Project on

Cache Performance Analysis with Write Policy using SimpleScalar3.0

A Report Submitted By

Sandeep
25CS06018

Sujeet Kumar Yadav
25CS06022

Under the Supervision of

Dr. Devashree Tripathy
Assistant Professor

Department of Computer Science and Engineering
School of Electrical and Computer Sciences (SECS)
IIT Bhubaneswar

Introduction

In modern computer systems, cache memory plays a vital role in bridging the speed gap between the processor and main memory. Caching allows frequently accessed data to be stored closer to the CPU, reducing access latency and improving system performance.
However, the performance of a cache depends on several factors, including size, associativity, replacement policy, and write policy.

This project focuses on the analysis of cache performance with different write policies — specifically Write-Through and Write-Back — using the SimpleScalar 3.0 simulator.

Objective

The main objectives of this project are:

- To understand the impact of different write policies on cache performance.
- To simulate cache operations using the SimpleScalar toolset.
- To analyze performance parameters such as miss rate, hit rate, and execution time.
- To compare Write-Back and Write-Through policies based on modified test case code c.

Tools and Environment

- **Simulator Used:** SimpleScalar 3.0
- **Programming Language:** C
- **Platform:** Linux / Windows (with GCC or WSL)
- **Configuration Files Modified:** cache.h, sim-cache.c,testcaselru ,llong

- **Commands Used:**
-

make clean

make

./sim-cache -cache:il1 il1:64:32:1:l -cache:dl1 dl1:128:32:2:l -cache:dl2 dl2:512:64:4:l benchmarks/somefile.ss

Write Policies Overview

1 Write-Through Policy

- Every write operation updates both the **cache** and the **main memory**.
- Ensures data consistency between cache and memory.
- Increases **memory traffic** and **write latency**.
- Simpler to implement.

2 Write-Back Policy

- Data is written only to the cache at first.
- The modified data is written to main memory **only when the cache block is replaced**.
- Reduces **memory write traffic**.
- Requires a **dirty bit** to track modified blocks.
- Offers better performance but more complex implementation.

Performance Analysis

- **Write-Through:** Ensures immediate consistency but introduces more write operations to main memory, increasing latency.
- **Write-Back:** Reduces write operations and improves speed, though at the cost of increased control complexity.

- **Result:** Write-Back outperforms Write-Through in most CPU cache scenarios, especially where write frequency is high.

Conclusion

From the analysis, it is concluded that:

- **Write-Back policy** provides better performance in terms of execution speed and memory efficiency.
- **Write-Through policy** ensures better data consistency but at the cost of higher latency.
- The choice of write policy depends on the application requirements — **performance-oriented** systems prefer *Write-Back*, while **data-sensitive** systems may prefer *Write-Through*.

Project on

Cache Performance Analysis with write policy using SimpleScalar

Project Member Name:-
Sujeet Kumar Yadav (25CS06022)
Sandeep (25CS06018)

M.tech(CSE)

Methods

This project examines Cache analysis and Performance on Different Policies with replacement write policy using the SimpleScalar sim-cache tool.
Performance is measured based on:

- Hit / Miss percentages
- Replacement Count (DL1 / IL1)
- Write-Back Count (DL1 / IL1)
  (Note: IL1 is read-only → IL1 Write-Back = 0 always)

Set environment variables as follows:

export IDIR=<your-simplesclar-installation-dir>
export HOST=i686-pc-linux
export TARGET=sslittle-na-sstrix
cd $IDIR

Compile test-case command -:

./bin/sslittle-na-sstrix-gcc -o test-case(compile test case file name) test.c(code file name)

for checking write_back policy(default) we modify the cache.h file line no.22 same for

for checking write_through policy below code is there

#define WRITE_POLICY WRITE_BACK /* change to WRITE_THROUGH for comparison

- Make clean

- make

./sim-cache -cache :dl1 dl1:<cache size>:<block-size>:<associativity>:<replacement policy> -cache :il1 il1:<cache size>:<block-size>:<associativity>:<replacement policy> <address of test-case>

Analysis Tables are below:-

Table1

Test case :test-llong using associativity 2

Replacement Policy	Write policy	Block size	Associativity	Data level_intst		Instruction level		No of replacement		Write_back	
				hit%	miss%	hit%	miss %	dl1	il1	dl1	il1
LRU	Write through	32	2	0.9514	0.0486	0.8528	0.1472	479	4378	5461	0
		64	2	0.9743	0.0257	0.9432	0.0568	238	1669	5461	0
	Write Back	32	2	0.9514	0.0486	0.8528	0.1472	479	4378	436	0
		64	2	0.9743	0.0257	0.9432	0.0568	238	1669	213	0
FIFO	Write through	32	2	0.9507	0.0493	0.8526	0.1474	486	4384	5461	0
		64	2	0.9745	0.0255	0.9432	0.0568	236	1670	5461	0
	Write Back	32	2	0.9507	0.0493	0.8526	0.1474	486	4384	444	0
		64	2	0.9745	0.0255	0.9432	0.0568	236	1670	212	0
RANDOM	Write through	32	2	0.9494	0.0506	0.8649	0.1351	500	4018	5461	0
		64	2	0.9727	0.0273	0.9450	0.0550	255	1615	5461	0
	Write Back	32	2	0.9494	0.0506	0.8649	0.1351	500	4018	448	0
		64	2	0.9727	0.0273	0.9450	0.0550	255	1615	221	0

Table 2

Test case :test-llong using associativity 4

Replacement Policy	Write policy	Block size	Associativity	Data level_intst		Instruction level		No of replacement		Write_back	
				hit%	miss%	hit%	miss %	dl1	il1	dl1	il1
LRU	Write through	32	4	0.9554	0.0446	0.9357	0.0643	405	1863	5461	0
		64	4	0.9766	0.0234	0.9640	0.0360	182	1015	5461	0
	Write Back	32	4	0.9554	0.0446	0.9357	0.0643	405	1863	393	0
		64	4	0.9766	0.0234	0.9640	0.0360	182	1015	176	0
FIFO	Write through	32	4	0.9554	0.0446	0.9354	0.0646	405	1872	5461	0
		64	4	0.9765	0.0235	0.9634	0.0366	183	1032	5461	0
	Write Back	32	4	0.9554	0.0446	0.9354	0.0646	405	1872	393	0
		64	4	0.9765	0.0235	0.9634	0.0366	183	1032	177	0
RANDOM	Write through	32	4	0.9536	0.0464	0.9305	0.0695	424	2020	5461	0
		64	4	0.9761	0.0239	0.9644	0.0356	188	1002	5461	0
	Write Back	32	4	0.9536	0.0464	0.9305	0.0695	424	2020	403	0
		64	4	0.9761	0.0239	0.9644	0.0356	188	1002	177	0

Table3

Test case :test-caselru using associativity 2

Replacement Policy	Write policy	Block size	Associativity	Data level_intst		Instruction level		No of replacement		Write_back	
				hit%	miss%	hit%	miss %	dl1	il1	dl1	il1
LRU	Write through	32	2	0.8945	0.1055	0.8876	0.1124	471	1043	3676	0
		64	2	0.9442	0.0558	0.9372	0.0628	234	569	3676	0
	Write Back	32	2	0.8945	0.1055	0.8876	0.1124	471	1043	437	0
		64	2	0.9442	0.0558	0.9372	0.0628	234	569	215	0
FIFO	Write through	32	2	0.8947	0.1053	0.8874	0.1126	470	1045	3676	0
		64	2	0.9440	0.0560	0.9373	0.0627	235	568	3676	0
	Write Back	32	2	0.8947	0.1053	0.8874	0.1126	470	1045	437	0
		64	2	0.9440	0.0560	0.9373	0.0627	235	568	217	0
RANDOM	Write through	32	2	0.8918	0.1082	0.8876	0.1124	484	1043	3676	0
		64	2	0.9425	0.0575	0.9369	0.0631	242	572	3676	0
	Write Back	32	2	0.8918	0.1082	0.8876	0.1124	484	1043	444	0
		64	2	0.9425	0.0575	0.9369	0.0631	242	572	223	0

Table4

Test case :test-caselru using associativity 4

Replacement Policy	Write policy	Block size	Associativity	Data level_intst		Instruction level		No of replacement		Write_back	
				hit%	miss%	hit%	miss %	dl1	il1	dl1	il1
LRU	Write through	32	4	0.8985	0.1015	0.8994	0.1006	420	898	3676	0
		64	4	0.9469	0.0531	0.9473	0.0527	189	440	3676	0
	Write Back	32	4	0.8985	0.1015	0.8994	0.1006	420	898	403	0
		64	4	0.9469	0.0531	0.9473	0.0527	189	440	182	0
FIFO	Write through	32	4	0.8987	0.1013	0.8992	0.1008	419	900	3676	0
		64	4	0.9467	0.0533	0.9471	0.0529	190	442	3676	0
	Write Back	32	4	0.8987	0.1013	0.8992	0.1008	419	900	404	0
		64	4	0.9467	0.0533	0.9471	0.0529	190	442	183	0
RANDOM	Write through	32	4	0.8943	0.1057	0.9006	0.0994	440	887	3676	0
		64	4	0.9455	0.0545	0.9459	0.0541	197	453	3676	0
	Write Back	32	4	0.8943	0.1057	0.9006	0.0994	440	887	414	0
		64	4	0.9455	0.0545	0.9459	0.0541	197	453	187	0

Final Conclusion

Best Configuration

LRU + Write-Back + 64B Block + 4-Way Associativity

✓ Highest Hit Rate
✓ Lowest Replacement
✓ Lowest Memory Traffic