가 줄어드므로, 512의 경우가 가장 성능이 좋다.

따라서 cc1 벤치마크에서 가장 좋은 성능을 보여주는 simul은 Associativity는 1, Number of sets는 512, block size는 512인 simulation 4이다.

<ijpeg>

Simulation 1. unified vs splits

# of Sets	Unified cache Miss rate	Unified Cache AMAT	Split cache		Split cache AMAT
			Inst. Miss rate	Data Miss rate	

64	$\frac{7029845}{37389281} = 0.1880$	38.8179	$\frac{7560571}{29242613} = 0.2585$	$\frac{2094612}{8146668} = 0.2571$	52.6390
128	$\frac{3759613}{37389281} = 0.1006$	21.3866	$\frac{3184598}{29242613} = 0.1089$	$\frac{1787696}{8146668} = 0.2194$	27.6353
256	$\frac{1709393}{37389281} = 0.0457$	10.4573	$\frac{618645}{29242613} = 0.0212$	$\frac{1208584}{8146668} = 0.1484$	10.8647
512	$\frac{1203872}{37389281} = 0.0322$	7.8100	$\frac{137231}{29242613} = 0.0047$	$\frac{644333}{8146668} = 0.0791$	5.3070

Simulation 2. L1/L2 size

L1/L1D/L2U	Inst. Miss rate	Data. Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	$\frac{14362745}{29242613} = 0.4912$	$\frac{3415580}{8146668} = 0.4193$	$\frac{580329}{18700211} = 0.0310$	13.4590
16/16/512	$\frac{10916907}{29242613} = 0.3733$	$\frac{2860633}{8146668} = 0.3511$	$\frac{1046953}{14351867} = 0.0729$	13.4980
32/32/256	$\frac{7560571}{29242613} = 0.2585$	$\frac{2094612}{8146668} = 0.2571$	$\frac{1275461}{10117078} = 0.1261$	12.3676
64/64/128	$\frac{3184598}{29242613} = 0.1089$	$\frac{1787696}{8146668} = 0.2194$	$\frac{2165063}{5308509} = 0.4078$	14.3347
128/128/0	$\frac{618645}{388757031} = 0.0212$	$\frac{1208584}{8146668} = 0.1484$	1	10.9529

Simulation 3. Associativity

# of Sets	Split Cache Miss rate / AMAT											
	1-way			2-way			4-way			8-way		
	Inst	data	AMAT	Inst	Data	AMAT	Inst	Data	AMAT	Inst	Data	AMAT
64	0.2585	0.2751	52.6308	0.1172	0.1812	27.6218	0.0173	0.0474	5.9874	0.0050	0.0251	3.0542
128	0.1089	0.2194	28.2790	0.0170	0.1537	11.2142	0.0055	0.0357	3.6740	0.0012	0.0154	2.0452
256	0.0212	0.1484	11.6057	0.0054	0.0711	5.4290	0.0014	0.0160	2.1266	0.0004	0.0113	1.7663
512	0.0047	0.0791	5.7405	0.0015	0.0517	3.9274	0.0003	0.0114	1.7937	0.0003	0.0090	1.6835
1024	0.0035	0.0518	4.2560	0.0010	0.0126	1.9663	0.0003	0.0092	1.7168	0.0003	0.0084	1.7016
2048	0.0025	0.0157	2.3688	0.0003	0.0097	1.7654	0.0003	0.0085	1.7309	0.0003	0.0081	1.7365

Simulation 4. Block size

Block size	Unified Cache Miss rate	AMAT
16	$\frac{1203872}{37389281} = 0.0322$	7.4400
64	$\frac{348760}{37389281} = 0.0093$	2.9416
128	$\frac{52199}{37389281} = 0.0014$	1.4049
256	$\frac{30037}{37389281} = 0.0008$	1.3299
512	$\frac{20779}{37389281} = 0.0006$	1.3367

ijpeg 벤치마크의 경우 먼저 simul1의 경우 miss rate는 cache size가 커질수록 줄어들며 split 512 set의 경우가 가장 성능이 좋다. simul2의 경우 L1(inst,data)의 miss rate는 줄어들지만 L2(unified)의 miss rate는 증가한다. 이 경우 L2를 사용하지 않는 경우, 즉 128/128/0의 경우가 가장 성능이 좋다. 이어서 simul3의 경우 cache size와 associativity가 증가할수록 성능이 좋지만, 512를 넘어가는 부분부터는 다시 증가하므로 8-way, 512 set의 경우가 가장 성능이 좋다. 마지막으로 simul4의 경우 block size가 커질수록 miss rate가 줄어들지만, 256을 넘어가는 부분부터는 다시 증가하게 되므로 256의 경우가 가장 성능이 좋다.

따라서 ijpeg 벤치마크에서 가장 좋은 성능을 보여주는 simul은 Associativity는 1, Number of sets는 512, block size는 256인 simulation 4이다.

<perl>

Simulation 1. unified vs splits

# of Sets	Unified cache Miss rate	Unified Cache AMAT	Split cache		Split cache AMAT
			Inst. Miss rate	Data Miss rate	
64	$\frac{167548}{451517} = 0.3711$	76.4591	$\frac{132588}{327284} = 0.4051$	$\frac{29711}{124233} = 0.2392$	72.8907
128	$\frac{131070}{451517} = 0.2903$	59.3862	$\frac{115217}{327284} = 0.3520$	$\frac{22455}{124233} = 0.1807$	62.0135
256	$\frac{103964}{451517} = 0.2303$	47.4392	$\frac{91390}{327284} = 0.2792$	$\frac{17246}{124233} = 0.1388$	49.1955

512	$\frac{81960}{451517} = 0.1815$	37.7344	$\frac{69811}{327284} = 0.2133$	$\frac{13435}{124233} = 0.1081$	37.9958

Simulation 2. L1/L2 size

L1/L1D/L2U	Inst. Miss rate	Data. Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	$\frac{149859}{327284} = 0.4579$	$\frac{46565}{124233} = 0.3748$	$\frac{68385}{219372} = 0.3072$	36.4293
16/16/512	$\frac{142095}{327284} = 0.4342$	$\frac{36813}{124233} = 0.2963$	$\frac{82012}{197676} = 0.4149$	41.5418
32/32/256	$\frac{132588}{327284} = 0.4051$	$\frac{29711}{124233} = 0.2392$	$\frac{102215}{177493} = 0.5759$	49.1301
64/64/128	$\frac{115217}{327284} = 0.3520$	$\frac{22455}{124233} = 0.1807$	$\frac{121070}{149329} = 0.8018$	55.9827
128/128/0	$\frac{91390}{327284} = 0.2792$	$\frac{17246}{124233} = 0.1388$	1	49.2838

Simulation 3. Associativity

# of Sets	Split Cache Miss rate / AMAT											
	1-way			2-way			4-way			8-way		
	Inst	data	AMAT	Inst	Data	AMAT	Inst	Data	AMAT	Inst	Data	AMAT
64	0.4051	0.2391	74.1514	0.3520	0.1507	61.8781	0.2676	0.1039	46.8008	0.1978	0.0693	34.5301
128	0.3520	0.1807	63.3202	0.2806	0.1108	49.1473	0.1998	0.0728	35.0221	0.1250	0.0461	22.3537
256	0.2792	0.1388	50.2665	0.2028	0.0821	35.9419	0.1297	0.0514	23.3538	0.0670	0.0360	13.0784
512	0.2133	0.1081	38.7982	0.1404	0.0594	25.3879	0.0728	0.0377	14.0618	0.0342	0.0301	7.8394
1024	0.1684	0.0717	30.2662	0.0871	0.0400	16.3807	0.0372	0.0304	8.3348	0.0214	0.0288	5.8722
2048	0.1418	0.0590	25.6518	0.0615	0.0333	12.2043	0.0240	0.0290	6.3028	0.0184	0.0288	5.4641

Simulation 4. Block size

Block size	Unified Cache Miss rate	AMAT
16	$\frac{81960}{451517} = 0.1815$	37.3000

64	$\frac{22243}{451517} = 0.0493$	10.0416
128	$\frac{11489}{451517} = 0.0254$	6.2049
256	$\frac{6914}{451517} = 0.0153$	4.2299
512	$\frac{3238}{451517} = 0.0072$	2.6567

perl 벤치마크의 경우 먼저 simul1의 경우 miss rate는 cache size가 커질수록 줄어들며 unified 512 set의 경우가 가장 성능이 좋다. simul2의 경우 L1(inst,data)의 miss rate는 줄어들지만 L2(unified)의 miss rate는 증가한다. 이 경우 8/8/1024의 경우가 가장 성능이 좋다. 이어서 simul3의 경우 cache size와 associativity가 증가할수록 성능이 좋으므로, 8-way, 2048 set의 경우가 가장 성능이 좋다. 마지막으로 simul4의 경우 block size가 커질수록 miss rate가 줄어드므로, 512의 경우가 가장 성능이 좋다.

따라서 perl 벤치마크에서 가장 좋은 성능을 보여주는 simul은 Associativity는 1, Number of sets는 512, block size는 512인 simulation 4이다.

정렬 프로그램은 데이터에 대한 반복적인 액세스를 수행하기 때문에 이러한 특성으로 인해 정렬 프로그램과 유사한 메모리 액세스 패턴을 갖는 벤치마크인 ijpeg의 결과를 참고하여 적합한 cache 구성을 찾을 수 있다고 생각한다. 따라서 ijpeg 벤치마크에서 가장 우수한 성능을 보여준 cache 구성을 기반으로 정렬 프로그램에 적합한 cache를 선택하면, block size가 512 이고 number of set이 512, associativity가 1인 unified cache인 simulation4가 가장 좋은 성능을 보이며 적합하다고 생각한다.

이렇게 벤치마크 프로그램과 정렬 프로그램 각각에 적절한 cache를 찾아보았는데, 각각의 프로그램에 적합한 cache가 다른 이유는 프로그램의 메모리 액세스 패턴과 특성에 따라 메모리 액세스 방식과 경향이 다르기 때문이다. 과제에서 수행한 3개의 벤치마크를 예로 들어보겠다. cc1 컴파일러는 명령어를 가져오는 액세스가 많을 것이며, 컴파일러가 소스코드를 분석하고 변환하여 명령어를 생성하는 작업을 수행하기 때문에 명령어에 대한 액세스가 빈번하게 발생할 것이며 명령어 cache 히트율을 높이기 위해 적절한 캐시 구성이 필요할 것이다. ijpeg 프로그램은 이미지 데이터에 대한 액세스가 많이 발생할 것이다. 이미지 데이터의 일부를 반복적으로 액세스하고 작업을 수행하는 경우가 많을 것이므로 데이터 cache의 효율적인 사용과 데이터 액세스 패턴을 잘 지원하는 cache 구성이 필요할 것이다. 마지막으로 perl 인터프리터는 명령어와 데이터에 대한 액세스가 모두 중요한 역할을 할 것이다. 다양한 작업을 수행하기 때문에 명령어와 데이터를 모두 적절하게 액세스 할 수 있는 캐시 구성이 필요할 것이다. 이처럼 각각의 프로그램은 메모리 액세스 방식과 경향이 다르기 때문에 적합한 cache 구성이 다르게 나타나므로 각각에 맞는 cache 구성을 선택하고 최적화 해야한다.

3. Consideration

이번 프로젝트에서는 cache access 방식과 cc1, ijpeg, perl 3 가지 벤치마크에 따른 cache access 성능을 비교하였다. Insertion sort 와 Random access 프로그램을 수행했을 때의 instruction cache 와 data cache 의 access 방식을 wave form 을 통해 확인해보았으며, hit 과 miss 의 동작 방식 그리고 동작 측면에서의 차이점 등에 대해 알아보았다. 이 실험을 통해 random access 는 random 으로 값을 가져오며, 또한 insertion sort 는 반복문을 통해 이전에 수행했던 것을 다시 가져오기 때문에 random access 가 insertion sort 보다 miss 발생율이 높다는 것을 알게 되었다. 또한 waveform 에서 동작방식을 하나하나 확인하며 insertion sort 와 random access 방식의 동작방식이 거의 유사하다는 것을 알았다. 벤치마크에 따른 cache access 성능을 비교하면서 miss rate 를 구하는 것은 크게 어렵지 않았으나, AMAT 를 구하는데 있어 큰 어려움이 있었다. AMAT 의 크기가 커지거나 작아진다, 이렇게 두 가지 결과만 나올 것이라고 예상하였지만, 커지다가 작아지거나, 작아지다가 다시 커지는 경우도 종종 있었다. cache size 를 증가시킨다고 해서 성능이 좋아진다는 것이 아니라는 것을 알 수 있었다. 또한 3 개의 벤치마크 모두 block size simul 에서 가장 성능이 좋다는 결과가 나왔다. set 을 증가시켜가며 성능을 비교하였는데, set 의 최대값은 512 에서 512 보다 조금 높은 수준인데 성능은 512 일 때가 가장 좋다. 즉 set 을 증가시키더라도 512 이상으로 성능이 좋아지지는 않았다. block size 는 set 디폴트 값을 다른 시뮬레이션에서 제일 좋은 수치를 주고 블록 사이즈도 올려주기 때문에 성능이 최적화될 수밖에 없다고 생각한다.

4. Reference

컴퓨터구조실험 강의자료

컴퓨터구조 강의자료