A Simple Multi-Core Functional Cache Design Simulator

Rano Mal
The University of Texas Rio Grande Valley

A SIMPLE MULTI-CORE FUNCTIONAL CACHE DESIGN SIMULATOR

A Thesis

by

RANO MAL

Submitted to the Graduate College of
The University of Texas Rio Grande Valley
In partial fulfillment of the requirement of the degree of

MASTER OF SCIENCE ENGIEERING

July 2017

Major Subject: Electrical Engineering

A SIMPLE MULTI-CORE FUNCTIONAL CACHE DESIGN SIMULATOR

A Thesis
by
RANO MAL

COMMITTEE MEMBERS

Dr. Yul Chu
Chair of Committee

Dr. Sanjeev Kumar
Committee Member

Dr. John Abraham
Committee Member

July 2017

ABSTRACT

Mal, Rano, A Simple Multi-Core Functional Cache Design Simulator. Master of Science Engineering (MSE), July, 2017, 130 pp, 2 tables, 123 figures, 24 references.

This thesis presents a flexible multi-core cache memory simulator to design and evaluate memory hierarchies for modern general-purpose or embedded processors. The proposed simulator needs to work with Pin, which is an open-source dynamic instrumentation tool provided by Intel. The Pin intercepts the execution of instructions and generates a sequence code (traces) to feed into the simulator for any selected benchmark programs, such as SPEC2006, SPLASH2, or PARSEC. We have a plan to release this simulator as an open-source (like Pin) to support research and/or academic community for their simulation works. In addition, we expect more functions can be updated on top of this simulator to share by the research community.

DEDICATION

The completion of my master studies would not have been possible without the blessing and support of my family. I would like to thank and dedicate my work to my parents, Mr. & Mrs. Sawai Mal, for their love and exception support. This thesis would be incomplete without the acknowledgment to one person who has been there for me all time, one and only my dearest brother Ashok Kumar.

ACKNOWLEDGMENTS

I will always be grateful to Dr. Yul Chu, chair of my thesis committee, for his constant encouragement and support throughout my graduate program. He has always taken out time to answer my questions and discuss new research ideas. His ability to quickly identify the weakest part of my research rationale has been an important source of guidance. I greatly appreciate my committee member, Dr. Sanjeev Kumar and Dr. John Abraham for taking the time to serve on my committee and to open my mind to new areas of electrical engineering through their lectures.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In general, cache memories have been considered as a major functional unit to improve system performance for multi-core processors. Therefore, for a processor architect, cache memory can be a key component to design in enhancing the performance of current multi-core processors [1]. After designing a cache memory, it is necessary to evaluate the cache through simulation before moving to next step, such as chip fabrication.

The goal of simulation is to evaluate detailed characteristics for the cache, such as cache misses, average memory access time, cache coherence, bus traffic, power consumption, cost, etc. There exist many open-source cache simulators and other useful software tools (such as Pin [2]) to measure the cache memory performance [3, 4]. Such memory system simulators can be categorized into either trace-driven or execution-driven [5] as follows: 1) trace-driven simulators (e.g., Dinero IV [6]) are used to read a trace file of pre-collected addresses of instructions. Therefore, they do not need to implement any functional units; and 2) execution-driven simulators (e.g. SimpleScalar [7]) need to run a benchmark program (or real application program) to generate instruction streams dynamically using all functional units. Therefore, it is very difficult to design and implement the execution-driven simulator precisely compared to trace-driven one since all execution should be executed accurately through functional units during runtime. Between the two types of simulators, trace-driven simulators look more popular for academic purpose since they are easy and fast to implement a new cache scheme. However, the result might not be accurate compared to execution-driven simulators since there are no feedbacks, such as any update, to traces [7].

Even though there are many open-source simulators, it is difficult to find appropriate ones to simulate any newly designed multi-core cache schemes because of limited parallel benchmark programs and software architectures. In other words, SPEC benchmark programs [8] have been popular to evaluate CPUs and cache memories in computer system community using the SimpleScalar simulator [7]. However, it has been difficult to simulate multi-core cache schemes using SimpleScalar and SPEC benchmark programs. Therefore, there have been strong demands to develop a flexible and expandable multi-core cache memory simulator, which can be run for any application or benchmark programs as an open-source.

To meet those demands, this thesis proposes an open-source functional multi-core cache simulator, which can design and simulate newly-designed cache memories for multi-core processors. This proposed simulator can be claimed as a trace-driven simulator since it needs to read a trace of instructions with memory addresses and functions such as load, store, and other instructions. The instruction and data traces will be collected by executing real application programs (or benchmark programs) through Pin Tool [2, 9].

CHAPTER II

RELATED WORK

Cache memory is just a buffer to hold instructions or data, and we discussed, in the previous section, that cache has been considered as a critical unit in improving the performance for modern computer systems with the most cost-effective way [1]. That is why multi-core shared cache memories have been one of popular research topics in computer research communities. In addition, some open-source cache memory simulators (such as SimpleScalar [7], Multi2Sim [10], and SMPCache [1]) have been used to evaluate performance for various multi-level and/or multi-core cache memories by those communities. According to [7], SimpleScalar is an open-source tool to perform accurate simulation for a modern processor, such as MIPS architecture [11]. It is a virtual computer system which can simulate various modern processors and memory hierarchies with benchmark programs, such as SPEC2006 [8].

The SimpleScalar simulator is efficient to evaluate superscalar architecture along with memory hierarchies, and uses a pedagogically appropriate instruction set, which is called PISA (Portable Instruction Set Architecture) derived from the MIPS architecture. However, it is difficult to simulate a current multiprocessor (or multi-core) architecture [12]. That is why it has not been used to evaluate various multi-core processors directly. Even though there have been various SimpleScalar extensions to support multithreading architecture, such as SSMT [13], M-Sim [14], or SMTSim [15], there are many restrictions in implementing sequential workloads and sharing resources among threads.

Multi2Sim, according to [16], is a simulator to implement modern CPUs and GPUs (Graphics Processing Units). In other words, it supports multicore and/or multithread processor simulation along with benchmark programs, such as PARSEC or SPLASH2x [19]. Multi2Sim is an open-source simulator, which is allowed to modify and port any functional units into this simulator. However, it has been generally agreed that porting any functional units, such as L1 cache memory, to the Multi2Sim is a bit more difficult than SimpleScalar simulator.

The SimpleScalar (including extensions such as SSMT) and Multi2Sim can be grouped as an execution-driven simulator. Meanwhile, SMPCache is categorized as a trace-driven simulator, which is used for the performance analysis and education purpose of cache memories only for symmetric multiprocessors [1]. Therefore, this simulator has some drawbacks as follows [1]: 1) Restricted portability problems because of UNIX simulator; 2) Limited interface to provide input parameters; and 3) Limited analysis based on fixed behaviors, such as replacement policies, program localities, etc. Beside the Multi2Sim, there have been several multicore simulators, such as Turandot simulator [17], which implements a parallel PowerPC microarchitecture. However, this simulator is not an open-source (not publicly available). Therefore, the goal of this thesis is to provide an open-source, simple, extensible, and flexible multi-core functional cache simulator which can be used for research and/or academic purpose for both single-thread (or single core) and multi-thread (or multi-core) architectures.

CHAPTER III

A FLEXABLE MULTI-CORE CACHE SIMULATOR (FM-SIM)

The goal of the proposed simulator, FM-SIM is to provide a flexible and extendable multi-core cache memory design and simulation for research and/or academic purpose. The major operational flow for the proposed simulator is shown in Figure 3.1.

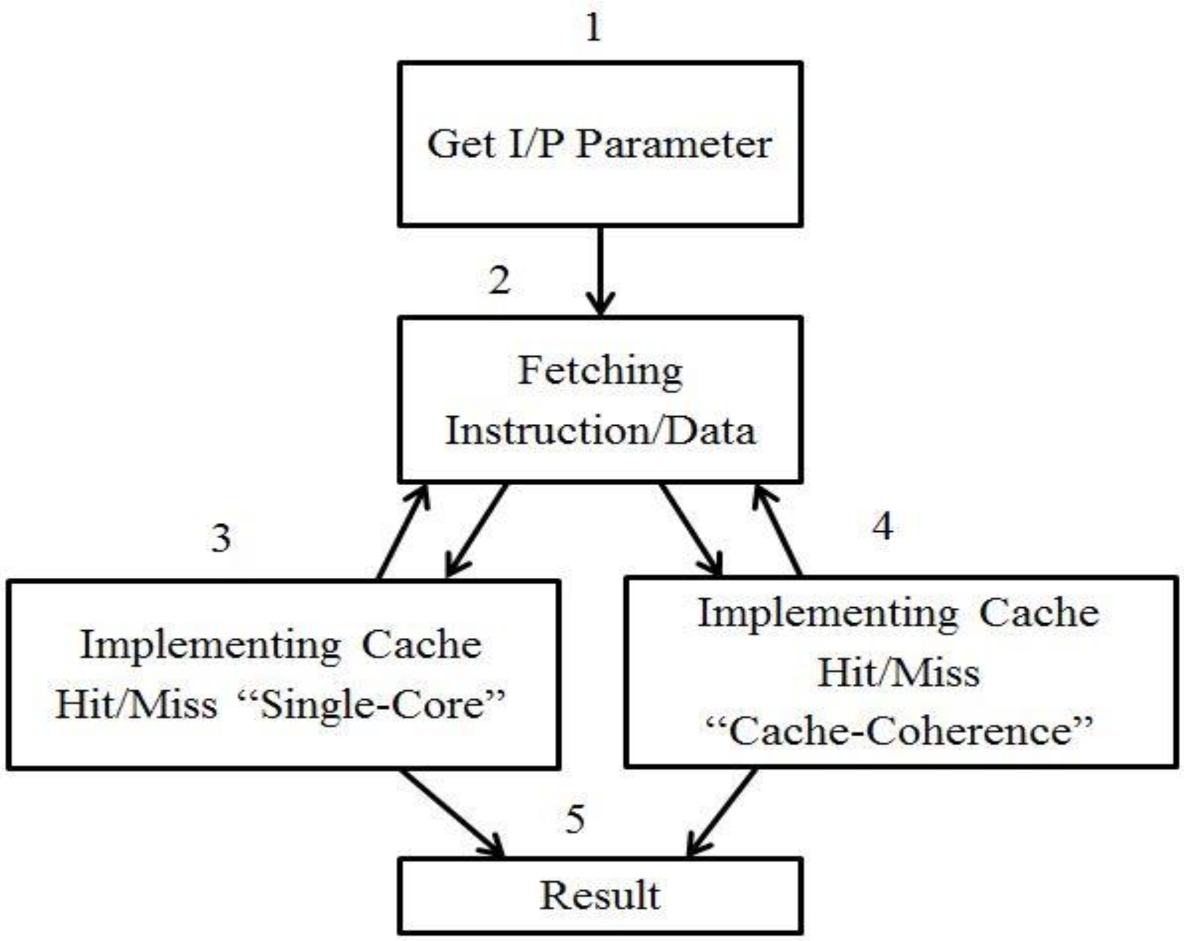

1
Get I/P Parameter
2
Fetching Instruction/Data
3
Implementing Cache Hit/Miss "Single-Core"
4
Implementing Cache Hit/Miss "Cache-Coherence"
5
Result

Figure 3.1: Major operational flow for the proposed cache simulator.

Figure 3.1 consists of 5 stages of operational flow as follows: 1) Getting input parameters from the command line; 2) Fetching instruction/data (machine code) from a trace file; 3) Implementing cache hits /miss for ‘Single-Core’; 4) Implementing cache hits /miss based on ‘Cache-Coherence’; and 5) Providing output results.

3.1 Getting Input Parameters

In this stage, the simulator reads the cache memory parameters to simulate from the command line. The parameters are related to cache size, block size, set-associativity, write policy, number of cores, coherence protocol, etc. Figure 3.2 shows the format of the command-line as an example. In Figure 3.2, FM-SIM is the name of simulator; L1 means Level 1 cache, and L2 as Level 2 cache. ‘13’ means the size of cache memory, which means 2^{13} bytes or 8 Kbytes.

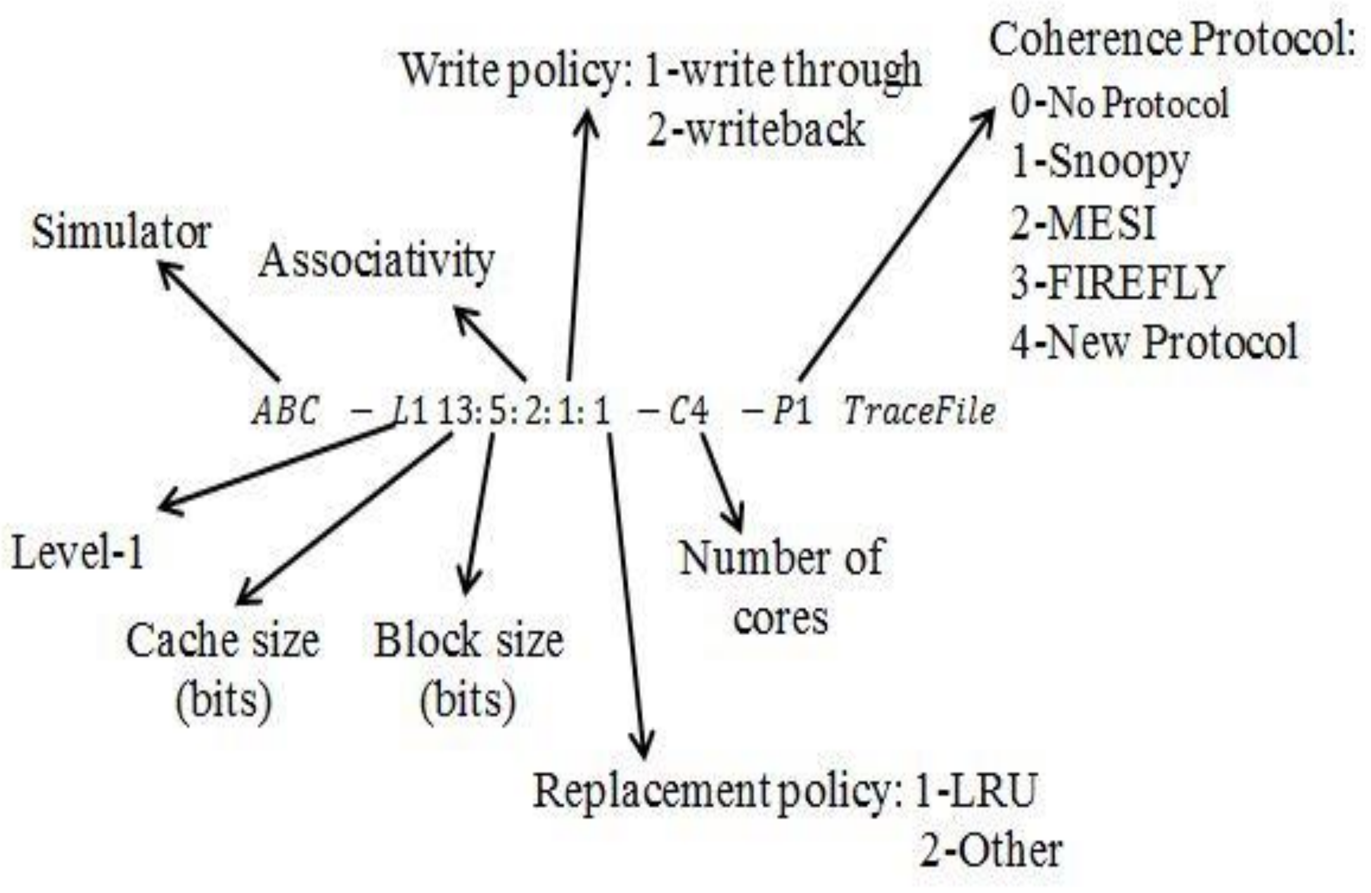


Figure 3.2: A Command-line for the proposed simulator, FM-SIM.

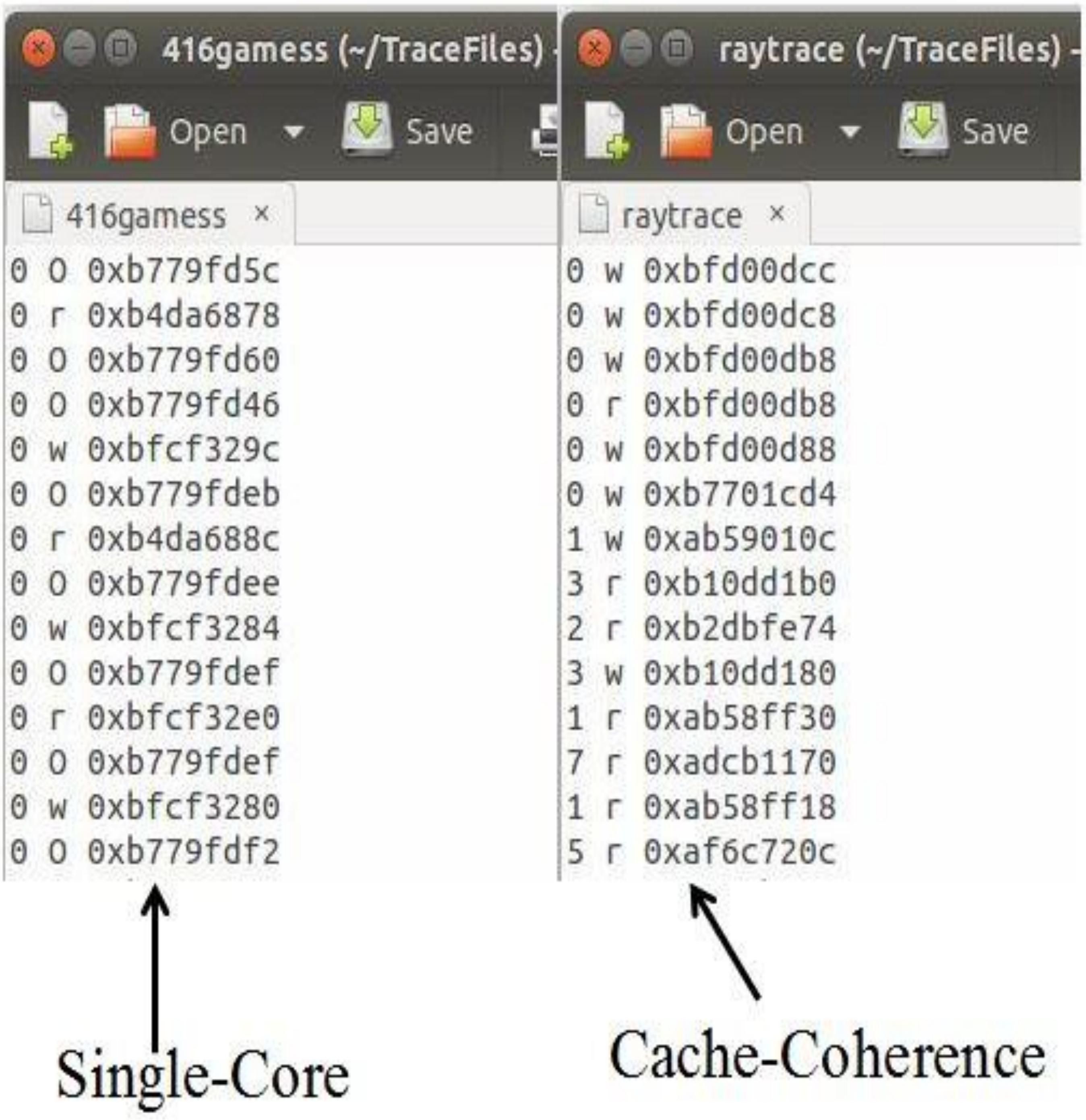
In Figure 3.2, after the colon, '5' shows the block size of a cache line, which means 2^5 bytes or 32 bytes of a cache line block size. '2' followed by a colon is the number of set-associativity, a 2-way set-associative cache memory: For example, if the number is '1', then it means 1-way (or direct-map) set-associative. If the number is '8', then 8-way set-associative cache. In addition, if the associativity is 'S', then 2-way skewed cache. '1' after the associativity shows write policy of write through; if the number is '2', then it means write back policy. For this simulator, the LRU (least recently used) replacement policy is used as default. Of course, other replacement policy can be added to this simulator for a design porting purpose. ' –C4' represents the number of cores and '*-Px*' works for coherence protocol type, such as '*P0-No Protocol, P1-Snoopy, P2-MESI, P3-Firefly and P4-Any new developed Protocol*.' In addition, Trace File is an output text file from Pin Tool (refer to section 4.2). In this thesis, we will provide experimental results in Chapter 5 using the MESI and FIREFLY coherence protocol (refer to Section 3.6.2).

From the command-line in Figure 3.2, the proposed simulator, FM-SIM, will compute other parameters for simulation, such as tag and index bits, since memory address (or memory reference) is divided into tag, index, and byte offset bits: For example, from the command line, if the cache size is 8Kbytes and block size is 32Bytes with 2-way set-associative, then the bit sizes for tag, index and byte-offset can be computed as follows using the equations 1) and 2): byte offset (5 bits based on the block size, 2^5 bytes), index (7 bits), and tag (20 bits for a 32-bit memory address). Followings are the formula to compute tag and index bits for a 32-bit memory address:

$$2^{index\ bit} = m;\ m = \frac{cache\ size}{\text{Set Associativity X Block Size}} \qquad \text{----} \qquad 3.1)$$

$$tag\ bit = 32 - index\ bit - block\ bit \qquad \text{----} \qquad 3.2)$$

3.2 Fetching Instruction/Data (Machine Code)



```
0 O 0xb779fd5c
0 r 0xb4da6878
0 O 0xb779fd60
0 O 0xb779fd46
0 w 0xbfcf329c
0 O 0xb779fdeb
0 r 0xb4da688c
0 O 0xb779fdee
0 w 0xbfcf3284
0 O 0xb779fdef
0 r 0xbfcf32e0
0 O 0xb779fdef
0 w 0xbfcf3280
0 O 0xb779fdf2
```

```
0 w 0xbfd00dcc
0 w 0xbfd00dc8
0 w 0xbfd00db8
0 r 0xbfd00db8
0 w 0xbfd00d88
0 w 0xb7701cd4
1 w 0xab59010c
3 r 0xb10dd1b0
2 r 0xb2dbfe74
3 w 0xb10dd180
1 r 0xab58ff30
7 r 0xadcb1170
1 r 0xab58ff18
5 r 0xaf6c720c
```

Figure 3.3: A Trace file generated by Pin-tool.

In this stage, the simulator accesses a trace file and fetch a line at a time, which was already generated by the Pin-tool (version 3.0). In other words, the proposed simulator uses a trace file as an input, which includes # of thread or core, all instructions along with load & store memory addresses in sequence like the Figure 3.3. Figure 3.3 shows an example of a trace file: The first column represents a # of thread or core, such as ‘0’ for single core and other #s for cache coherence (i.e., multi-core). The 2nd column shows the type of address, such as ‘r’ for load memory address (reading), ‘w’ for store memory address (writing), and ‘O’ for other instructions. In addition, the 3rd column shows memory addresses to access for each instruction.

3.3 Implementing Cache Hit/Miss For Single-Core

As discussed, a fetched command line for single-core consists of ‘0 (core 0)’, address types (r, w, or O) and memory addresses. For the proposed cache simulator, part of a fetched memory address (tag) should be placed or updated into cache memories, such as L1 and L2 cache memories. In order to update cache memories, it is necessary to check cache hit/miss before any update according to the following four cases: 1) cache hit for instructions except Store; 2) cache hit for Store instruction; 3) cache miss for instructions except Store; and 4) cache miss for Store instructions. Figure 3.4 shows the four cases along with a replacement policy (e.g., Least Recently Used or LRU) and a write policy (e.g., write through or write back) in detail.

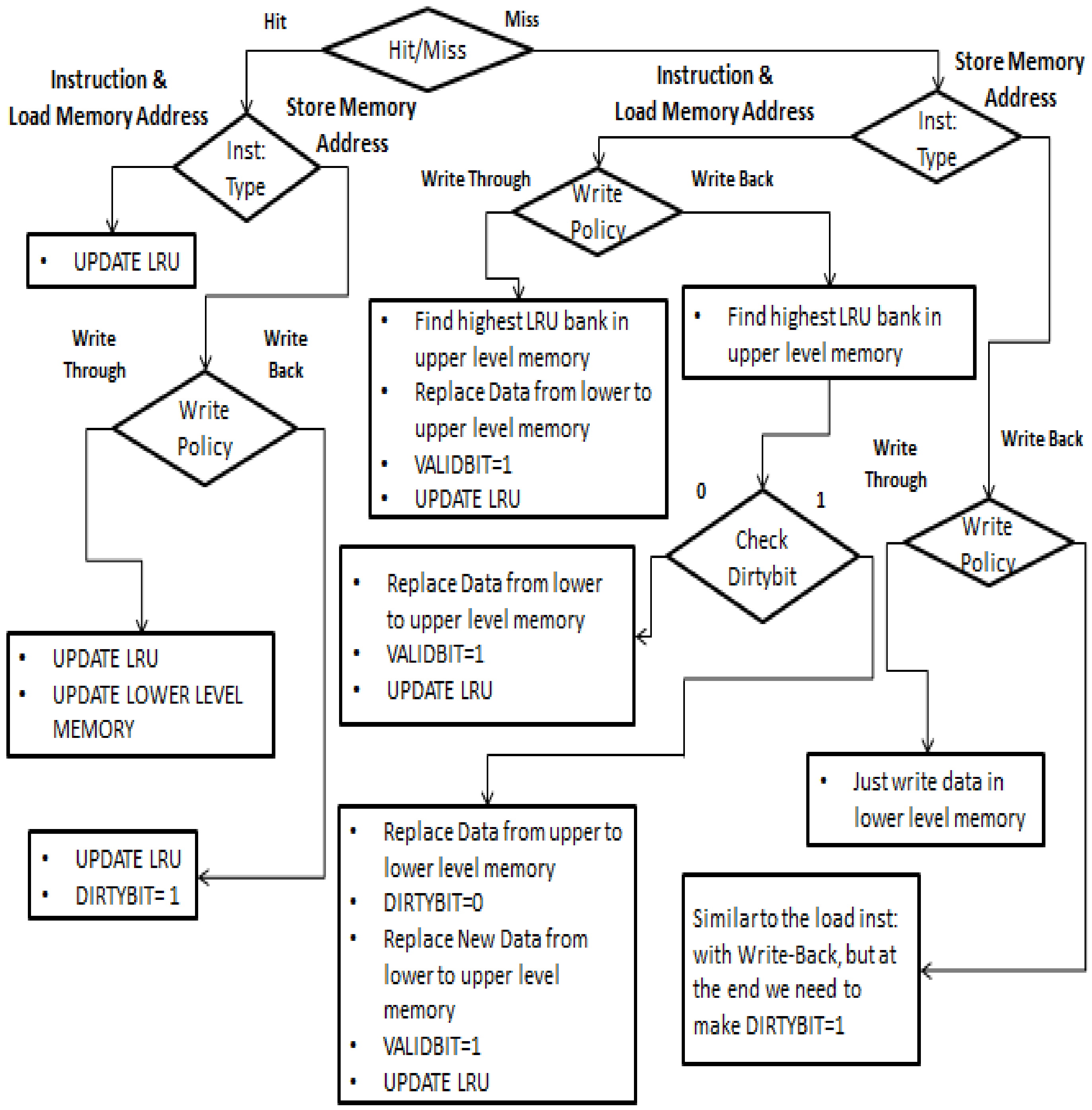

Hit/Miss
Hit
Miss
Instruction & Load Memory Address
Store Memory Address
Inst: Type
UPDATE LRU
Write Through
Write Back
Write Policy
UPDATE LRU
UPDATE LOWER LEVEL MEMORY
UPDATE LRU
DIRTYBIT= 1
Instruction & Load Memory Address
Store Memory Address
Inst: Type
Write Through
Write Back
Write Policy
Find highest LRU bank in upper level memory
Replace Data from lower to upper level memory
VALIDBIT=1
UPDATE LRU
Find highest LRU bank in upper level memory
Check Dirtybit
0
1
Replace Data from lower to upper level memory
VALIDBIT=1
UPDATE LRU
Replace Data from upper to lower level memory
DIRTYBIT=0
Replace New Data from lower to upper level memory
VALIDBIT=1
UPDATE LRU
Write Through
Write Back
Write Policy
Just write data in lower level memory
Similar to the load inst: with Write-Back, but at the end we need to make DIRTYBIT=1

Figure 3.4: Four cases for implementing cache hit/miss.

3.3.1 Cache Hit for Instructions & Load Memory Address

The leftmost part in Figure 3.4 shows the case for the cache hit for instructions except store. In this case, the proposed simulator will process only one step, which is for a replacement policy, such as ‘UPDATE LRU.’ The ‘UPDATE LRU’ is to update the LRU replacement control bits for a cache line. The pseudo code for ‘UPDATE LRU’ is as follows: 1) Check if a fetched address is in cache; 2) If yes (cache hit), set the LRU control bits to zero to indicate it is used most recently. Then, increment other control bits which are less than the current control bits; 3) If not present in cache (cache miss), then check if cache is full; 4) If full, replace the line of the highest control bits with current one and set the control bits to zero. Then, increment other control bits; and 5) If not full, add it to end of cache and set the control bits to zero and increment other control bits. For example, 2-way LRU replacement policy has a 1-bit control bit, which ‘0’ means the least replacement and ‘1’ means the newly replaced one. It is possible to add more replacement policies according to your design.

3.3.2 Cache Hit for Store Memory Address

In this case, the proposed simulator checks write policy. The operation for write policy is as follows: 1) Check the write policy value in the command line, ‘0’ mean write-through and ‘1’ mean write-back; 2) If write policy is write-through then the simulator performs two operations, such as ‘UPDATE LRU (refer to section 3.3.1)’ and ‘UPDATE LOWER LEVEL MEMORY.’ The pseudo code for ‘UPDATE LOWER LEVEL MEMORY’ is as follows: the updated data in the memory address replaces old data at the lower level memory; 3) If write policy is write-back, then the simulator performs two steps, such as ‘UPDATE LRU’ and ‘DIRTYBIT =1.’ The operation for ‘DIRTYBIT =1’ is as follows: whenever data is written into the cache, there is a change in dirty bit in the cache line. It should be set to ‘1’ if the data in the cache is different from the data in the lower level memory.

3.3.3 Cache Miss for Instructions & Load Memory Address

In this case, the proposed simulator checks write policy. If the write policy is write-through, then the simulator performs four operations as follows: 1) Find the cache line which has the highest control bits (LRU) in the upper memory; 2) Replace data brought from the lower level to the upper level memory. The operation for ‘Replace data from lower to upper level memory’ is as follows: Replace the current tag of the upper level memory with the new tag of the fetched memory address. 3) Check ‘VALIDBIT=1’ whether it is ‘0’ or not. If it is ‘0’, then set valid control bit as ‘1’; and 4) Do ‘UPDATE LRU.’

If write policy is write-back, find the line of the highest control bits (LRU) in the upper memory. Then, the simulator checks the dirty bit for the cache line:

- If dirty bit is zero, then simulator perform the following operations: 1) Do ‘Replace data from lower level to upper level memory’; 2) Do ‘VALIDBIT=1’; then 3) Do ‘UPDATELRU’;
- If dirty bit is ‘1’, the simulator performs the following operations: 1) Do ‘Replace data from upper level to lower level memory.’ The operation for ‘Replace data from upper to lower level memory’ is as follows: First, it combines index and tag value derived from the upper level memory to compute the lower level memory address. Then, access to the lower level memory to update the data; 2) Do ‘DIRTYBIT=0’; 3) Do ‘Replace data from lower to upper level memory’; and 4) Do ‘VALIDBIT=1’; then 5) Do ‘UPDATELRU.’

3.3.4 Cache Miss for Store Memory Address

In this case, the proposed simulator checks write policy. If the write policy is write-through then the simulator performs only one operation which is to update data into the lower level memory. Otherwise, the write policy is write-back. Then, the simulator does all operations, which are mentioned in section 3.3.3 including ‘DIRTYBIT=1.’

3.4 Implementing Cache Hit/Miss based on Snoopy Cache coherence

In this step, cache hit/miss is implemented based on Snoopy cache-coherence protocol for multi-core cache schemes. Figure 3.5 shows the transition diagram for Snoopy cache coherence protocol [18]. Snoopy protocol contains three different states, such as Modified, Shared, and Invalid.

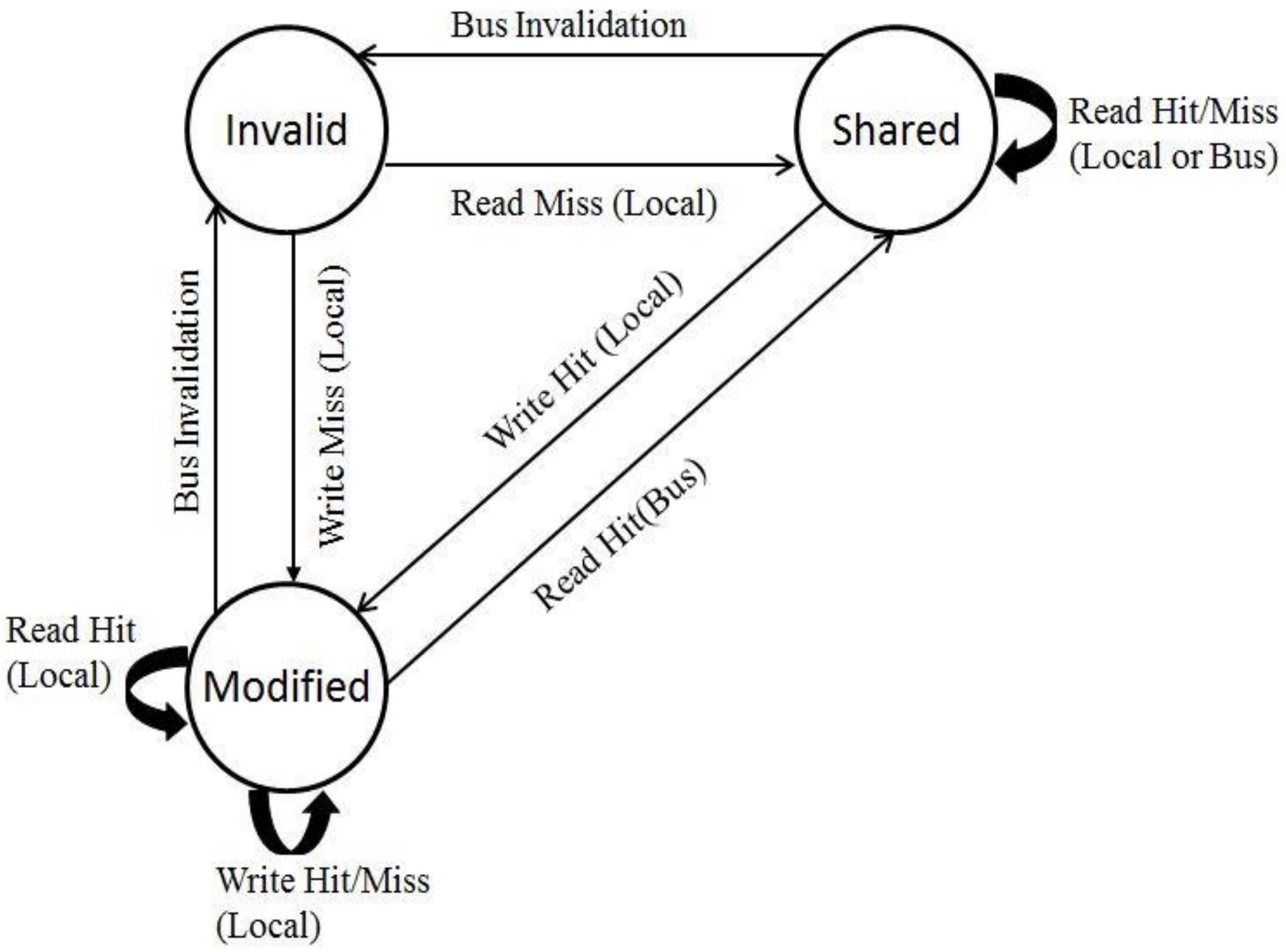


Figure 3.5: Snoopy coherence transition diagram [18]

Three states in Figure 3.5 are explained in brief as follows: 1) Modified state: For read hit (bus), current state will be changed into Shared state. In case of ‘Bus invalidation signal,’ the state will be transferred to Invalid state. For read/write hit (local) write miss (local), the state will be stayed at the current state; 2) Shared state: For write hit (local), current state will be moved to Modified state. For 'Bus invalidation signal,' it will be changed into Invalid state. For read hit/miss (local or bus), state will be remained at the current state; and 3) Invalid state: For read miss (local), current state will be moved to Shared state. For write miss (local), it will be moved to Modified state.

3.4.1 Cache Hit for Snoopy Cache Coherency Protocol

Figure 3.6 is the flowchart of Snoopy Cache coherence protocol. The leftmost part in Figure 3.6 shows the cache hit condition. In this case, the proposed simulator checks the instruction type for a fetched line from a trace file (refer to section 3.2):

- If the instruction type is 'r' (2nd column in Figure 3.3, read operation), then the simulator proceeds one operation, such as 'UPDATE LRU' (refer to section 3.3.1);
- If the instruction type is 'w,' then the simulator checks the status bit of the cache line:
    - If the status bit is 'S (Shared),' then simulator performs the following operations: 1) Do 'UPDATE LRU'; 2) Update 'STATUSBIT ='M'; and 3) Do 'SEND INVALID SIGNAL ON THE BUS';
    - If the status bit is 'M (Modified)' then the simulator only does 'UPDATE LRU'.

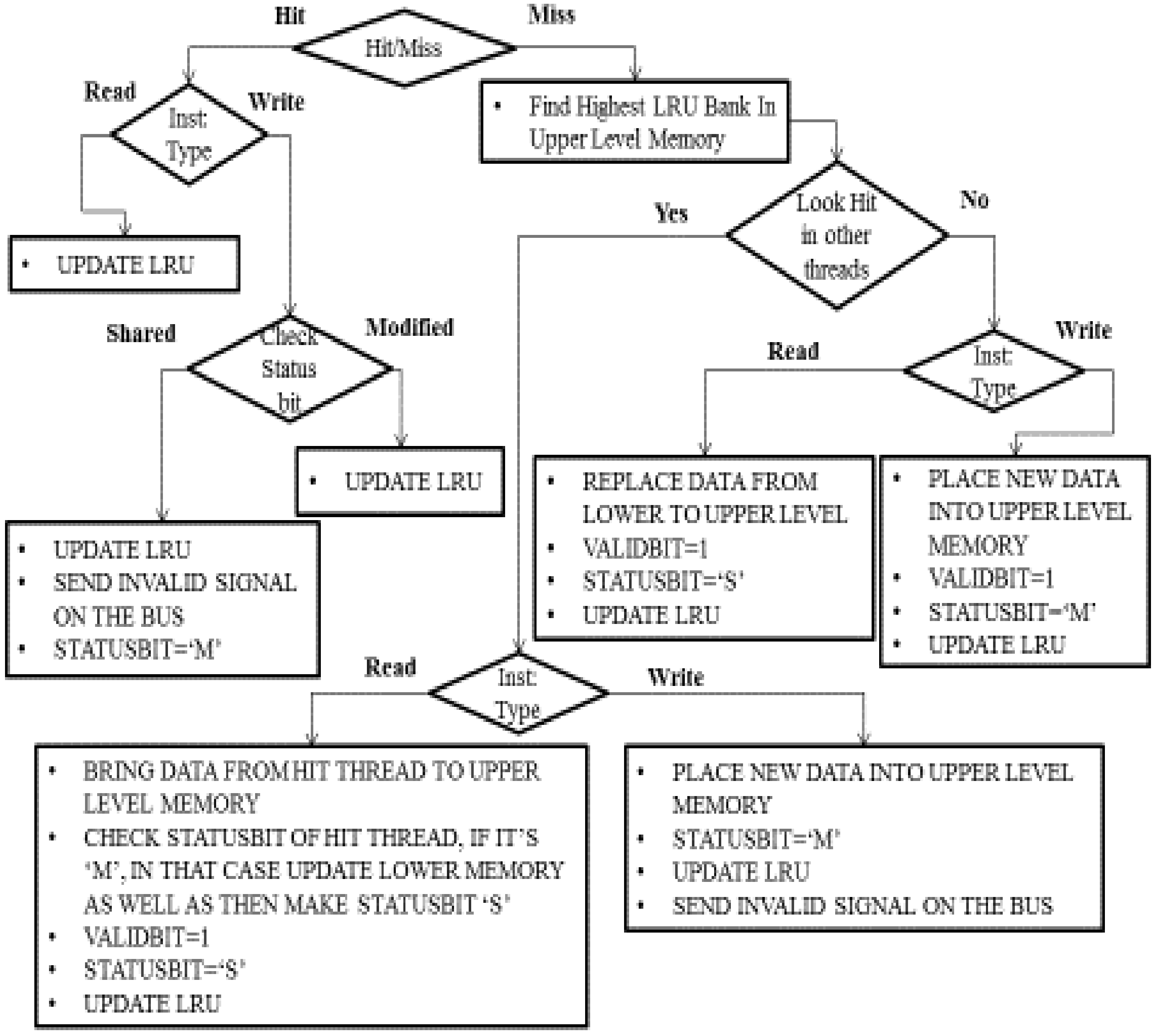

Hit
Hit/Miss
Miss
Read
Inst: Type
Write
Find Highest LRU Bank In Upper Level Memory
UPDATE LRU
Yes
Look Hit in other threads
No
Shared
Check Status bit
Modified
Read
Inst: Type
Write
UPDATE LRU
REPLACE DATA FROM LOWER TO UPPER LEVEL
VALIDBIT=1
STATUSBIT='S'
UPDATE LRU
PLACE NEW DATA INTO UPPER LEVEL MEMORY
VALIDBIT=1
STATUSBIT='M'
UPDATE LRU
UPDATE LRU
SEND INVALID SIGNAL ON THE BUS
STATUSBIT='M'
Read
Inst: Type
Write
BRING DATA FROM HIT THREAD TO UPPER LEVEL MEMORY
CHECK STATUSBIT OF HIT THREAD, IF IT'S 'M', IN THAT CASE UPDATE LOWER MEMORY AS WELL AS THEN MAKE STATUSBIT 'S'
VALIDBIT=1
STATUSBIT='S'
UPDATE LRU
PLACE NEW DATA INTO UPPER LEVEL MEMORY
STATUSBIT='M'
UPDATE LRU
SEND INVALID SIGNAL ON THE BUS

Figure 3.6: Snoopy cache coherence protocol

3.4.2 Cache Miss for Snoopy cache coherency protocol

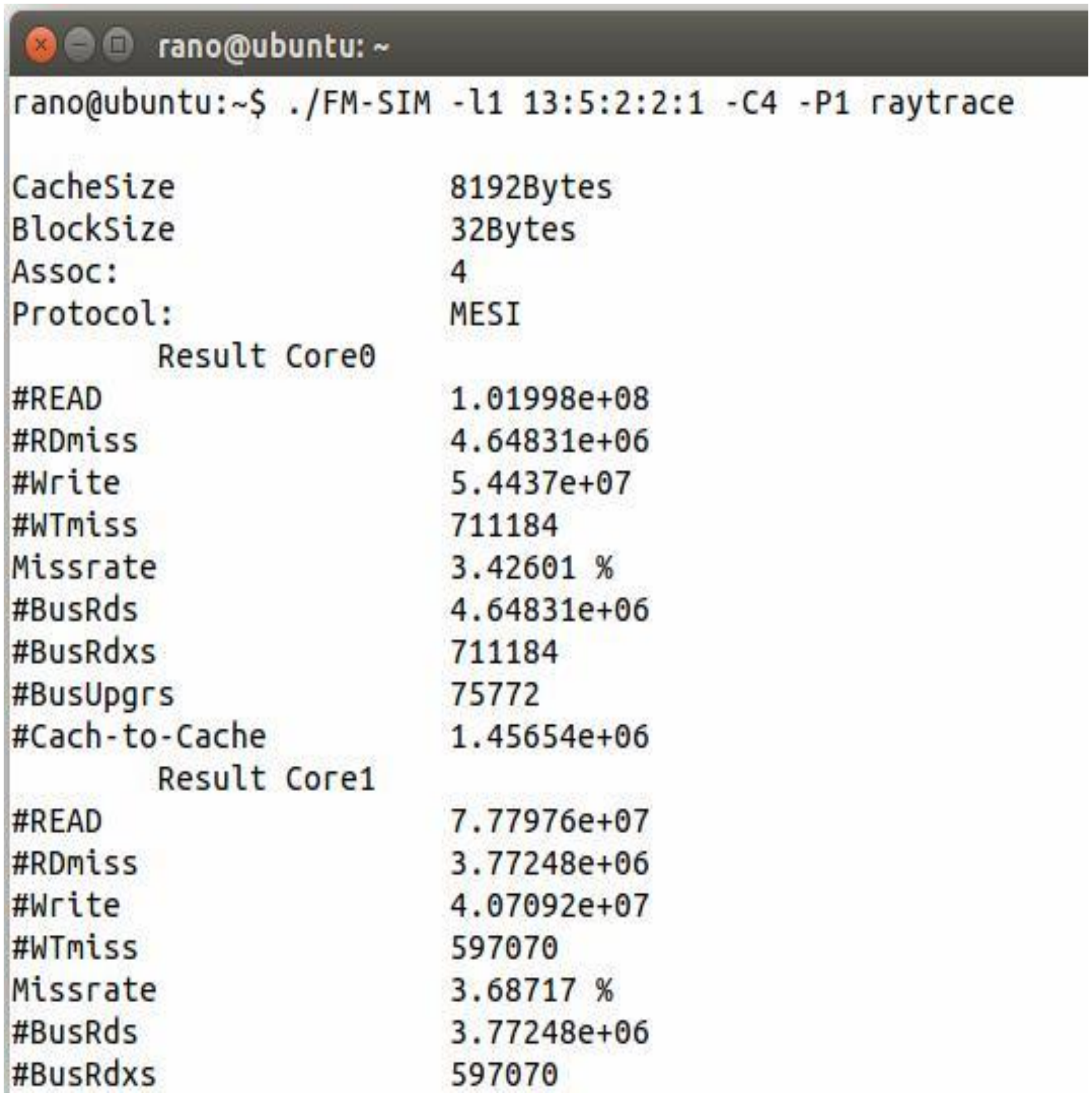
The rightmost part in Figure 3.6 shows the cache miss condition. In this case, the simulator checks the highest LRU control bit in the upper memory (e.g., cache memory). Then, the simulator tries to check a hit from other threads:

- If the instruction is 'hit' in other threads, the simulator checks the instruction type:

    - If the instruction is ‘r (read)’, the simulator performs the following operations: 1) Bring data from the hit thread to upper-level memory of local thread; 2) Check status bit of the hit thread. If it is ‘M’, the simulator updates the lower-level memory as well before changing the status bit into ‘S’ for the hit thread; 3) Do ‘VALIDBIT=1’; 4) Do ‘STATUSBIT=’S’; and 5) Do ‘UPDATE LRU’;
    - If the instruction is ‘w (write)’, then simulator performs the following operation: 1) Do ‘PLACE NEW DATA INTO the UPPER-LEVEL MEMORY OF LOCAL THREAD’; 2) Update ‘STATUS=’M’; 3) Do ‘UPDATE LRU’; and 4) Do ‘SEND INVALID SIGNAL ON THE BUS’;
- If the instruction is not a hit from any threads, then the simulator checks the instruction type:
    - If the instruction is ‘r’, then simulator performs the following operation: 1) Do ‘Replace data from lower to the upper level memory’; 2) Do ‘VALIDBIT=1’; 3) Change ‘STATUSBIT=’S’; and 4) Do ‘UPDATE LRU’;
    - If the instruction is ‘w’, then simulator performs the following operation: 1) Do ‘Place new into upper level memory’; 2) Do ‘VALIDBIT=1’; and 3) Update ‘STATUSBIT=’M’; 4) Do ‘UPDATE LRU.’

3.5 Providing Output Results

After running a trace file, the simulator produces the following results per core in Figure 3.7, which provides a total number of loads, stores and other instructions, number of misses, miss rates, etc. It is possible to add more output results according to your research analysis purpose (e.g., power consumption).



```
rano@ubuntu:~$ ./FM-SIM -l1 13:5:2:2:1 -C4 -P1 raytrace

CacheSize                 8192Bytes
BlockSize                 32Bytes
Assoc:                    4
Protocol:                 MESI
        Result Core0
#READ                     1.01998e+08
#RDmiss                   4.64831e+06
#Write                    5.4437e+07
#WTmiss                   711184
Missrate                  3.42601 %
#BusRds                   4.64831e+06
#BusRdxs                  711184
#BusUpgrs                 75772
#Cach-to-Cache            1.45654e+06
        Result Core1
#READ                     7.77976e+07
#RDmiss                   3.77248e+06
#Write                    4.07092e+07
#WTmiss                   597070
Missrate                  3.68717 %
#BusRds                   3.77248e+06
#BusRdxs                  597070
```

Figure 3.7: Output results.

3.6 Porting a Cache Memory and Coherency Protocol to FM-SIM

This section explains how to port (or implement) a designed cache memory or cache coherency protocol into the FM-SIM. The cache design includes any cache memories (e.g., 2-way skewed-associative in Section 3.6.1) and any cache coherence protocols (e.g., MESI in Section 3.6.2).

3.6.1 Cache memory porting: 2-way skewed-associative cache memory

2-way skewed cache memory [23] is similar to 2-way set-associative cache memory except for the mapping (or indexing) function and (re)placement policy. The 2-way skewed uses an XORing mapping function for the two banks; for Bank 0, it uses un-hashed indexed, for Bank 1, it uses a hashed indexed function. It employs *Pseudo-LRU* replacement policy [23]. The replacement control bit is located at each cache line in Bank 0; if the data is missed in Bank 0, then control bit value will be set to '1' after replacement from the lower memory. The '1' means for the next cache miss in that line, the data will be updated to Bank 1 (then, control bit is updated to '0'). The flowchart in Figure 3.8 is similar to Figure 3.4 except the doted boxes, which are related with mapping function and replacement policy (refer to [23]).

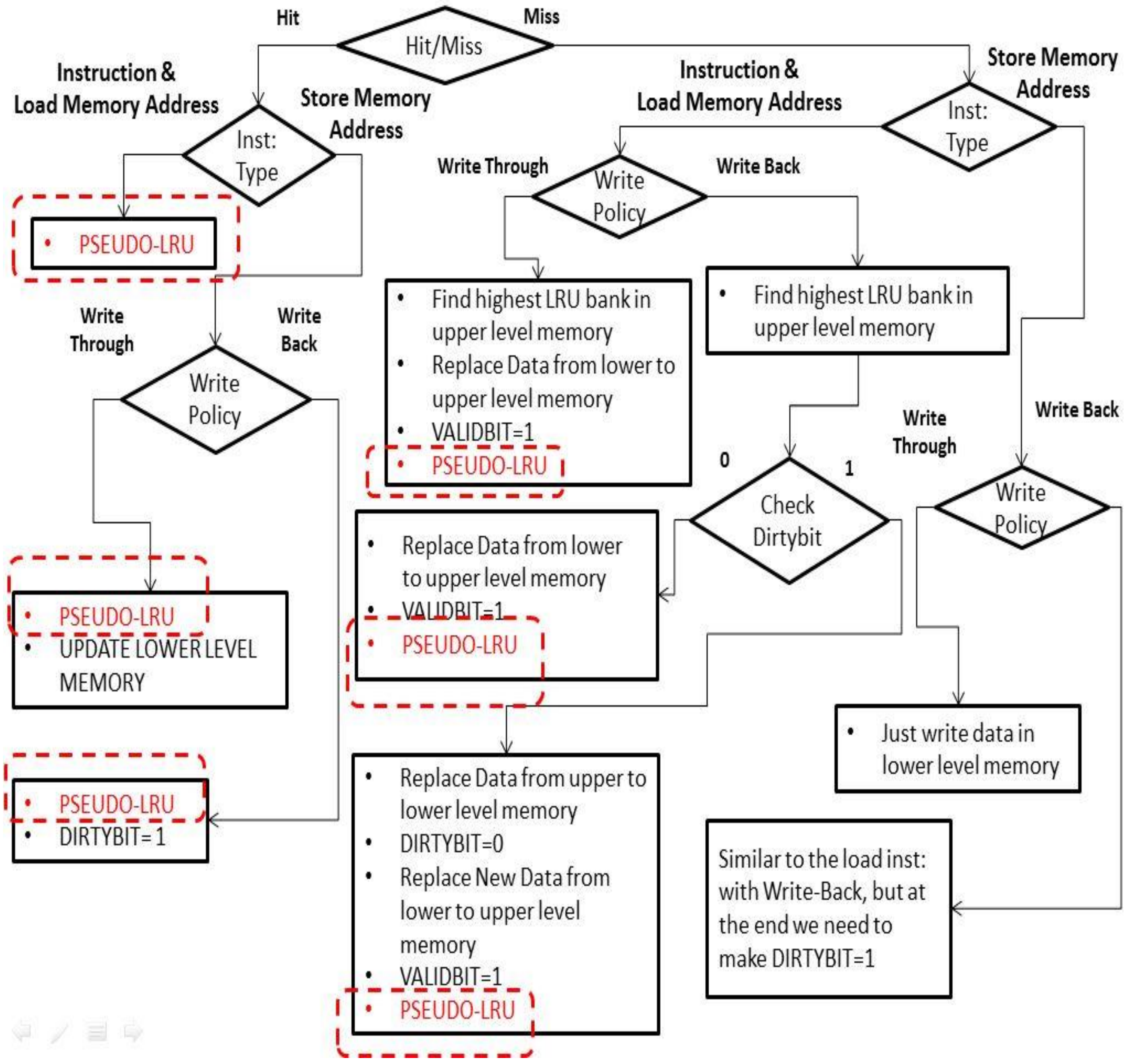

Hit/Miss
Hit
Miss
Instruction & Load Memory Address
Store Memory Address
Inst: Type
PSEUDO-LRU
Write Through
Write Back
Write Policy
PSEUDO-LRU
UPDATE LOWER LEVEL MEMORY
PSEUDO-LRU
DIRTYBIT=1
Instruction & Load Memory Address
Store Memory Address
Inst: Type
Write Through
Write Back
Write Policy
Find highest LRU bank in upper level memory
Replace Data from lower to upper level memory
VALIDBIT=1
PSEUDO-LRU
Find highest LRU bank in upper level memory
0
1
Check Dirtybit
Replace Data from lower to upper level memory
VALIDBIT=1
PSEUDO-LRU
Replace Data from upper to lower level memory
DIRTYBIT=0
Replace New Data from lower to upper level memory
VALIDBIT=1
PSEUDO-LRU
Write Through
Write Back
Write Policy
Just write data in lower level memory
Similar to the load inst: with Write-Back, but at the end we need to make DIRTYBIT=1

Figure 3.8: Four cases for implementing cache hit/miss for 2-way skewed-associative cache memory

3.6.2 Cache coherence protocol porting: MESI

MESI contains four different states, such as Modified, Exclusive, Shared, and Invalid. Figure 3.9 shows the transition diagram for MESI (Modified-Exclusive-Shared-Invalid) cache coherence protocol [18].

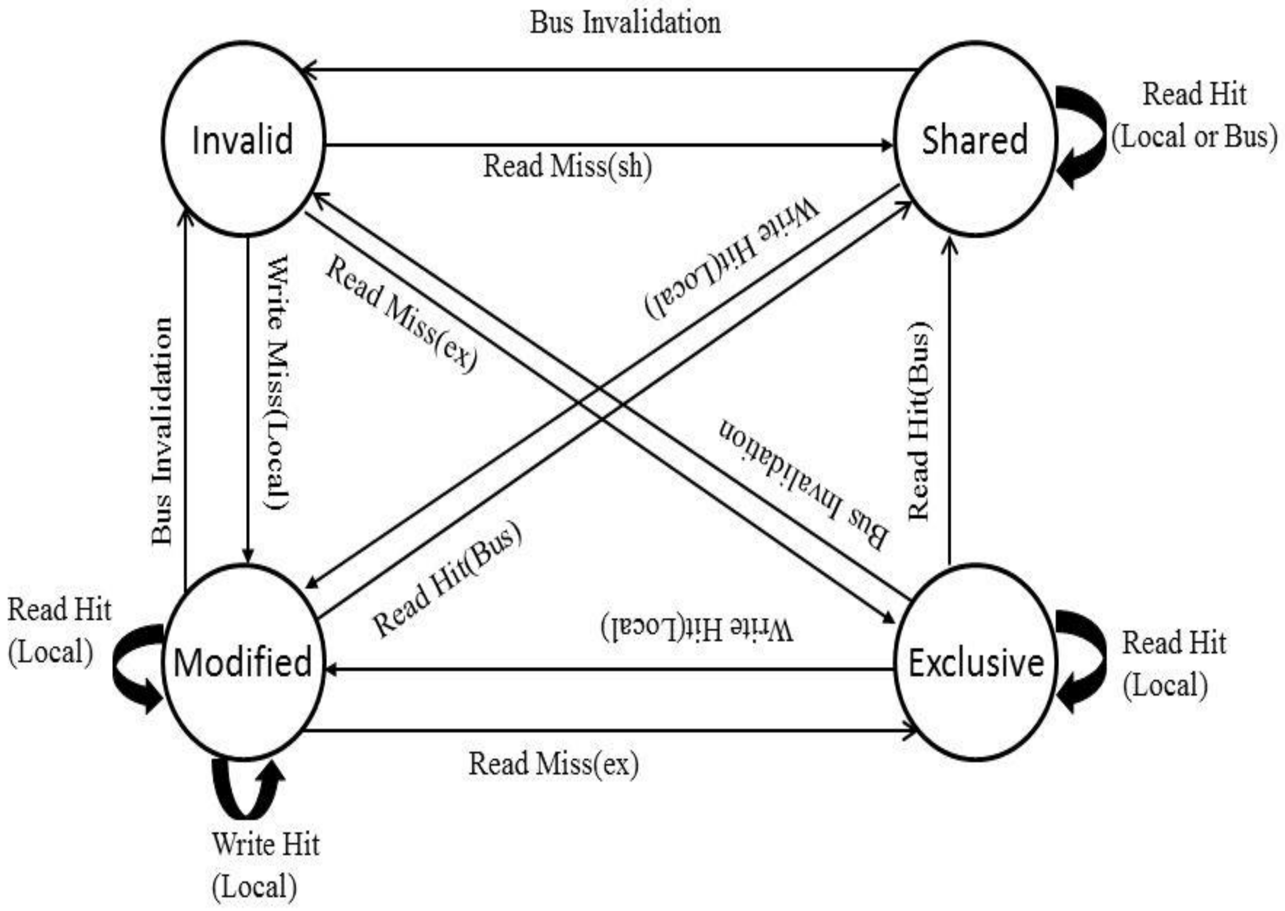


Figure 3.9: MESI coherence transition diagram [18]

Four states in Figure 3.9 are explained in brief as follows: 1) Modified state: For read miss, current state will be changed into Exclusive state. For read hit (bus), current state will be moved to Shared state. In case of 'Bus invalidation signal,' the state will be transferred to Invalid state. For read/write hit (local), the state will be stayed at the current state; 2) Exclusive state: For read hit (bus), current state will be changed into Shared state. For write hit (local), the state will be changed into Modified state, and for 'Bus invalidation signal,' the state will be changed into Invalid state, and for read hit(local), the state will be remained the same state; 3) Shared state: For write hit (local), current state will be moved to Modified state. For 'Bus invalidation signal,' it will be changed into Invalid state. For read hit (local or bus), state will be remained at the current state; and 4) Invalid state: For read miss (ex), current state will be moved to Exclusive state. For read miss (sh), current state will be moved to Shared state. For write miss (local), it will be moved to Modified state.

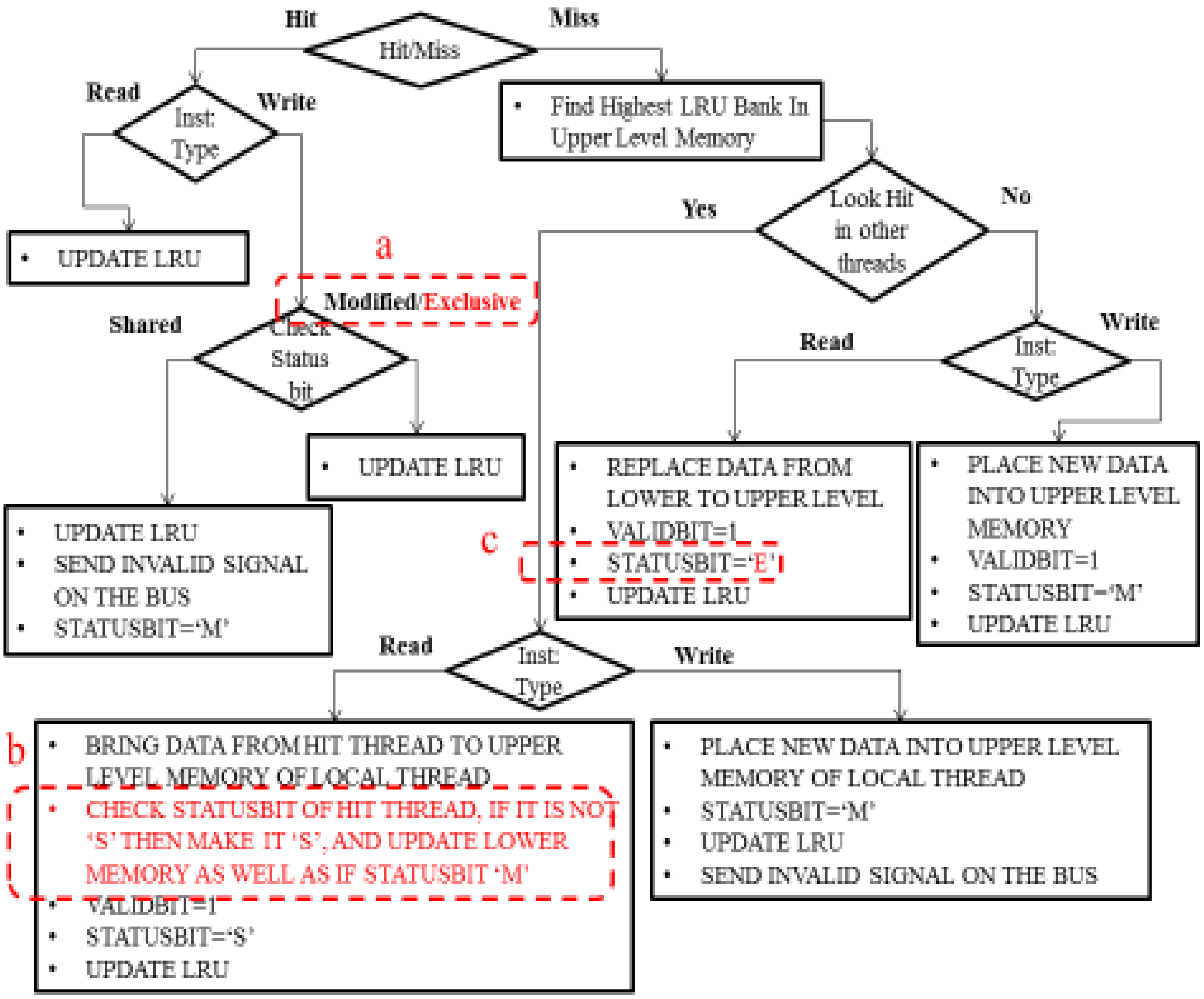

Hit
Hit/Miss
Miss
Read
Inst: Type
Write
Find Highest LRU Bank In Upper Level Memory
UPDATE LRU
Yes
Look Hit in other threads
No
a
Modified/Exclusive
Shared
Check Status bit
UPDATE LRU
Read
Inst: Type
Write
UPDATE LRU
SEND INVALID SIGNAL ON THE BUS
STATUSBIT='M'
REPLACE DATA FROM LOWER TO UPPER LEVEL
VALIDBIT=1
c
STATUSBIT='E'
UPDATE LRU
PLACE NEW DATA INTO UPPER LEVEL MEMORY
VALIDBIT=1
STATUSBIT='M'
UPDATE LRU
Read
Inst: Type
Write
b
BRING DATA FROM HIT THREAD TO UPPER LEVEL MEMORY OF LOCAL THREAD
CHECK STATUSBIT OF HIT THREAD, IF IT IS NOT 'S' THEN MAKE IT 'S', AND UPDATE LOWER MEMORY AS WELL AS IF STATUSBIT 'M'
VALIDBIT=1
STATUSBIT='S'
UPDATE LRU
PLACE NEW DATA INTO UPPER LEVEL MEMORY OF LOCAL THREAD
STATUSBIT='M'
UPDATE LRU
SEND INVALID SIGNAL ON THE BUS

Figure 3.10: MESI cache coherence protocol

The Figure 3.10 shows the flowchart for the MESI protocol [18] and it is similar to Figure 3.6 (Snoopy cache coherence protocol), but the MESI has one more state, 'Exclusive.'

The differences (compared to Snoopy) are indicated in doted boxes in Figure 3.10. The doted box ‘a’ means: 1) Check the Status bit whether it is Modified or Exclusive. If then, the MESI will do ‘UPDATE LRU’; 2) Doted box ‘b’ is for ‘STATUS BIT is EXCLUSIVE,’ when the fetched instruction is ‘read miss and not found in other threads’; and 3) Doted box ‘c’ is for the case that the instruction is ‘read miss but hit in other thread’. If then, check the STATUSBIT OF HIT THREAD. If it is not SHARED, then update it to SHARED, and update lower memory as well such that the status bit is modified.

3.6.3 Cache coherence protocol porting: FIREFLY

FIREFLY contains three different states, such as Valid-Exclusive, Shared, and Dirty. Figure 3.11 shows the transition diagram for FIREFLY cache coherence protocol [24].

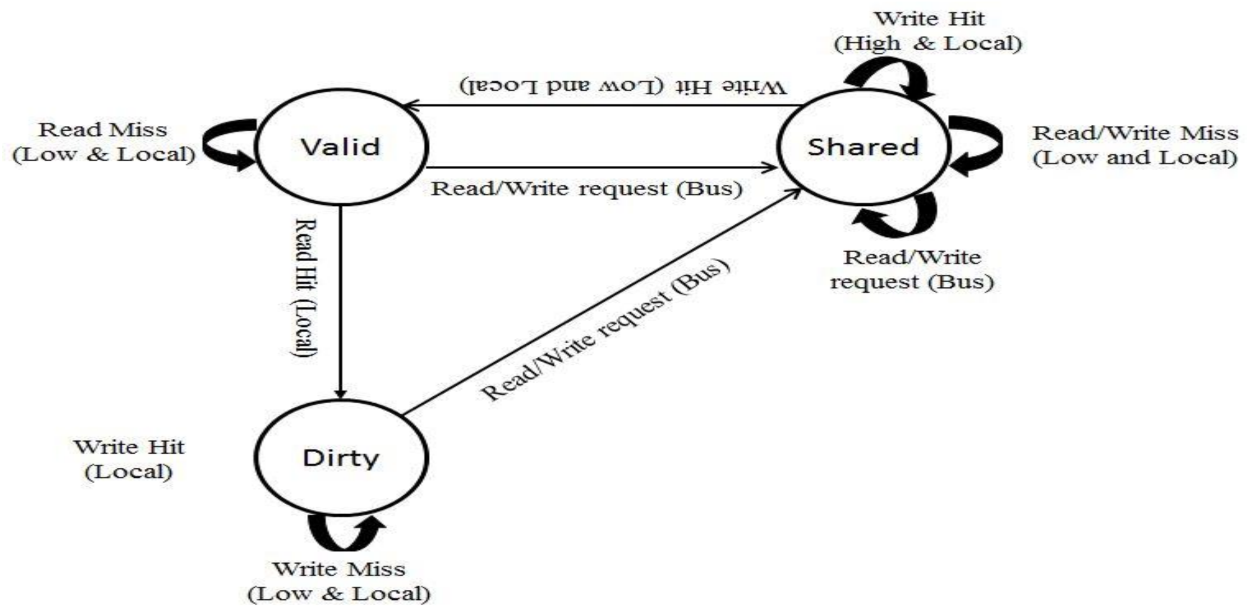


Figure 3.11: FIREFLY coherence transition diagram [24]

In Figure 3.11, ‘High’ means the requested data by processor, which might exists in other processor’s cache memory, and ‘Low’ means the requested data does not exist in any other cache memory.   There are three states in Figure 3.11 as follows: 1) Valid-Exclusive state - For write hit (Local), current state will be changed into Dirty state. For bus (read or write) request, current state will be changed to Shared state. For read miss (Low & local), state will be remained at the current state; 2) Shared state - For write hit (Low & local), current state will be changed into Valid-Exclusive state. For read/write miss (Low & local), write hit (High and local), and bus (read or write) request, state will be remained at the current state; and 3) Dirty state - For bus (read or write) request, current state will be changed to Shared state. For write hit (local) and write miss (Low & local), state will be remained at the current state.

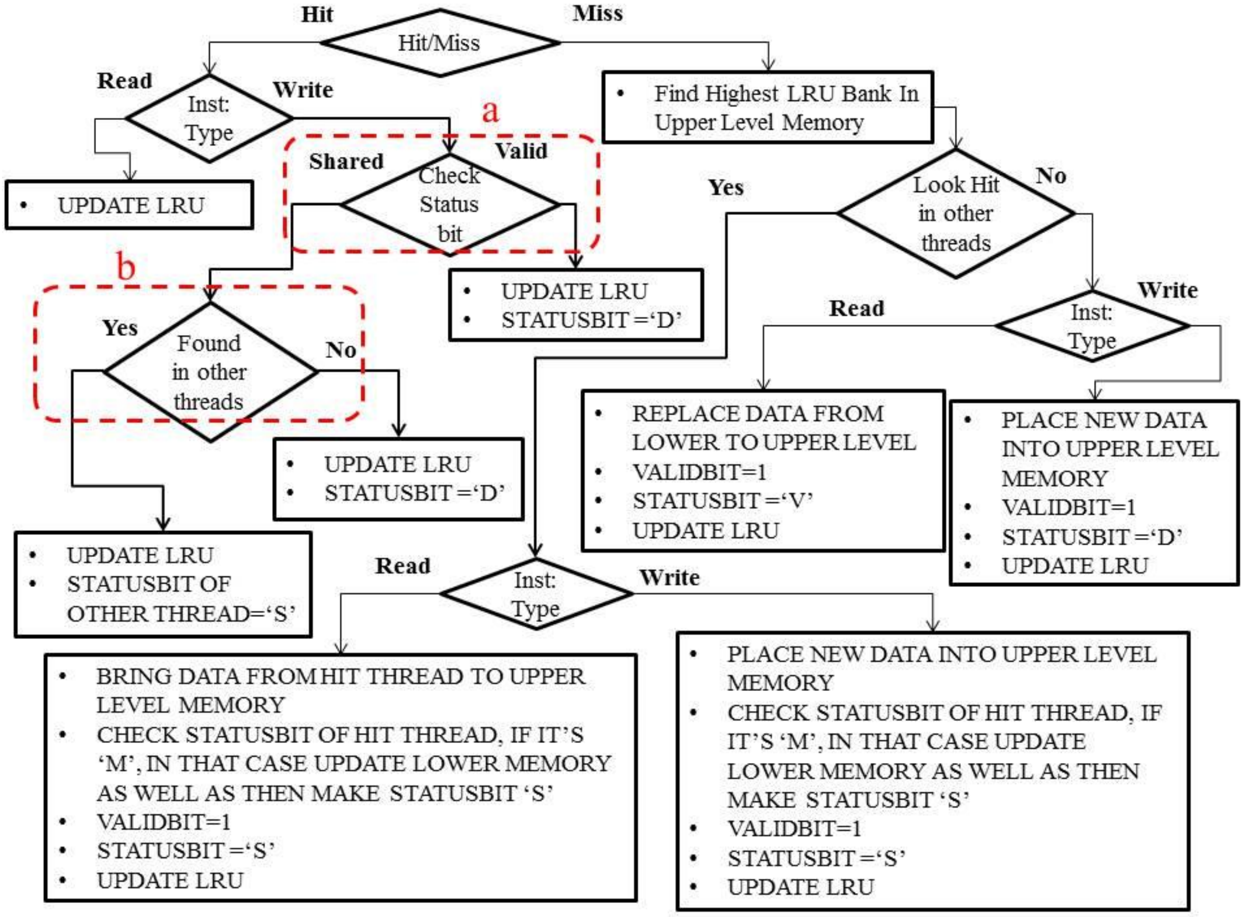

Hit
Hit/Miss
Miss
Read
Inst: Type
Write
a
Find Highest LRU Bank In Upper Level Memory
Shared
Check Status bit
Valid
UPDATE LRU
Yes
Look Hit in other threads
No
b
UPDATE LRU
STATUSBIT ='D'
Read
Inst: Type
Write
Yes
Found in other threads
No
REPLACE DATA FROM LOWER TO UPPER LEVEL
VALIDBIT=1
STATUSBIT ='V'
UPDATE LRU
PLACE NEW DATA INTO UPPER LEVEL MEMORY
VALIDBIT=1
STATUSBIT ='D'
UPDATE LRU
UPDATE LRU
STATUSBIT ='D'
UPDATE LRU
STATUSBIT OF OTHER THREAD='S'
Read
Inst: Type
Write
BRING DATA FROM HIT THREAD TO UPPER LEVEL MEMORY
CHECK STATUSBIT OF HIT THREAD, IF IT'S 'M', IN THAT CASE UPDATE LOWER MEMORY AS WELL AS THEN MAKE STATUSBIT 'S'
VALIDBIT=1
STATUSBIT ='S'
UPDATE LRU
PLACE NEW DATA INTO UPPER LEVEL MEMORY
CHECK STATUSBIT OF HIT THREAD, IF IT'S 'M', IN THAT CASE UPDATE LOWER MEMORY AS WELL AS THEN MAKE STATUSBIT 'S'
VALIDBIT=1
STATUSBIT ='S'
UPDATE LRU

Figure 3.12: FIREFLY cache coherence protocol

The Figure 3.12 shows the flowchart for FIREFLY protocol [24] and it is similar to fFigure 3.6 (Snoopy cache coherence protocol), but the FIREFLY has 'valid' state instead of 'invalid' state. The differences (compared to Snoopy) are indicated in the doted boxes in Figure

3.12. In the doted box, ‘a’ means 1) Check the Status bit whether it is Valid or Shared and 2) ‘b’ is for ‘STATUSBIT BIT is Shared,’ when the fetched instruction is ‘read hit and found in other threads’.

CHAPTER IV

SIMULATION METHODOLOGY

This section introduces how to simulate a designed cache scheme using the FM-SIM to measure its performance. For our experiments, SPEC 2006, Parsec, and SPLASH2x benchmark programs [8, 19] are used to evaluate performance for the multi-core cache memories. Figure 4.1 shows the overview of the simulation methodology, which is as follows: 1) SPEC, Parsec and SPLASH2x benchmark programs are compiled by using GCC compiler (or other compilers) to make execution file (binary code or machine language); 2) the execution file will be run at a Linux machine (ubuntu-15.04) using the Pin Tool to collect trace files; 3) Then, trace files are inserted into the proposed cache simulator to get the outputs. Here, you can port your designed multi-core cache scheme into the FM-SIM for your purpose; and 4) you can get the outputs for the performance analyses.

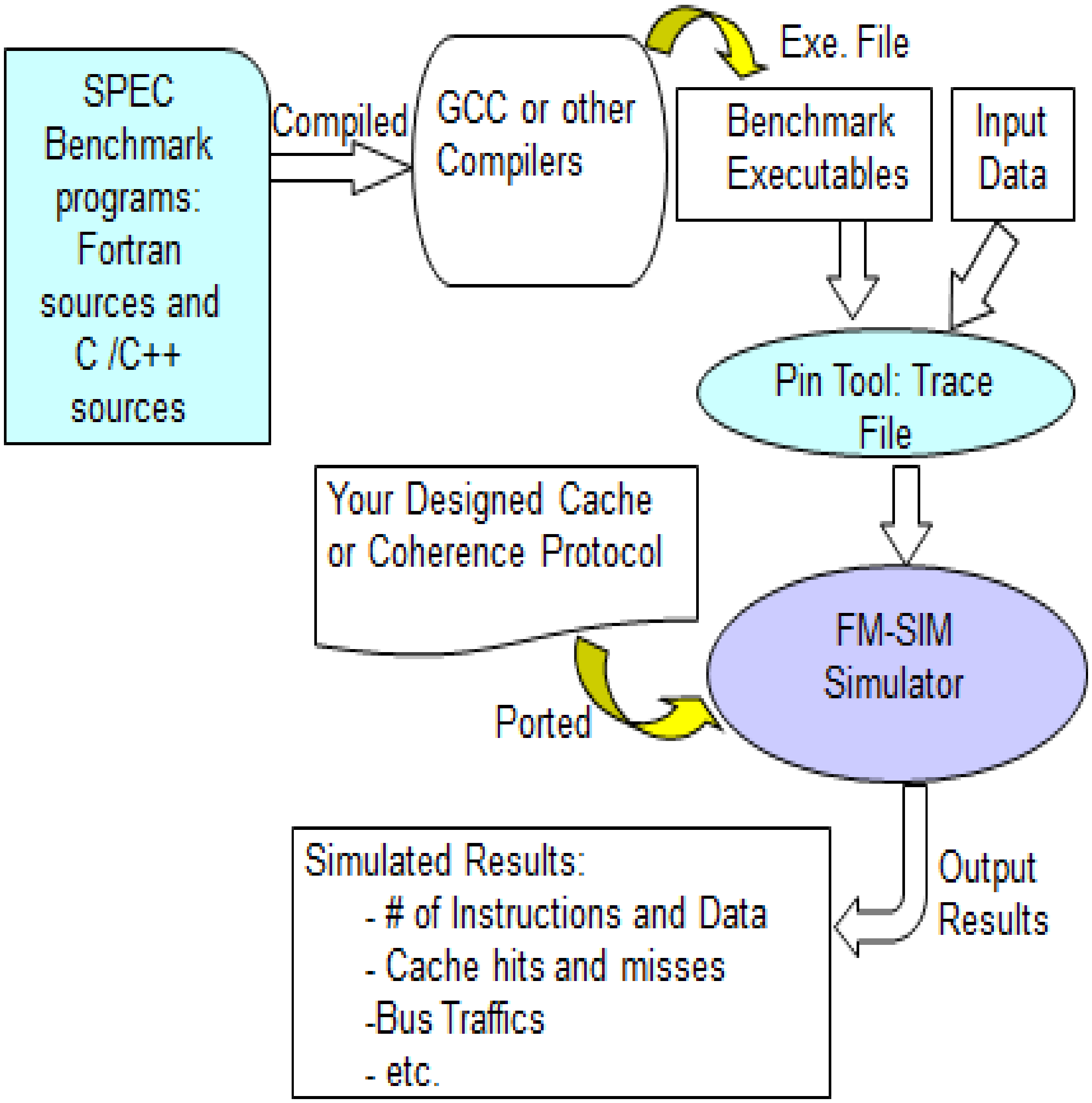


Figure 4.1: Simulation Methodology.

4.1 Trace Files for Benchmark Programs: SPEC2006, Parsec, and SPLASH2x

Table 4.1 and Table 4.2 show the SPEC-2006, Parsec, and SPLASH2x benchmark programs [8, 22] with the input data (train, ref, simsmall, simmedium, and simlarge) and # of instructions during run time. The Pin Tool was used to collect the following data, which are listed in Table 4.1 and 4.2. For example, 401.bzip2 is a SPEC2006 benchmark program, which has 142 BILLIONS dynamic instructions (train input).

Table 4.1: List of Trace files for SPEC2006 Benchmark Programs.

Programs	Instruction # (Input: Train)	Instruction # (Input: ref)
400.perlbench	2 BILLIONS	705 BILLIONS
401.bzip2	142 BILLIONS	348 BILLIONS
403.gcc	4 BILLIONS	63 BILLIONS
410.bwaves	142 BILLIONS	1,992 BILLIONS
416.gamess	177 BILLIONS	1,043 BILLIONS
429.mcf	20 BILLIONS	329 BILLIONS
433.milc	44 BILLIONS	1,354 BILLIONS
434.zeusmp	122 BILLIONS	2,154 BILLIONS
435.gromacs	504 BILLIONS	2,345 BILLIONS
436.cactusADM	84 BILLIONS	2,907 BILLIONS
437.leslie3d	239 BILLIONS	1,682 BILLIONS
444.namd	73 BILLIONS	2,703 BILLIONS
445.gobmk	37 BILLIONS	327 BILLIONS
450.soplex	8 BILLIONS	393 BILLIONS
453.povray	38 BILLIONS	1,188 BILLIONS
454.calculix	7 BILLIONS	6,951 BILLIONS
456.hmmer	314 BILLIONS	2,164 BILLIONS
458.sjeng	481 BILLIONS	2,321 BILLIONS
459.GemsFDTD	131 BILLIONS	1,788 BILLIONS

462.libquantum	15 BILLIONS	2,675 BILLIONS
464.h264ref	523 BILLIONS	3,018 BILLIONS
465.tonto	1,351 BILLIONS	3,466 BILLIONS
470.lbm	107 BILLIONS	1,418 BILLIONS
471.omnetpp	265 BILLIONS	701 BILLIONS
473.astar	136 BILLIONS	834 BILLIONS
481.wrf	310 BILLIONS	3,204 BILLIONS
482.sphinx3	32 BILLIONS	2,576 BILLIONS
483.xalancbmk	291 BILLIONS	1,106 BILLIONS

Table 4.2: List of Trace files for Parsec and SPLASH2x Benchmark Programs.

Programs	Instruction# (Input:simsmall)	Instruction# (Input:simmedium)	Instruction# (Input:simlarge)
barnes	1,912 M	4,135 M	37,839 M
fmm	2,541 M	10,890 M	44,336 M
ocean_cp	1,432 M	5,234 M	20,414 M
radiosity	1,438 M	1,438 M	1,438 M
raytrace	2,151 M	13,926 M	55,082 M
volrend	866 M	9,645 M	17,543 M
water_nsquared	618 M	24,113 M	134,105 M
water_spatial	4,445 M	9,084 M	36,210 M
blackscholes	233 M	930 M	3,719 M
bodytrack	1,415 M	4,520 M	15,873 M

ferret	1,451 M	3,957 M	11,948 M
fluidanimate	1,797 M	3,954 M	12,069 M
freqmine	993 M	3,026 M	9,259 M
vips	682 M	1,766 M	4,650 M

4.2 Pin Tool and Cache Configurations

Pin Tool (version 3.2) is an open-source software tool to generate a trace file to feed into the proposed cache simulator, FM-SIM, using benchmark programs. For doing this, we modified ‘pinatrace.cpp’ source code of the Pin Tool. Figure 4.2 shows a flowchart how Pin Tool produce trace files as follows: 1) Modify source code; 2) The instructions in a program (execution file) are intercepted by the Pin to produce a trace file as a result.

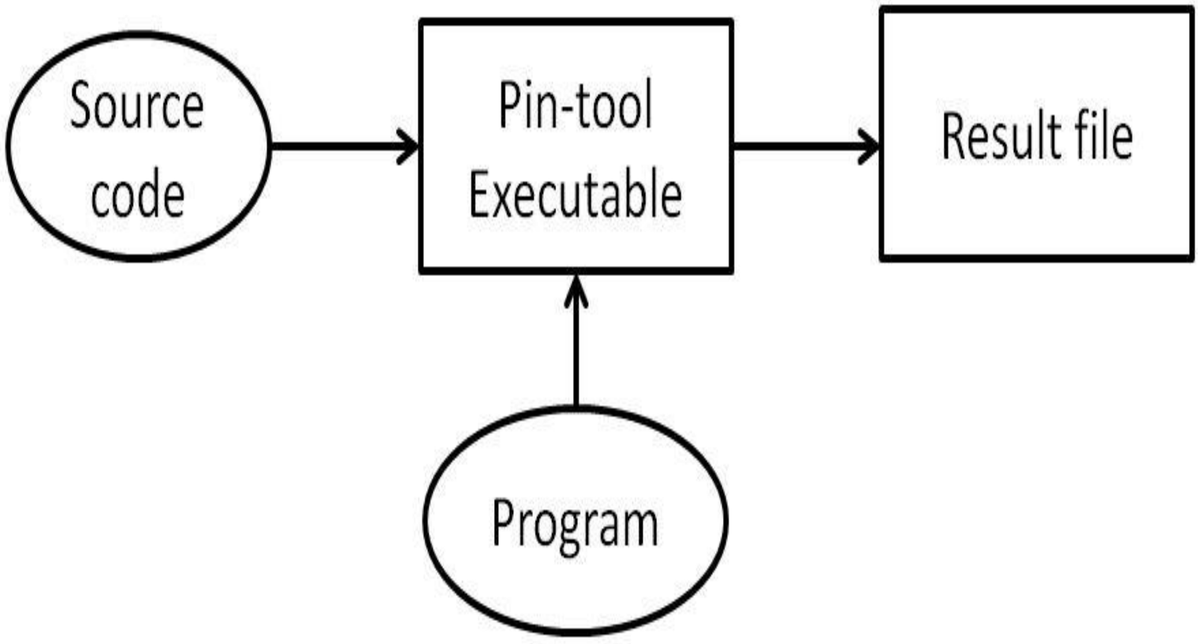


Figure 4.2: Pin-tool and Trace Files.

For reference, Figure 4.3 shows the command line for the Pin Tool to create trace files for benchmark programs as follows: 1) pin: Pin-tool simulator; 2) pinatrace.so: binary source code; 3) Benchmark program name (i.e., executable input file to Pin Tool): ‘raytrace (for example)’ is a SPLACH2x benchmark program; and 4) Benchmark program parameters: for example, ‘raytrace’ needs the following parameters to execute, such as ‘-s –pl –a4 trapot.env.’

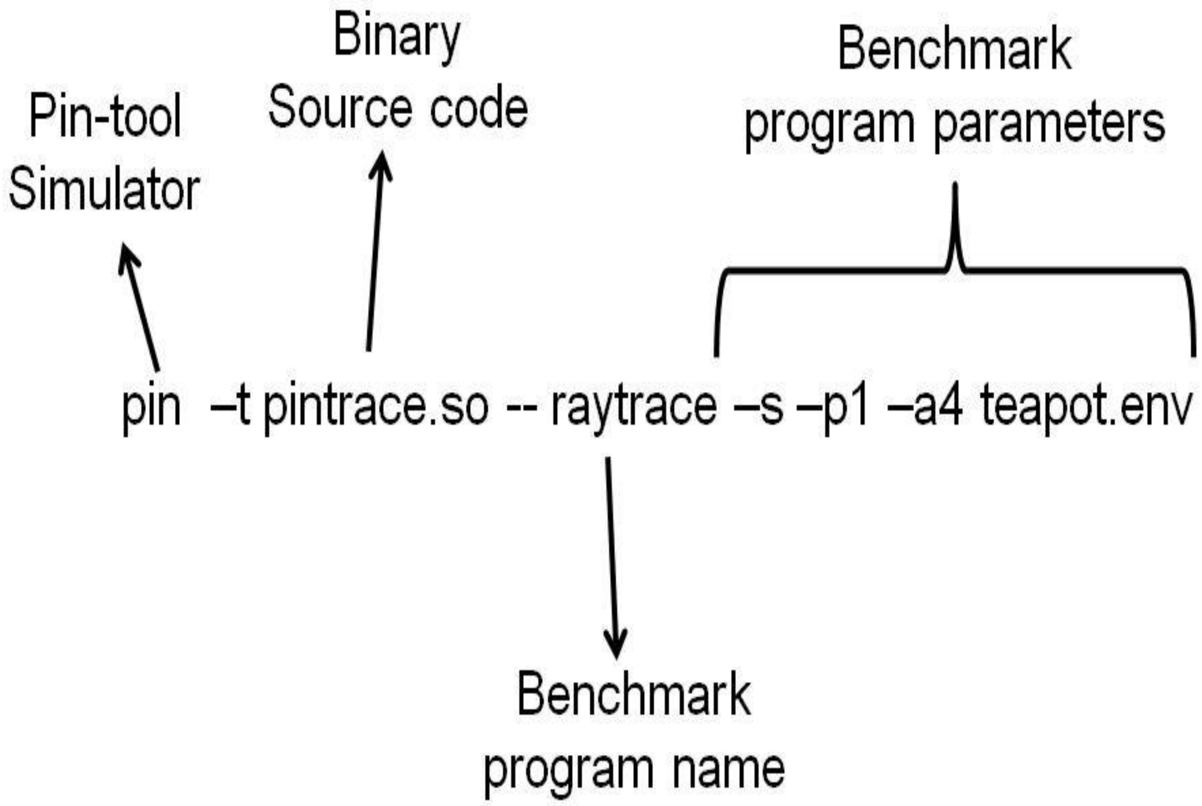

Binary Source code
Pin-tool Simulator
Benchmark program parameters
pin –t pintrace.so -- raytrace –s –p1 –a4 teapot.env
Benchmark program name

Figure 4.3: Command line for Pin Tool

Figure 4.4 shows the multi-core cache configuration implemented for the experiment results in Chapter 5. Each core has its own private cache and shares main memory with other cores. There is no limit for the number of cores in implementing in FM-SIM.

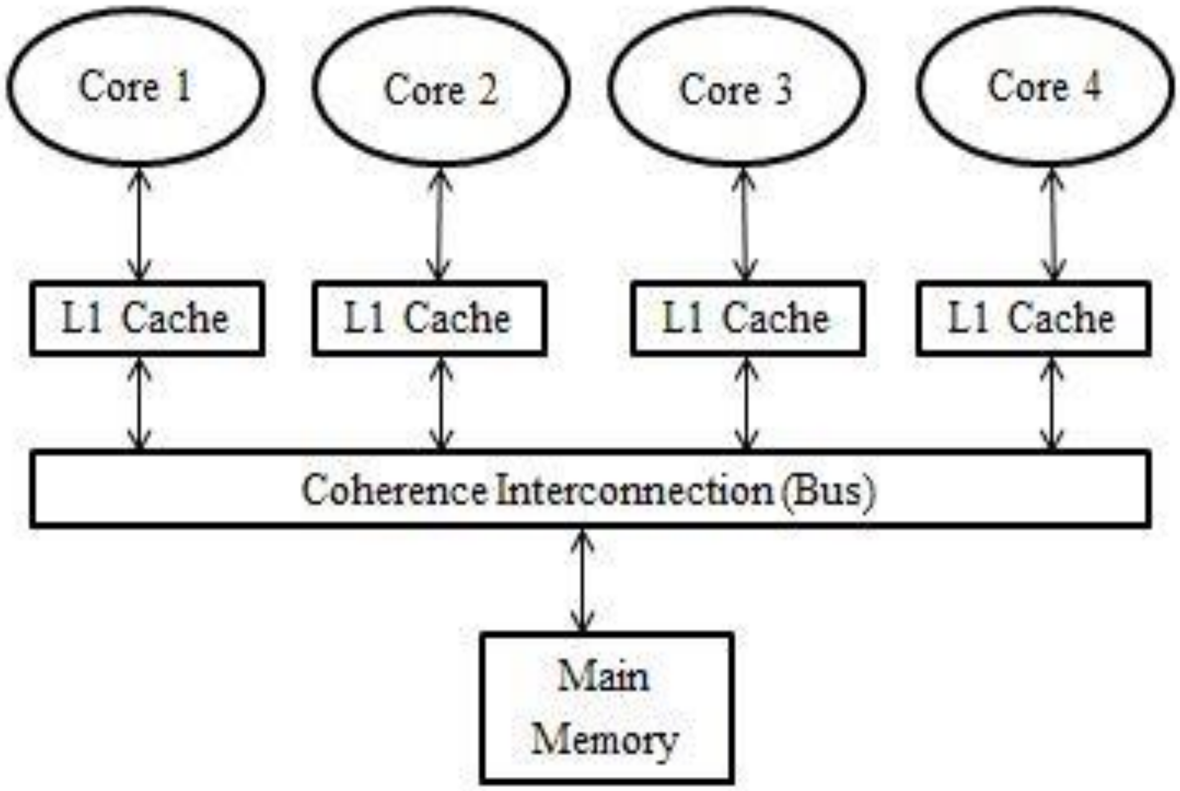


Figure 4.4: Multi-core cache configuration

CHAPTER V

EXPERIMENTAL RESULTS

The goal of the proposed multi-core cache simulator, FM-SIM, is to support for designing various types of cache memories for general-purpose CPU or embedded processors. Therefore, the FM-SIM should implement any cache memories (conventional or specific-purposed) to evaluate the performance using real application programs, such as SPEC 2006, PARSEC and SPLASH2x [8, 19]. This section provides four experimental results to verify and analyze the simulator outputs for single-core and multi-core cache schemes, such as direct-mapped (D-Way), 2-way set-associative, 4-way set-associative, 8-way set-associative, 2-way skewed (Skewed), FIREFLY cache-coherence protocol, and MESI cache-coherence protocol.

Figure 5.1 to 5.64 show the experimental results from the following multi-core cache parameters: 1) Multi-core parameters: # of cores (4 cores and 8 cores), set-associative (4-way and 8-way), block size (32 bytes and 64 bytes), cache sizes (2KB, 4KB, 8KB, 16KB, 32KB, and 64KB), MESI cache coherence protocol and Firefly cache coherence protocol; 2) Results: Miss rates (read and write cache misses), WHP (write-hit private), RHP (read-hit private), WHS (write-hit shared), RHS (read-hit shared), and RM (read miss). Figure 5.1 to 5.16 shows miss rates based on the cache sizes, such as from 4 KB to 256 KB using 4 benchmark programs and the results are similar to the ones on [21, 22]. In Figures 5.17 to 5.32, all the number of local hits are counted as ‘private.’ in addition, all the number of ‘Bus’ hits are counted as ‘shared.’ From the figures, we can claim that the proposed multi-core simulator works appropriately since the experimental results are similar to the results on [21, 22]. Figure 5.33 to 5.48 shows data transfer between caches based on the cache sizes, such as from 4 KB to 256 KB using 4 benchmark programs. Figure 5.49 to 5.64 shows bus traffics based on the cache sizes, such as from 4 KB to 256 KB using 4 benchmark programs. Figures 5.65 to 5.84 show the following results: 1) the cache miss rates of the Direct-mapped (D-Way) are the highest among other conventional cache memories; 2) 8-way set-associative (8-Way) works the best among other cache memories, which has the lowest cache miss rates; and 3) 2-way skewed (Skewed) performs in between 2-way set-associative (2-Way) and 4-way set-associative (4-Way). From the following figures, we can claim that the FM-SIM works appropriately since the experimental results are very reasonable compared to other results on [18, 20]. Therefore, based on the experimental results, we would like to claim that multi-core cache schemes can be performed accurately at the proposed FM-SIM like other open-source simulators, such as Multi2Sim. Figure 5.85 to 5.87 show results regarding Bus traffic corresponding Number of cores, bus traffic increase with increase the core number [21].

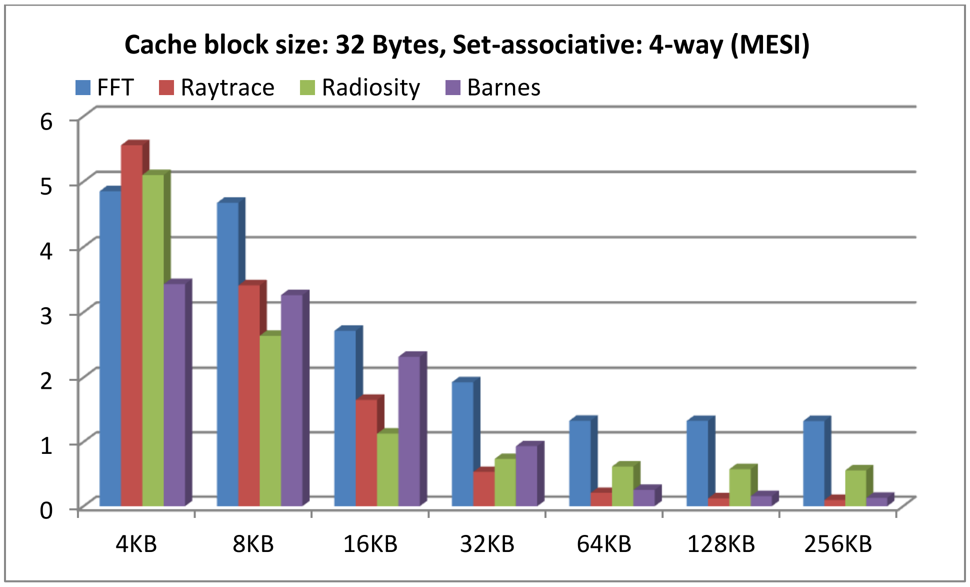


Figure 5.1: Miss Rate MESI (Block size 32 Bytes, Set-associative 4-Way, Core-4)

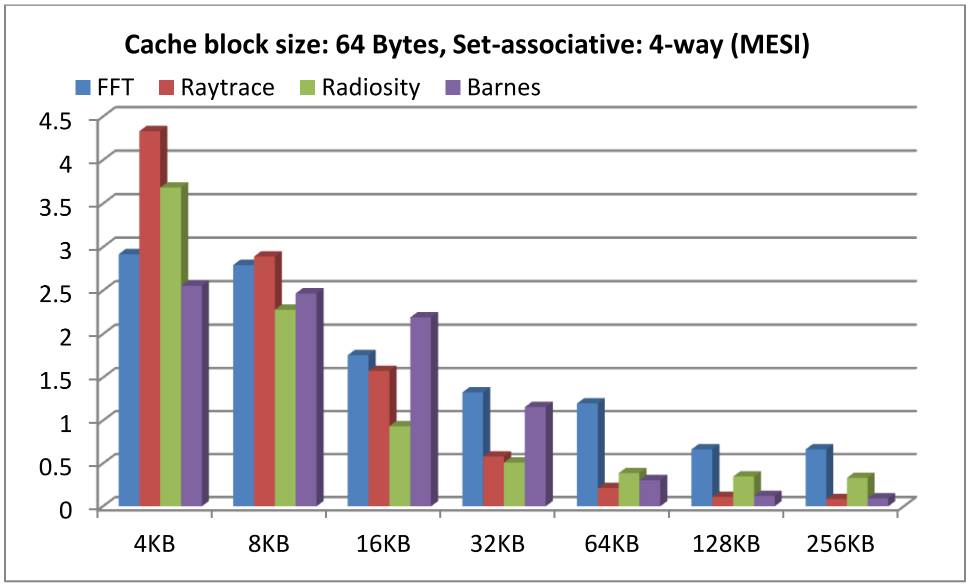


Figure 5.2: Miss Rate MESI (Block size 64 Bytes, Set-associative 4-Way, Core-4)

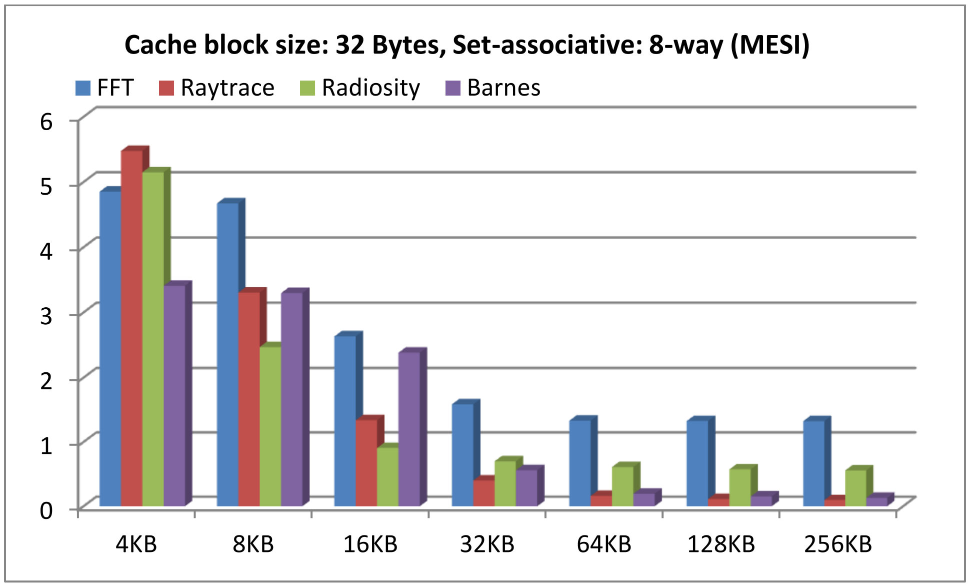


Figure 5.3: Miss Rate MESI (Block size 32 Bytes, Set-associative 8-Way, Core-4)

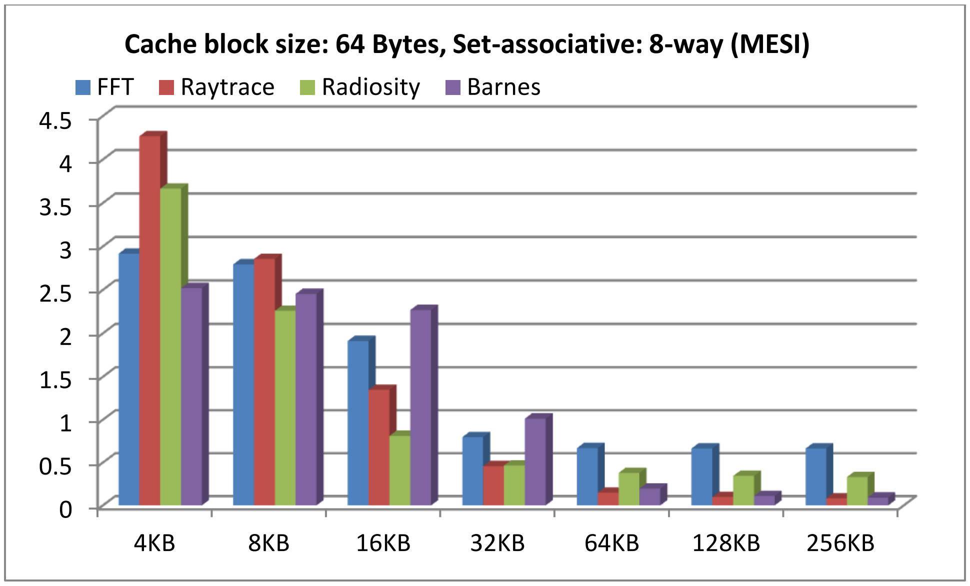


Figure 5.4: Miss Rate MESI (Block size 64 Bytes, Set-associative 8-Way, Core-4)

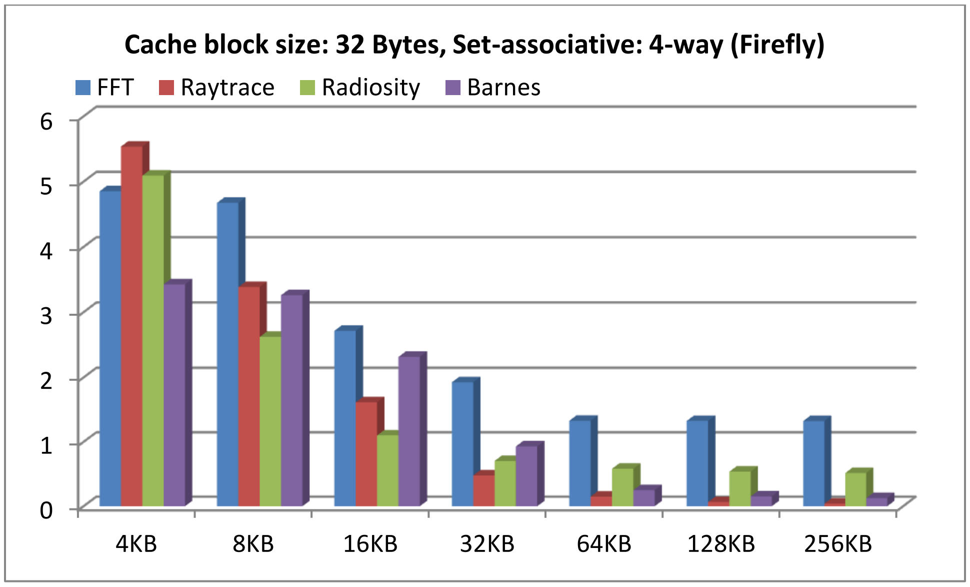


Figure 5.5: Miss Rate FIREFLY (Block size 32 Bytes, Set-associative 4-Way, Core-4)

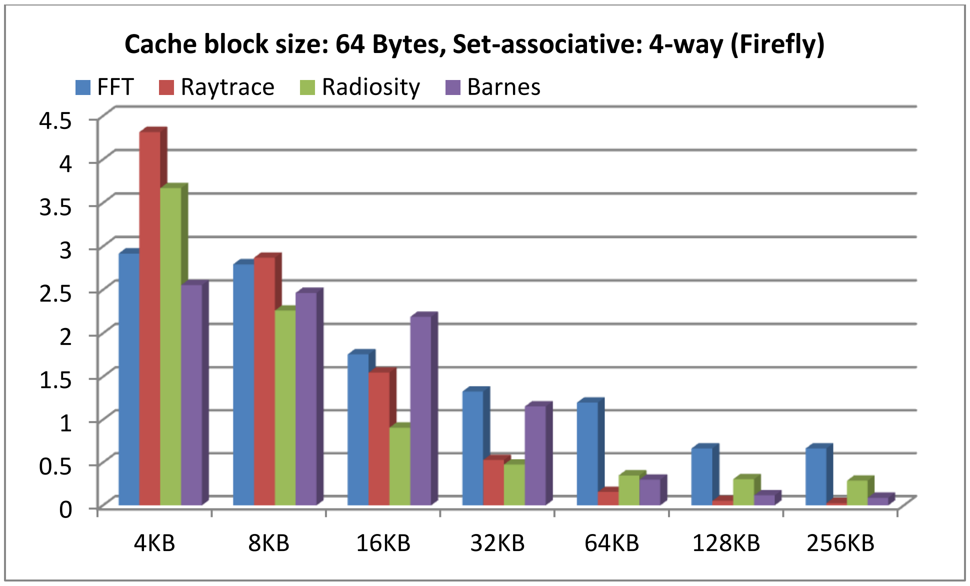


Figure 5.6: Miss Rate FIREFLY (Block size 64 Bytes, Set-associative 4-Way, Core-4)

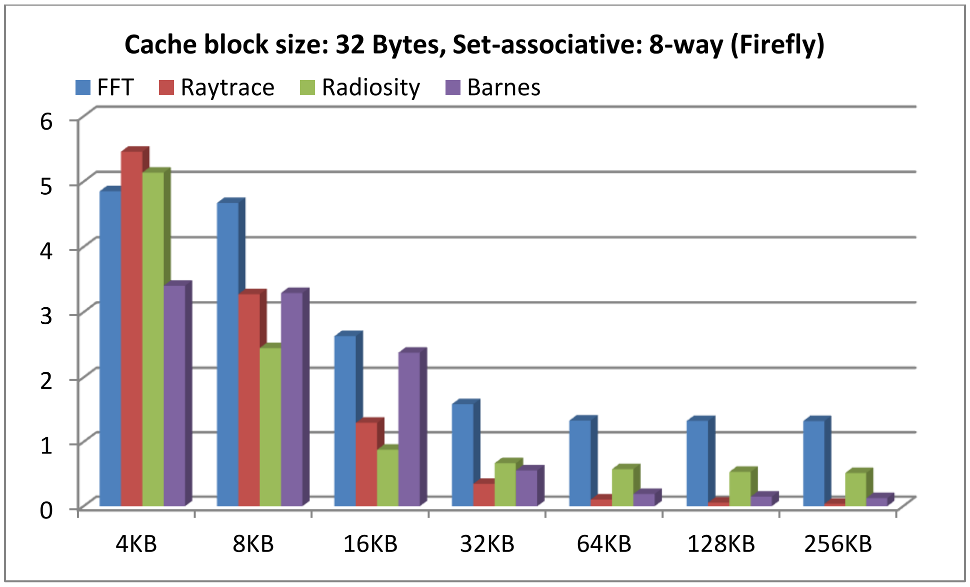


Figure 5.7: Miss Rate FIREFLY (Block size 32 Bytes, Set-associative 8-Way, Core-4)

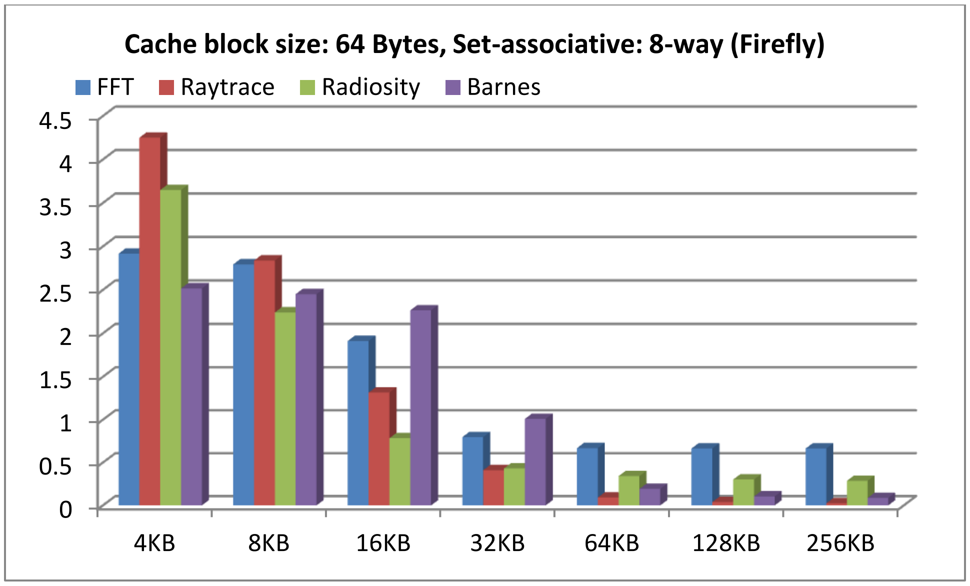


Figure 5.8: Miss Rate FIREFLY (Block size 64 Bytes, Set-associative 8-Way, Core-4)

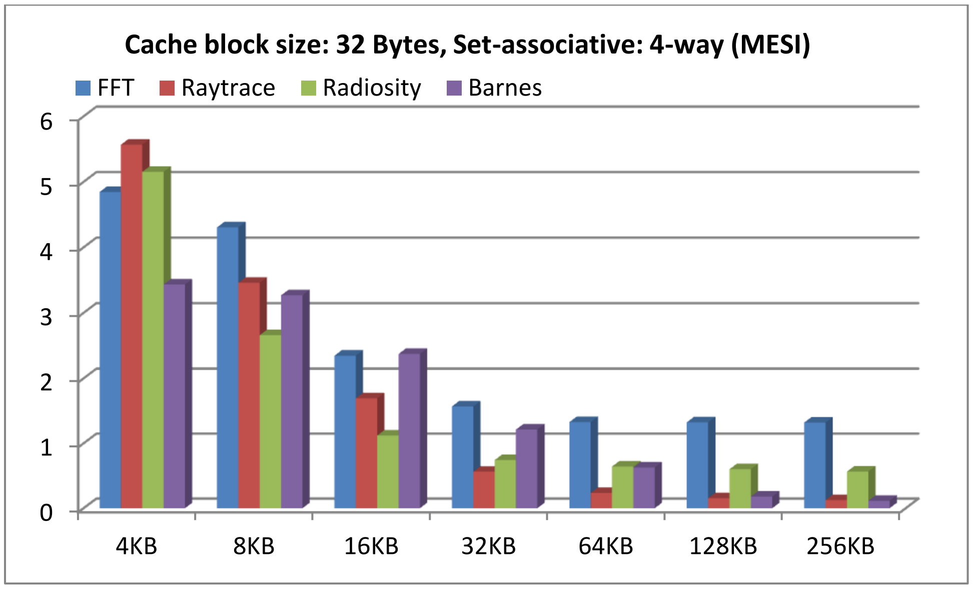


Figure 5.9: Miss Rate MESI (Block size 32 Bytes, Set-associative 4-Way, Core-8)

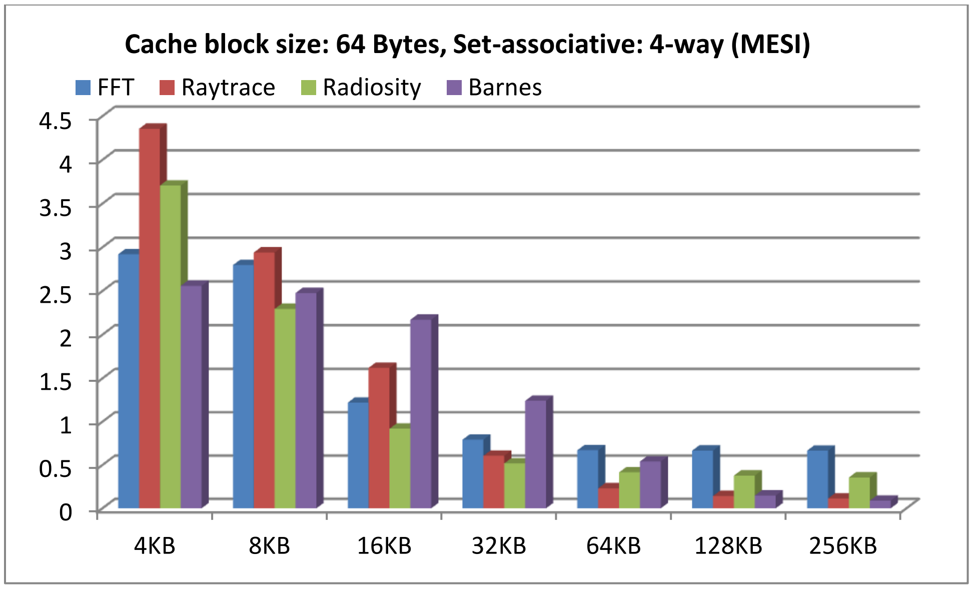


Figure 5.10: Miss Rate MESI (Block size 64 Bytes, Set-associative 4-Way, Core-8)

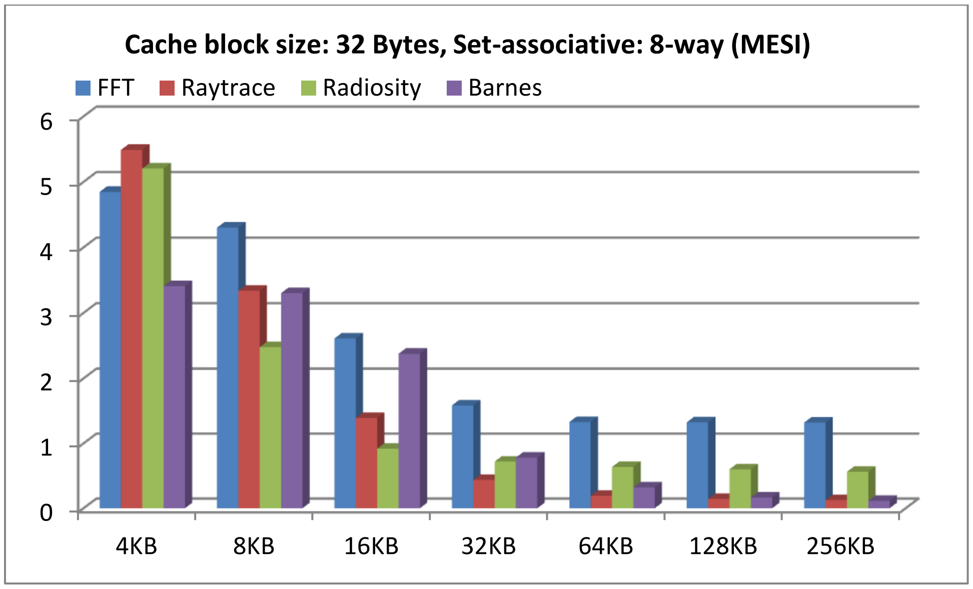


Figure 5.11: Miss Rate MESI (Block size 32 Bytes, Set-associative 8-Way, Core-8)

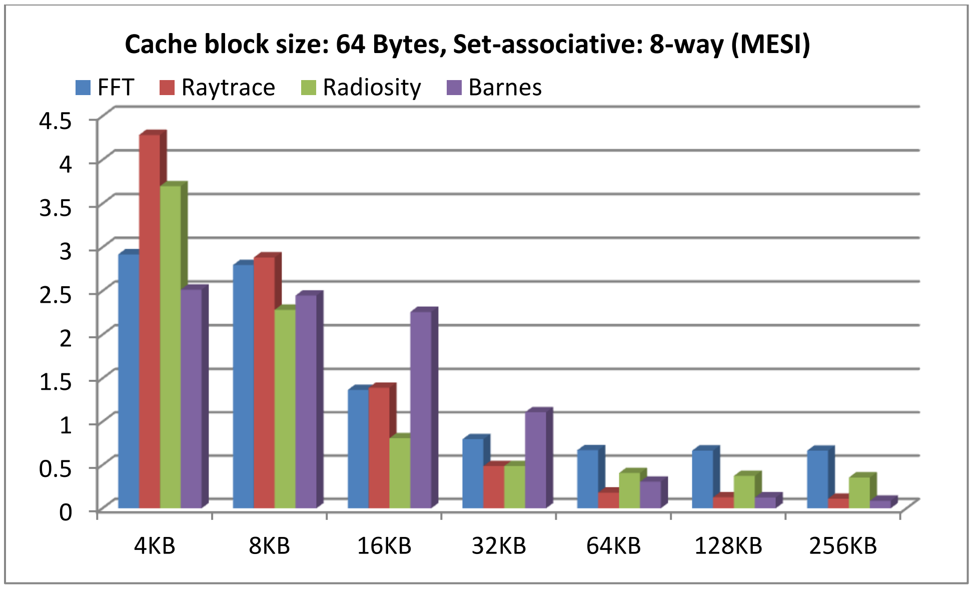


Figure 5.12: Miss Rate MESI (Block size 64 Bytes, Set-associative 8-Way, Core-8)

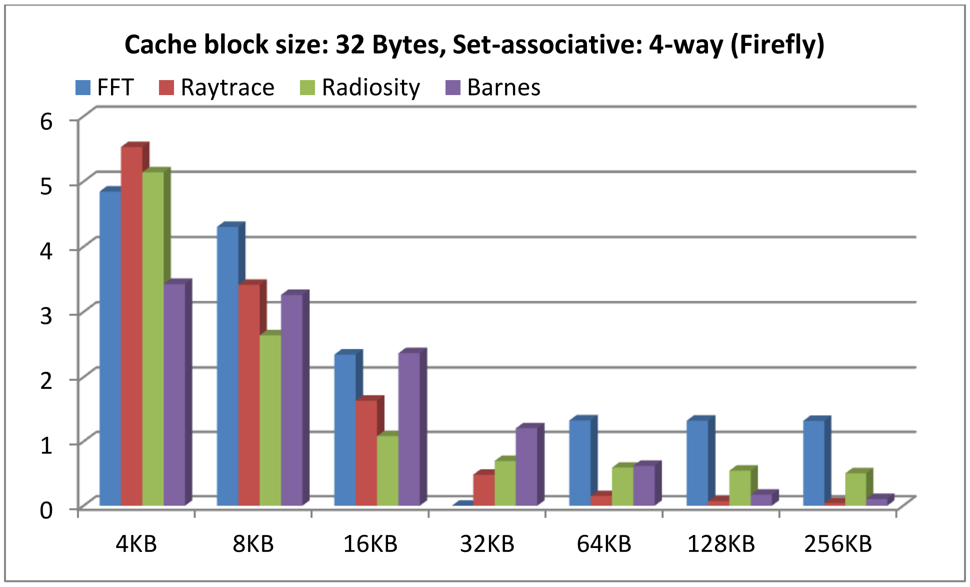


Figure 5.13: Miss Rate FIRELY (Block size 32 Bytes, Set-associative 4-Way, Core-8)

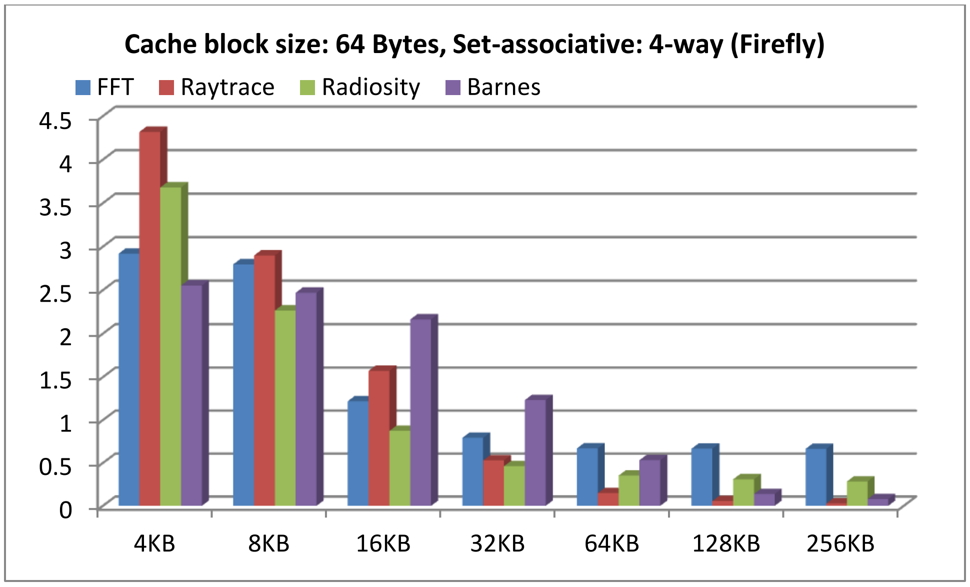


Figure 5.14: Miss Rate FIRELY (Block size 64 Bytes, Set-associative 4-Way, Core-8)

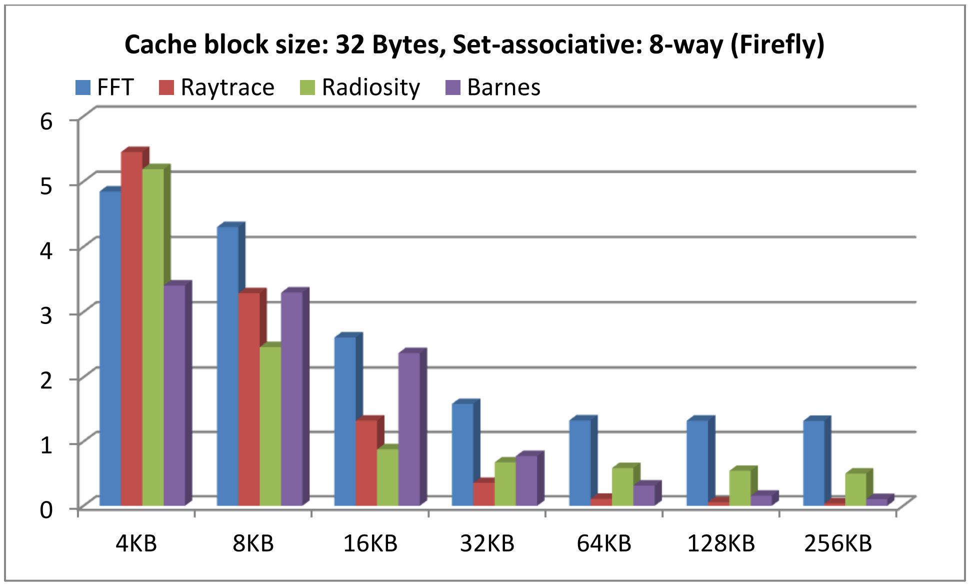


Figure 5.15: Miss Rate FIRELY (Block size 32 Bytes, Set-associative 8-Way, Core-8)

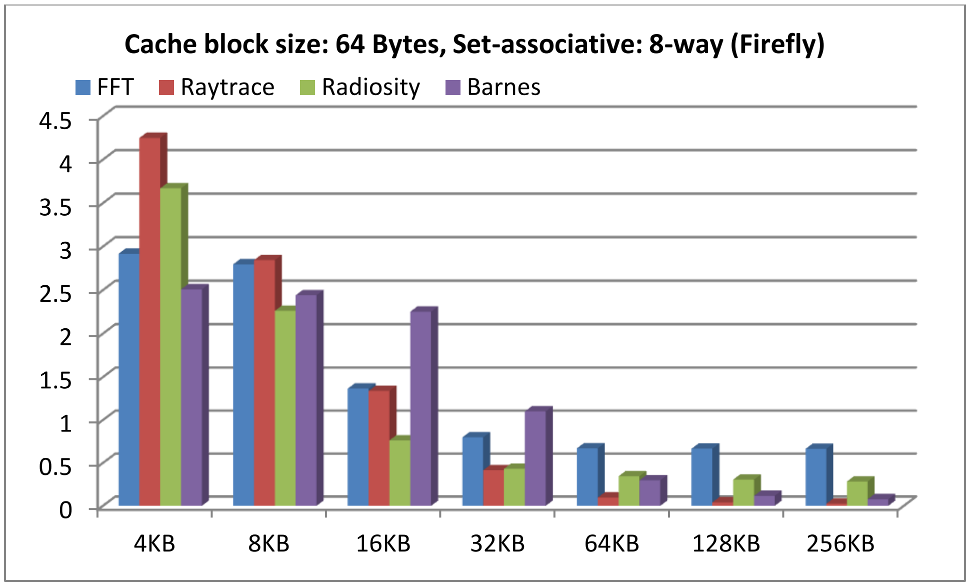


Figure 5.16: Miss Rate FIRELY (Block size 64 Bytes, Set-associative 8-Way, Core-8)

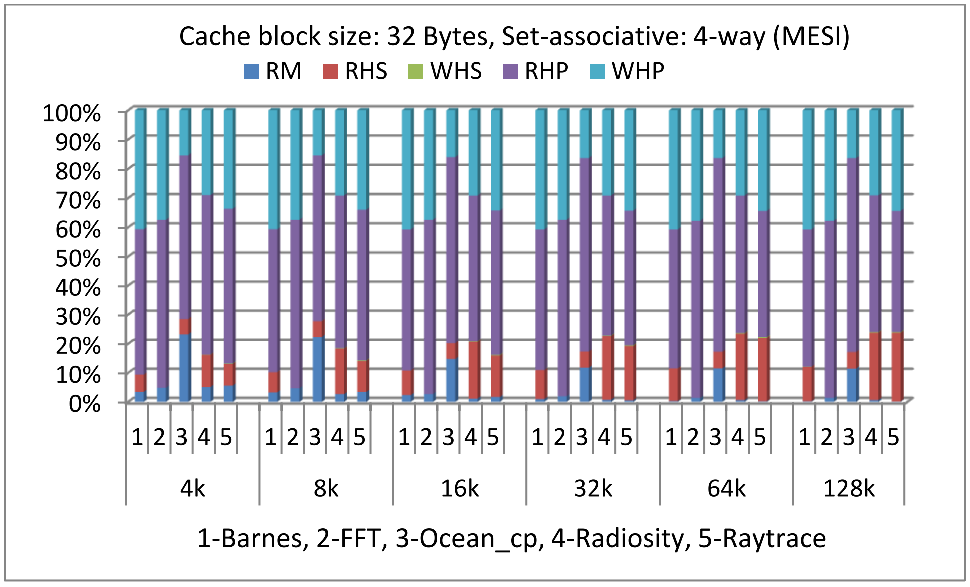


Figure 5.17: Compare Miss Rate Percentage MESI (Block size 32 Bytes, Set-associative 4-Way, Core-4)

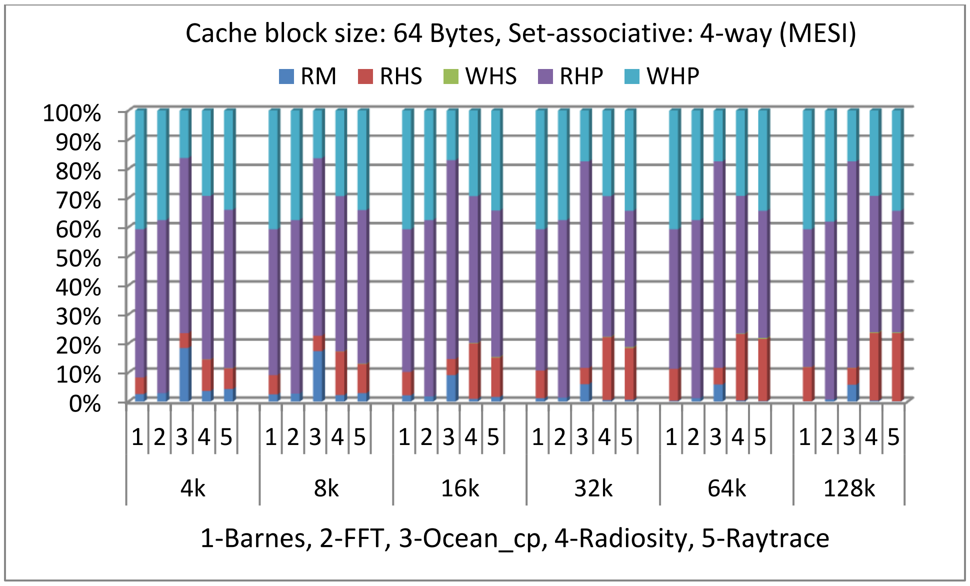


Figure 5.18: Compare Miss Rate Percentage MESI (Block size 64 Bytes, Set-associative 4-Way, Core-4)

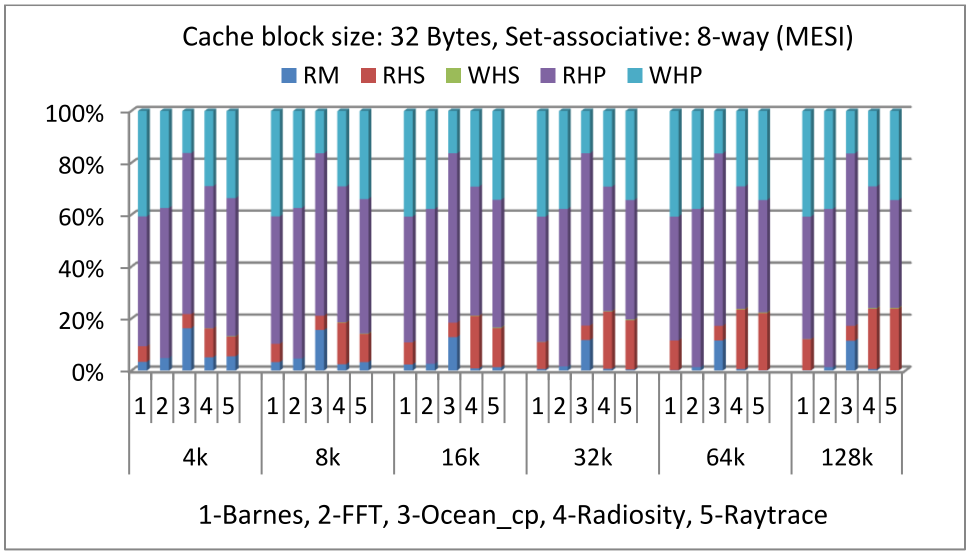


Figure 5.19: Compare Miss Rate Percentage MESI (Block size 32 Bytes, Set-associative 8-Way, Core-4)

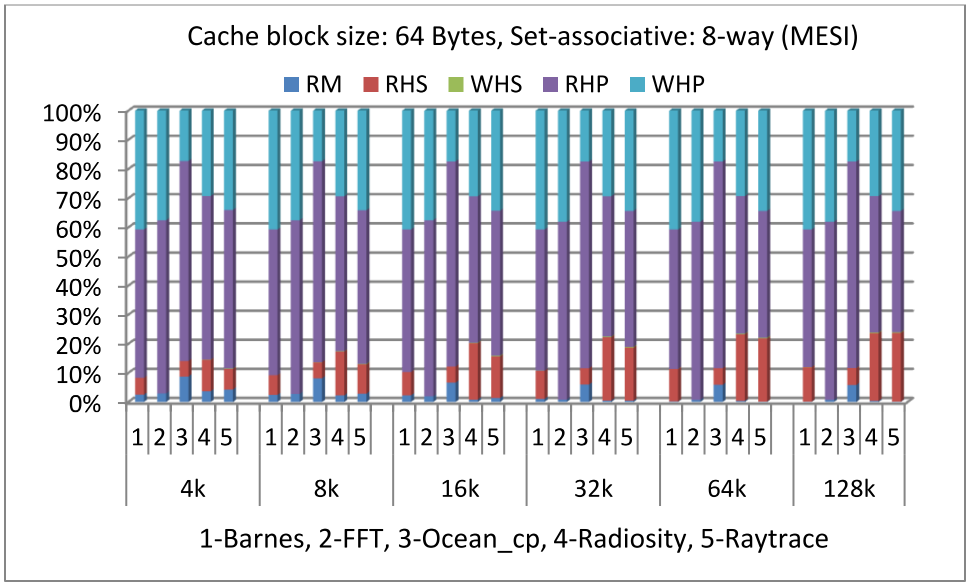


Figure 5.20: Compare Miss Rate Percentage MESI (Block size 64 Bytes, Set-associative 8-Way, Core-4)

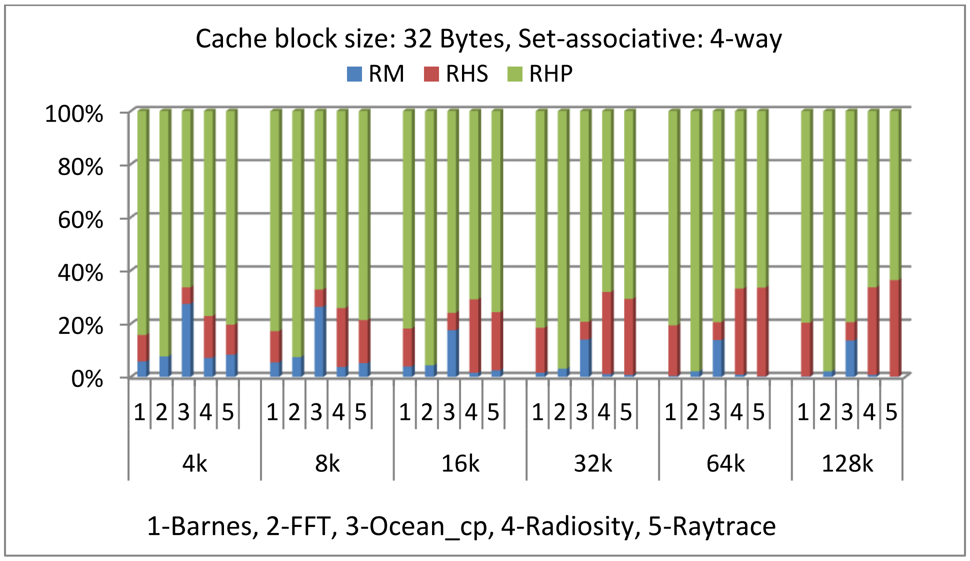


Figure 5.21: Compare Miss Rate Percentage FIREFLY (Block size 32 Bytes, Set-associative 4-Way, Core-4)

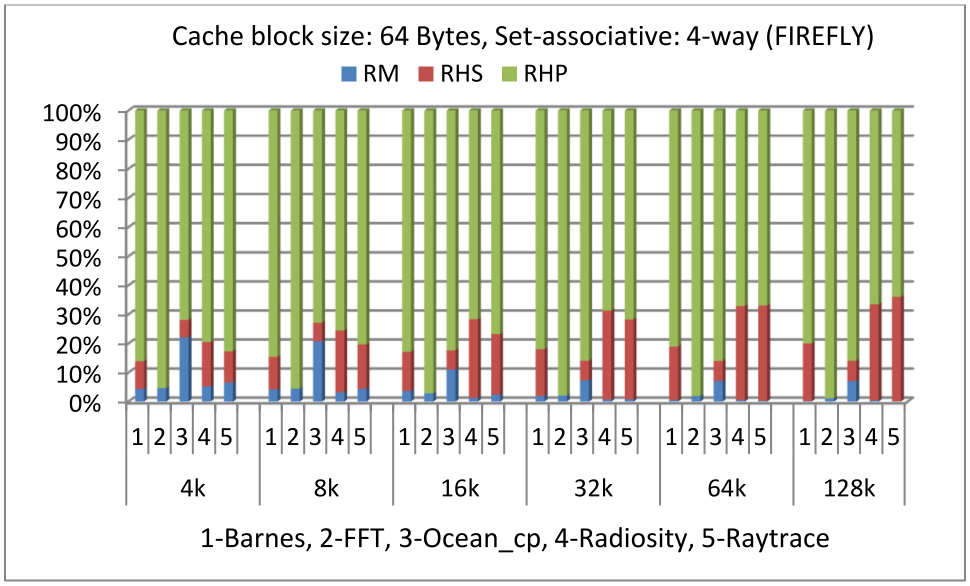


Figure 5.22: Compare Miss Rate Percentage FIREFLY (Block size 64 Bytes, Set-associative 4-Way, Core-4)

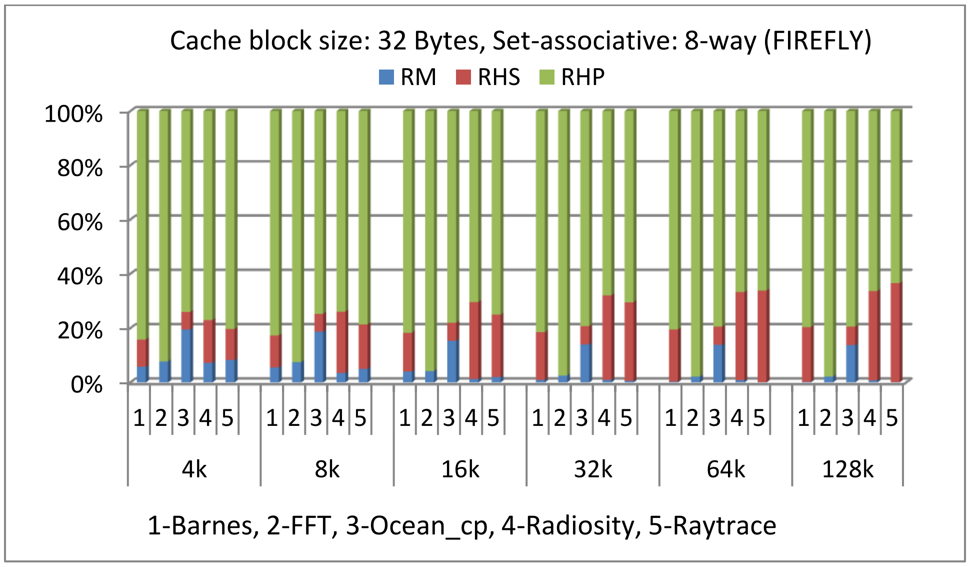


Figure 5.23: Compare Miss Rate Percentage FIREFLY (Block size 32 Bytes, Set-associative 8-Way, Core-4)

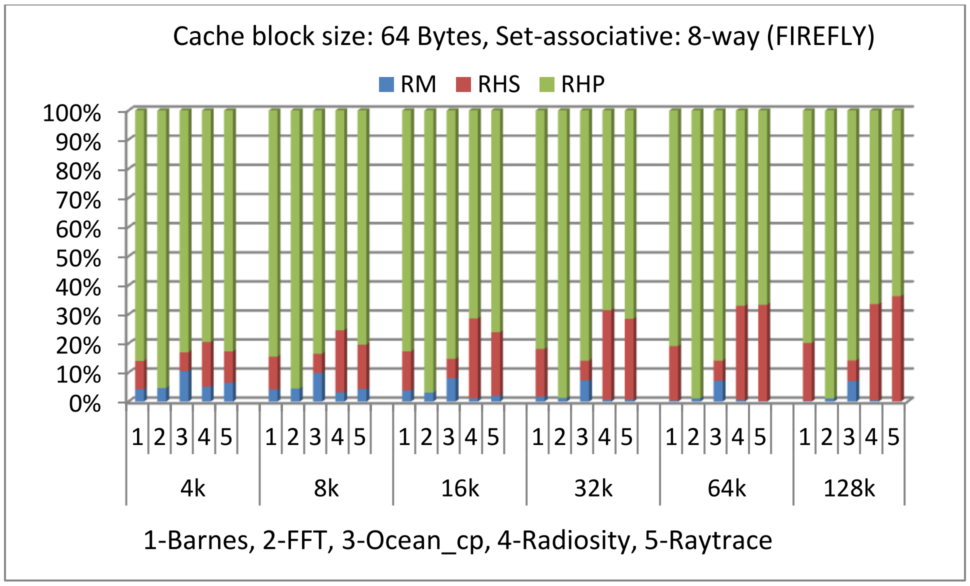


Figure 5.24: Compare Miss Rate Percentage FIREFLY (Block size 64 Bytes, Set-associative 8-Way, Core-4)

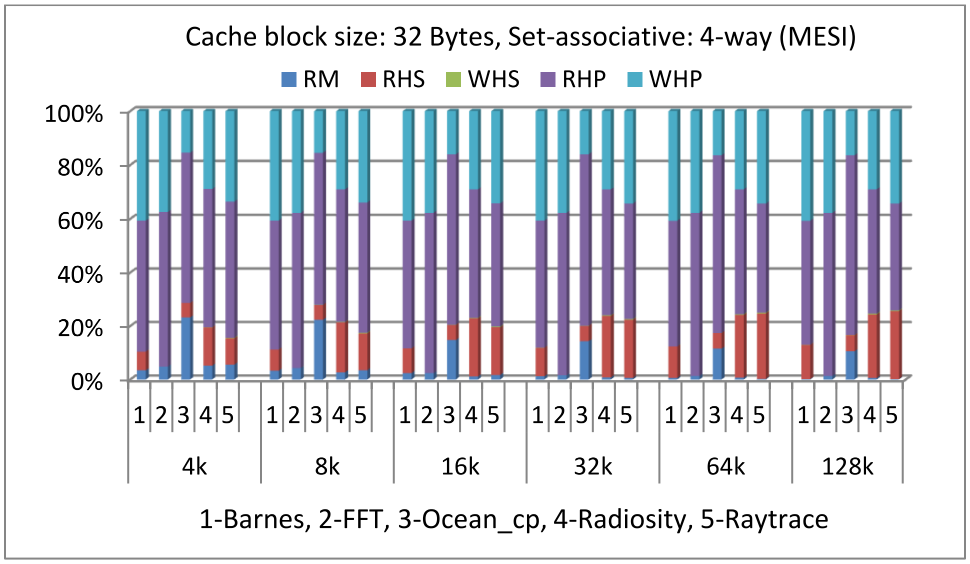


Figure 5.25: Compare Miss Rate Percentage MESI (Block size 32 Bytes, Set-associative 4-Way, Core-8)

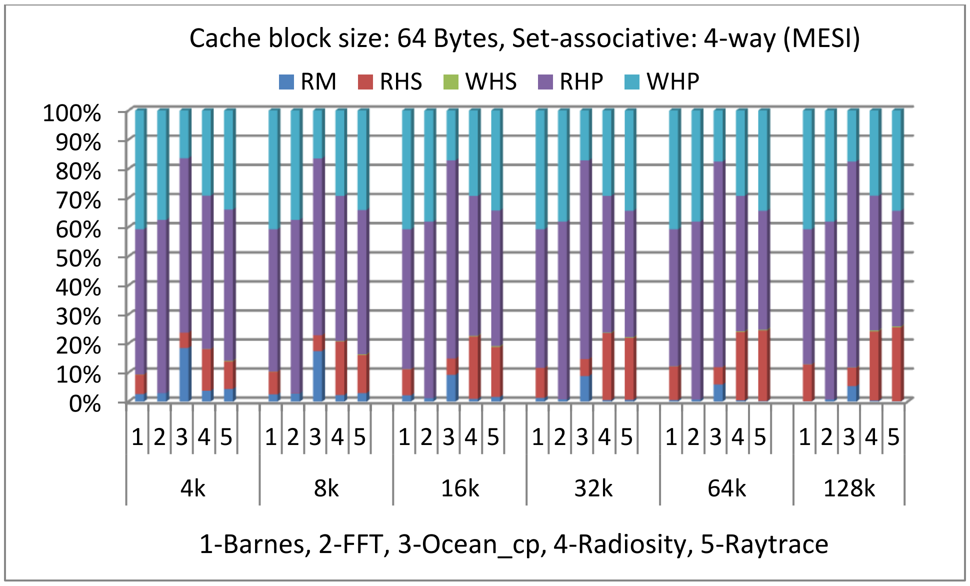


Figure 5.26: Compare Miss Rate Percentage MESI (Block size 64 Bytes, Set-associative 4-Way, Core-8)

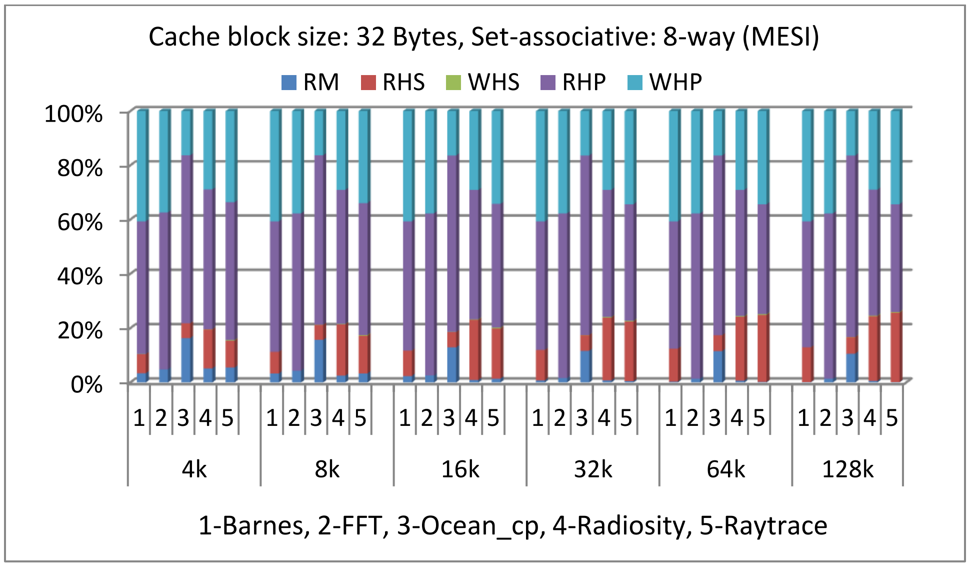


Figure 5.27: Compare Miss Rate Percentage MESI (Block size 32 Bytes, Set-associative 8-Way, Core-8)

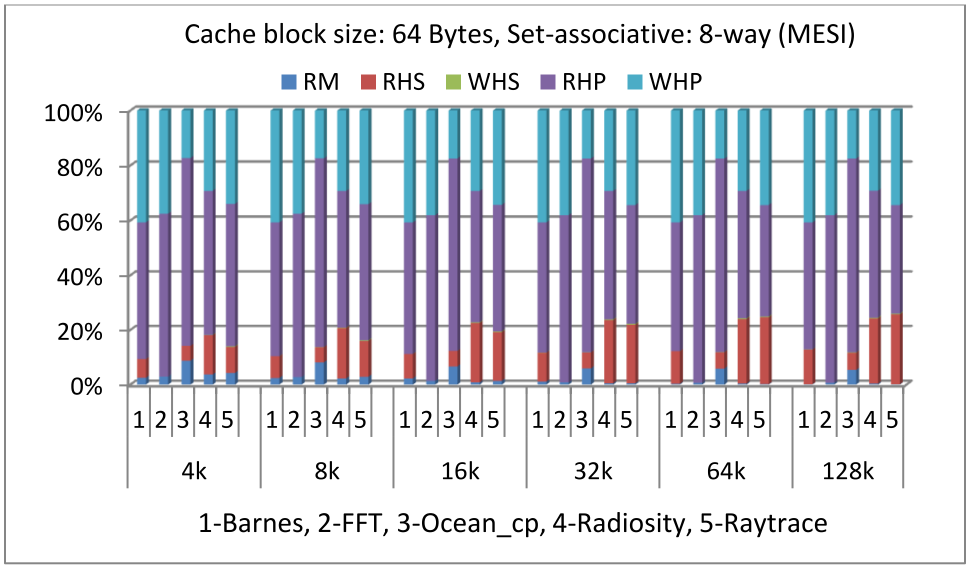


Figure 5.28: Compare Miss Rate Percentage MESI (Block size 64 Bytes, Set-associative 8-Way, Core-8)

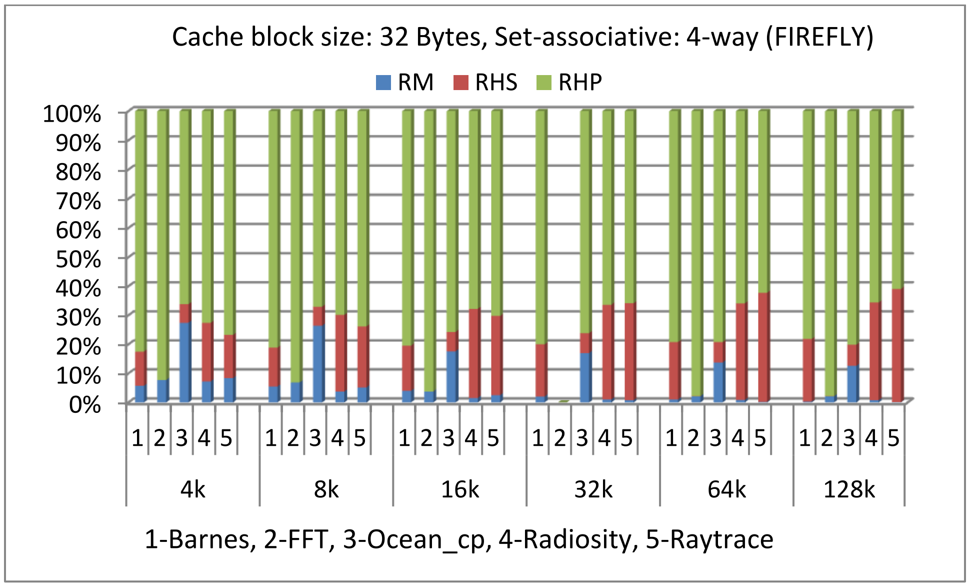


Figure 5.29: Compare Miss Rate Percentage FIREFLY (Block size 32 Bytes, Set-associative 4-Way, Core-8)

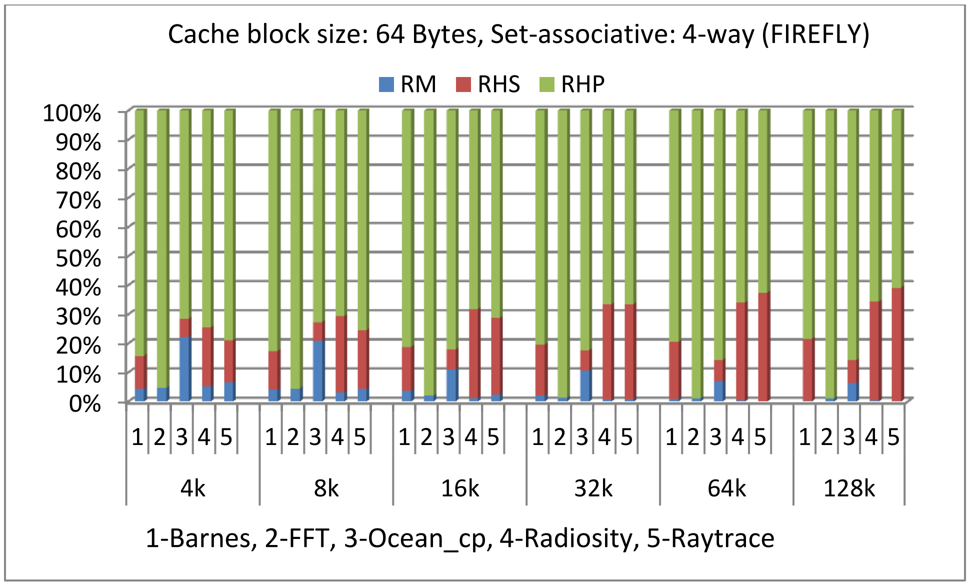


Figure 5.30: Compare Miss Rate Percentage FIREFLY (Block size 64 Bytes, Set-associative 4-Way, Core-8)

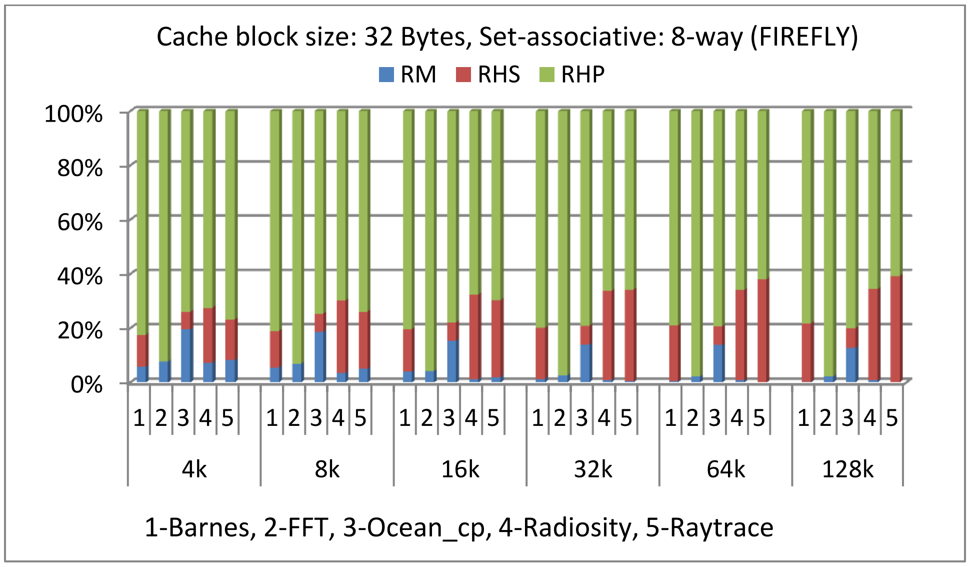


Figure 5.31: Compare Miss Rate Percentage FIREFLY (Block size 32 Bytes, Set-associative 8-Way, Core-8)

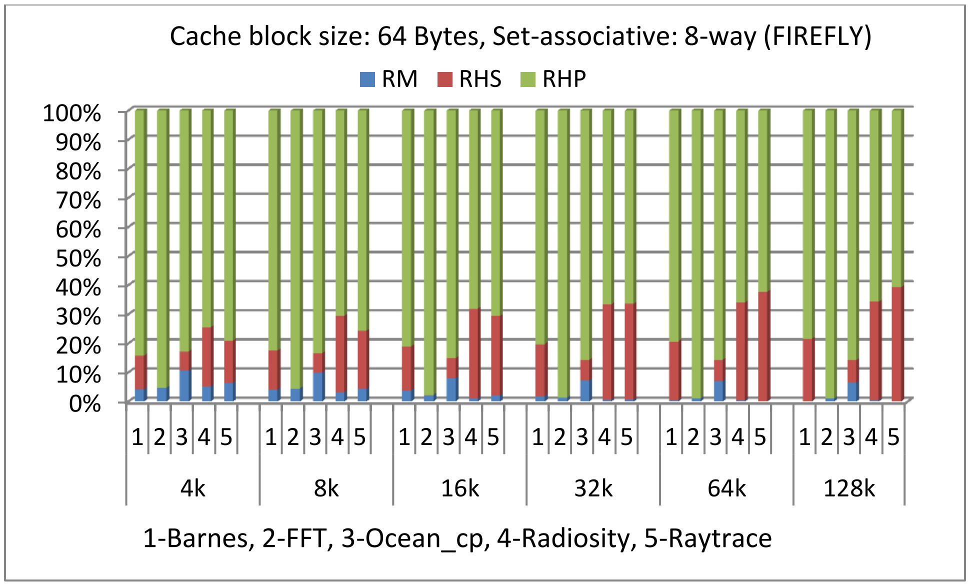


Figure 5.32: Compare Miss Rate Percentage FIREFLY (Block size 64 Bytes, Set-associative 8-Way, Core-8)

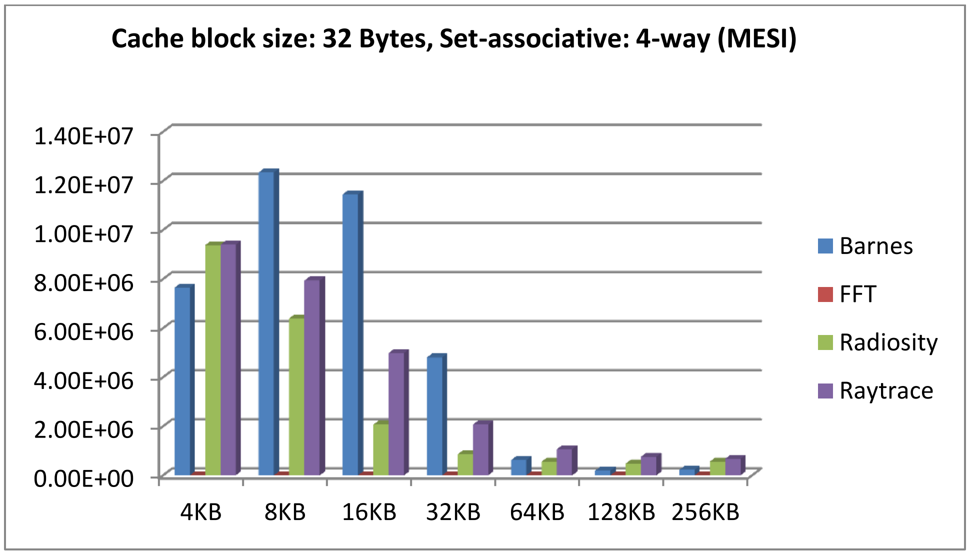


Figure 5.33: # Data Transfer between Caches MESI (Block size 32 Bytes, Set-associative 4-Way, Core-4)

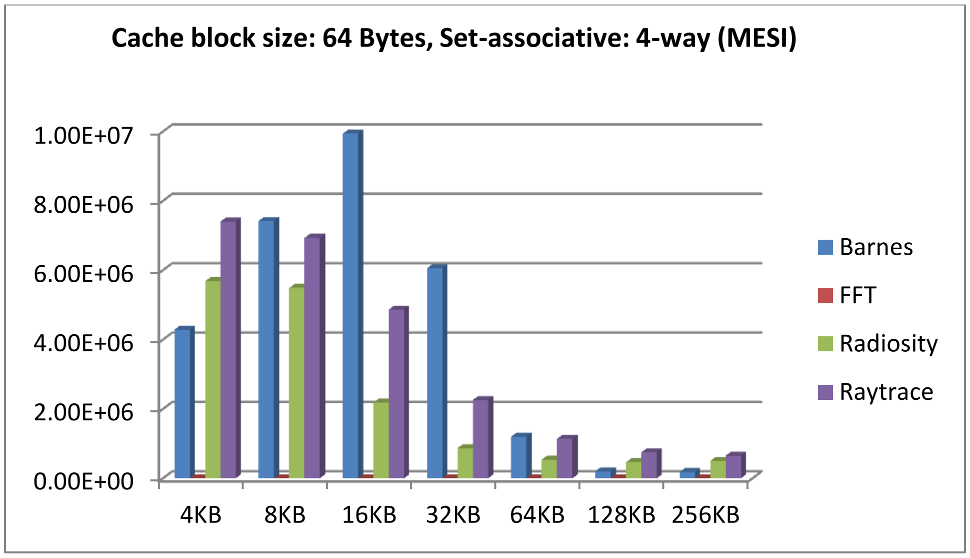


Figure 5.34: # Data Transfer between Caches MESI (Block size 64 Bytes, Set-associative 4-Way, Core-4)

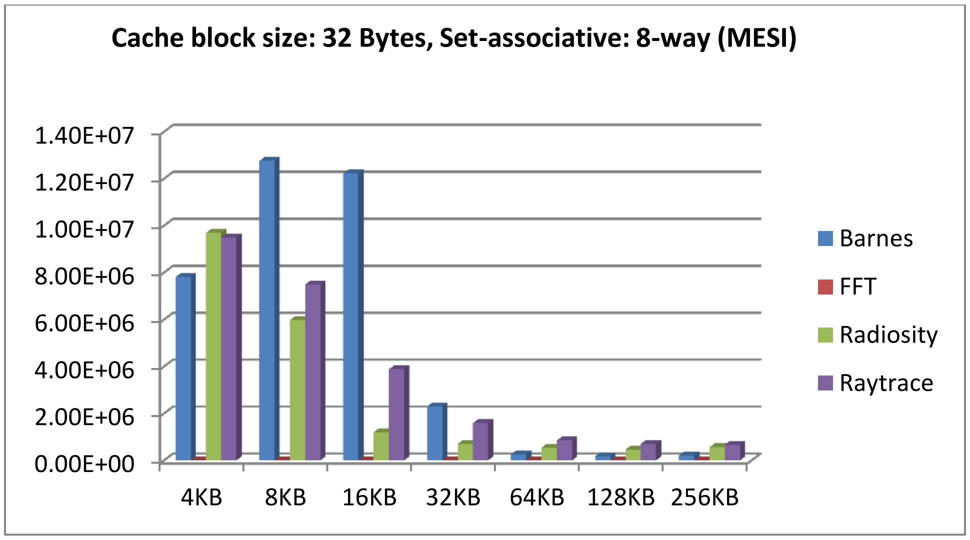


Figure 5.35: # Data Transfer between Caches MESI (Block size 32 Bytes, Set-associative 8-Way, Core-4)

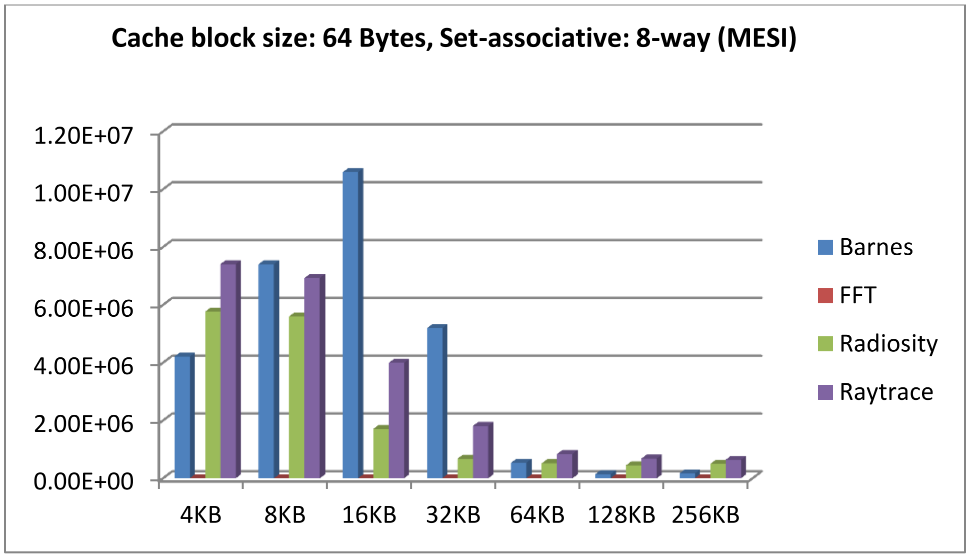


Figure 5.36: # Data Transfer between Caches MESI (Block size 64 Bytes, Set-associative 8-Way, Core-4)

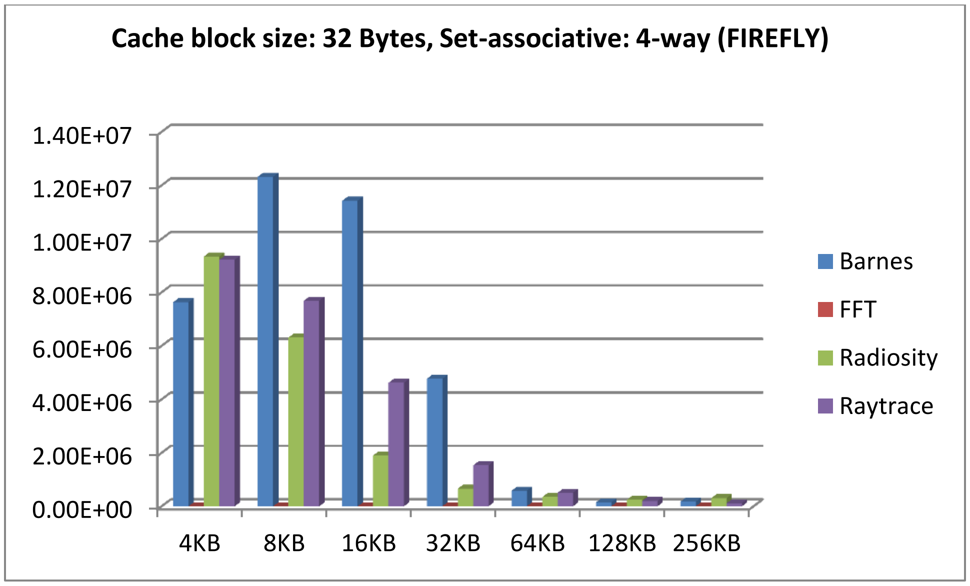


Figure 5.37: # Data Transfer between Caches FIREFLY (Block size 32 Bytes, Set-associative 4-Way, Core-4)

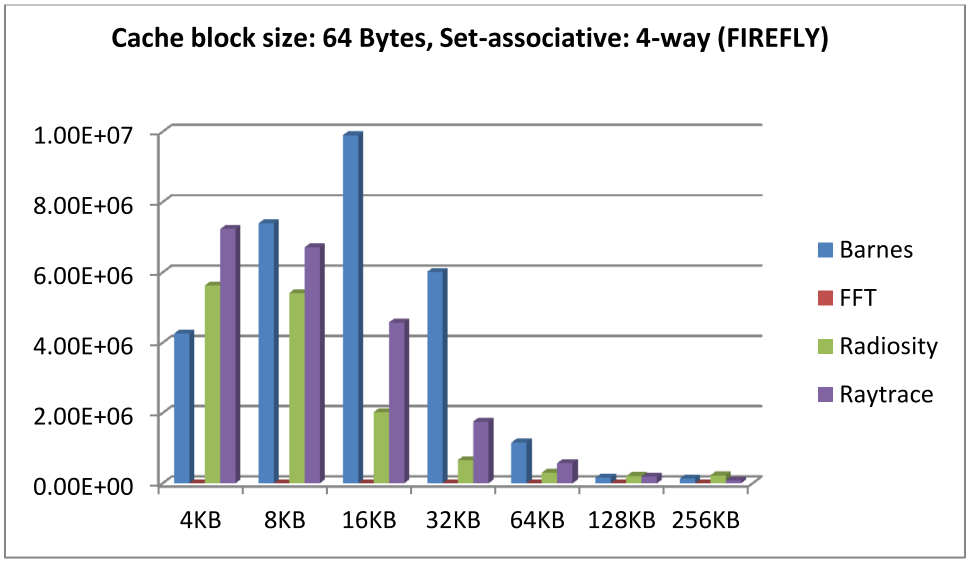


Figure 5.38: # Data Transfer between Caches FIREFLY (Block size 64 Bytes, Set-associative 4-Way, Core-4)

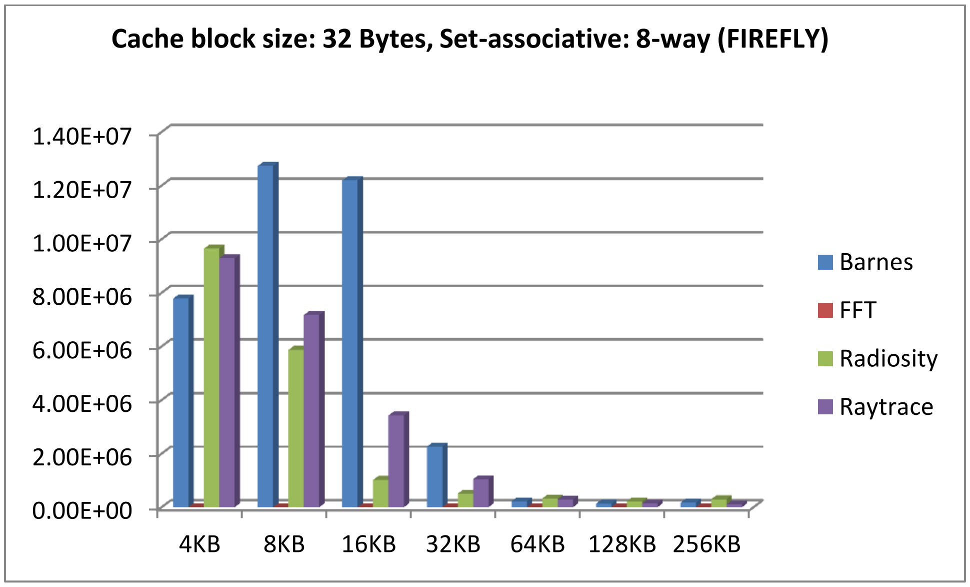


Figure 5.39: # Data Transfer between Caches FIREFLY (Block size 32 Bytes, Set-associative 8-Way, Core-4)

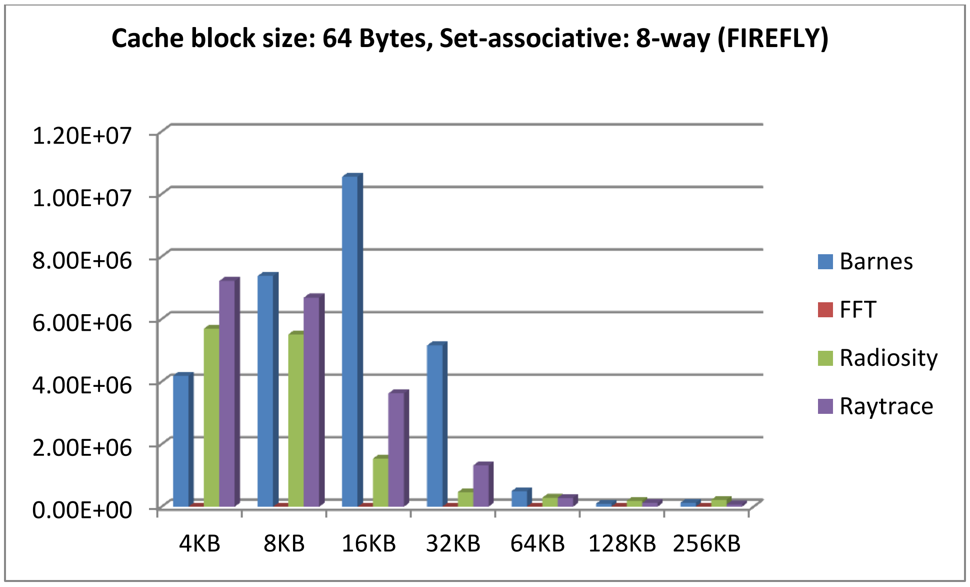


Figure 5.40: # Data Transfer between Caches FIREFLY (Block size 64 Bytes, Set-associative 8-Way, Core-4)

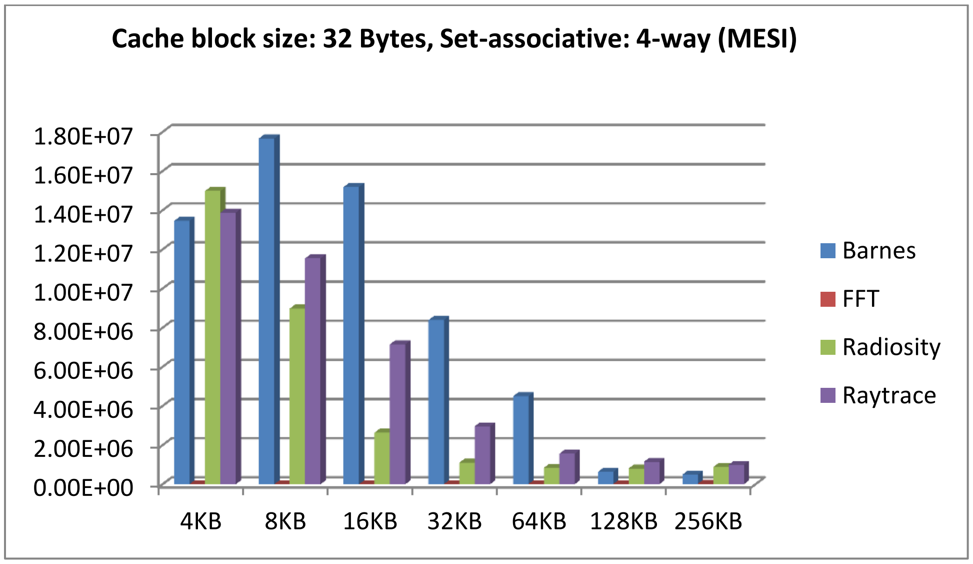


Figure 5.41: # Data Transfer between Caches MESI (Block size 32 Bytes, Set-associative 4-Way, Core-8)

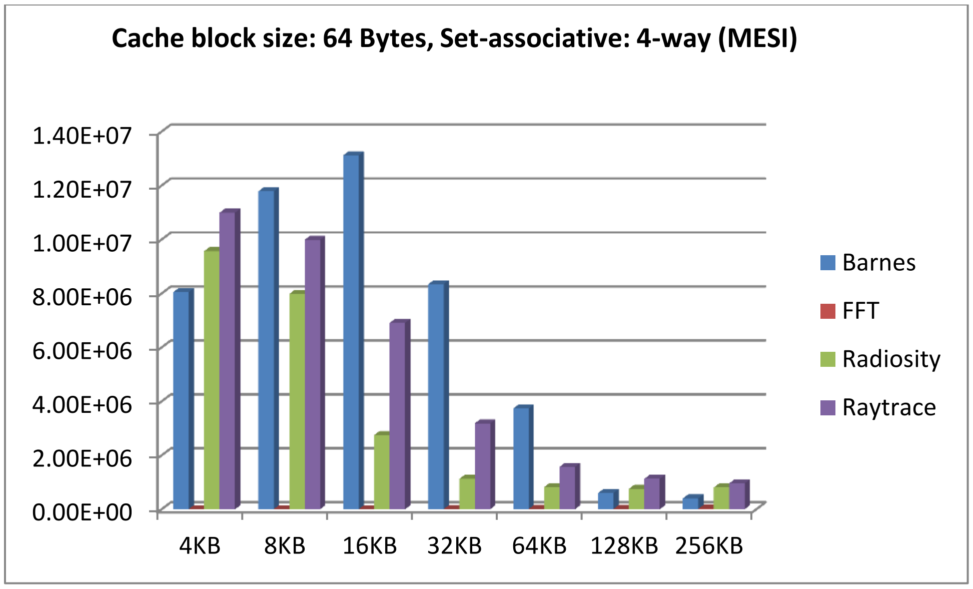


Figure 5.42: # Data Transfer between Caches MESI (Block size 64 Bytes, Set-associative 4-Way, Core-8)

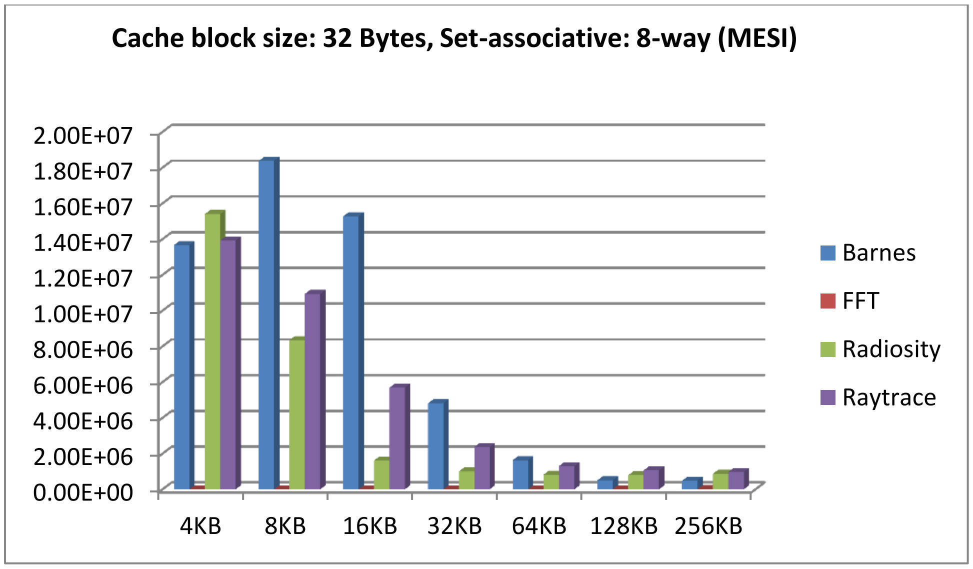


Figure 5.43: # Data Transfer between Caches MESI (Block size 32 Bytes, Set-associative 8-Way, Core-8)

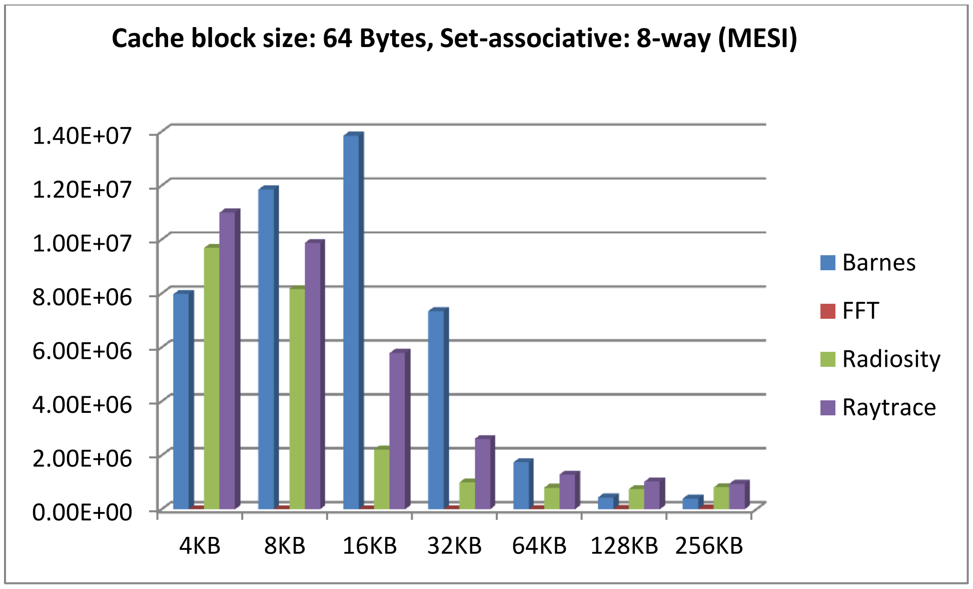


Figure 5.44: # Data Transfer between Caches MESI (Block size 64 Bytes, Set-associative 8-Way, Core-8)

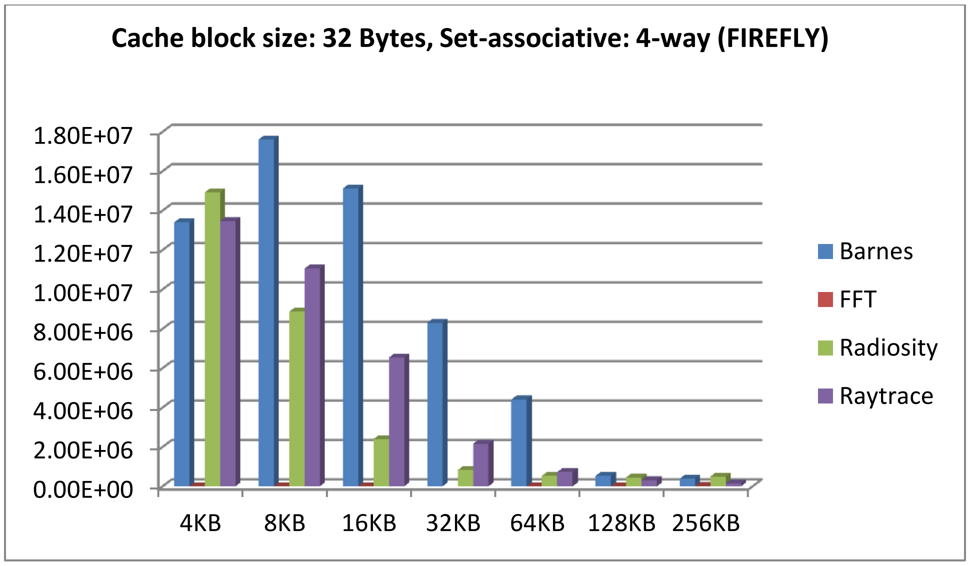


Figure 5.45: # Data Transfer between Caches FIREFLY (Block size 32 Bytes, Set-associative 4-Way, Core-8)

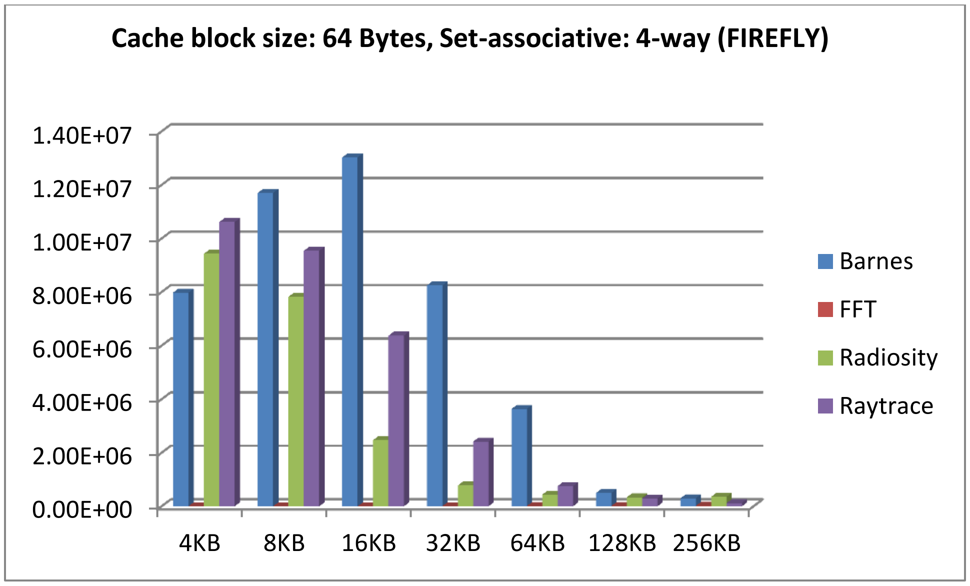


Figure 5.46: # Data Transfer between Caches FIREFLY (Block size 64 Bytes, Set-associative 4-Way, Core-8)

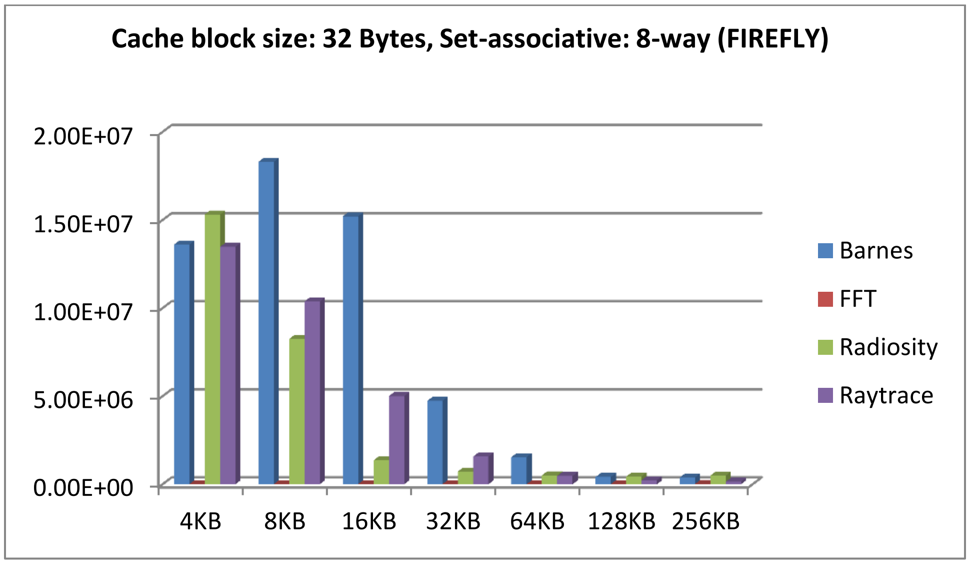


Figure 5.47: # Data Transfer between Caches FIREFLY (Block size 32 Bytes, Set-associative 8-Way, Core-8)

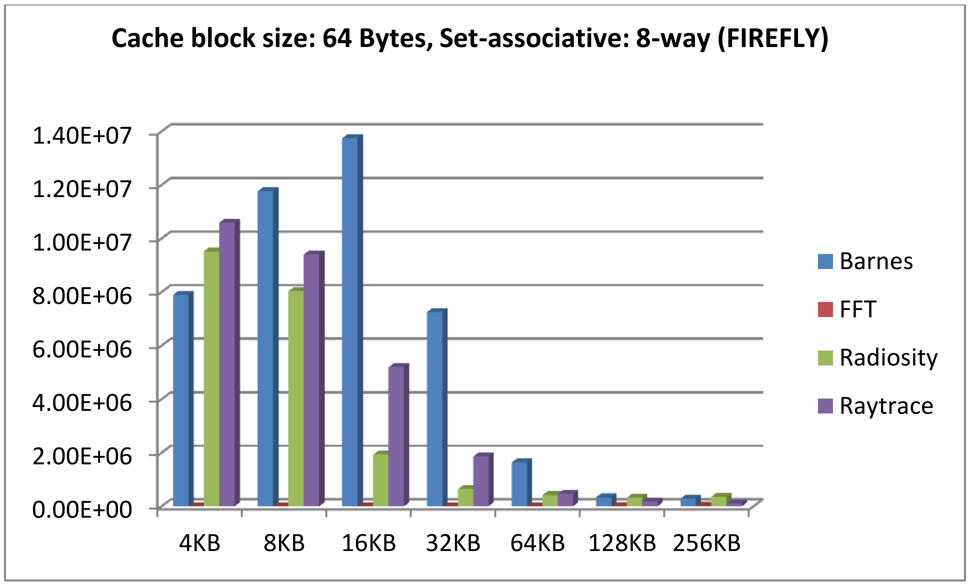


Figure 5.48: # Data Transfer between Caches FIREFLY (Block size 64 Bytes, Set-associative 8-Way, Core-8)

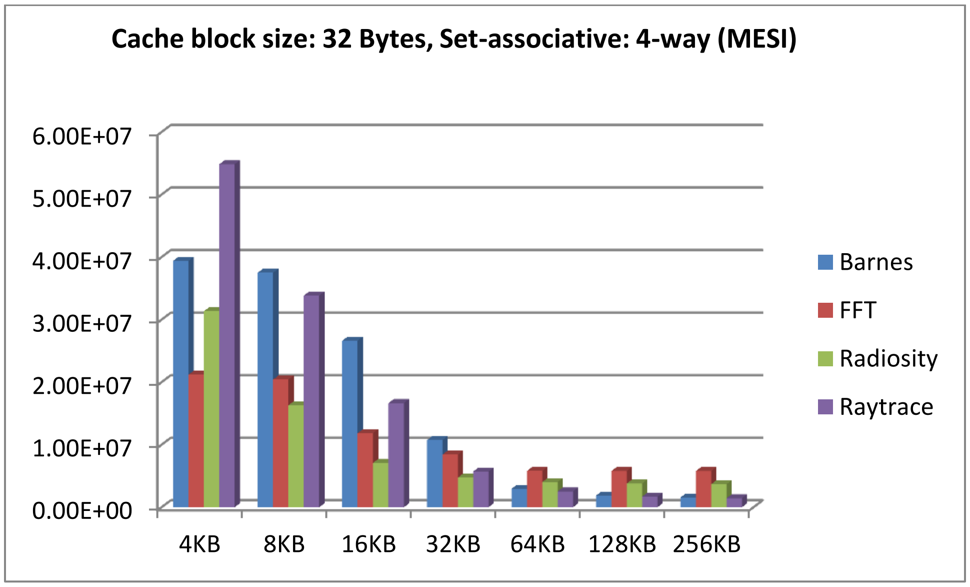


Figure 5.49: # Bus Traffic MESI (Block size 32 Bytes, Set-associative 4-Way, Core-4)

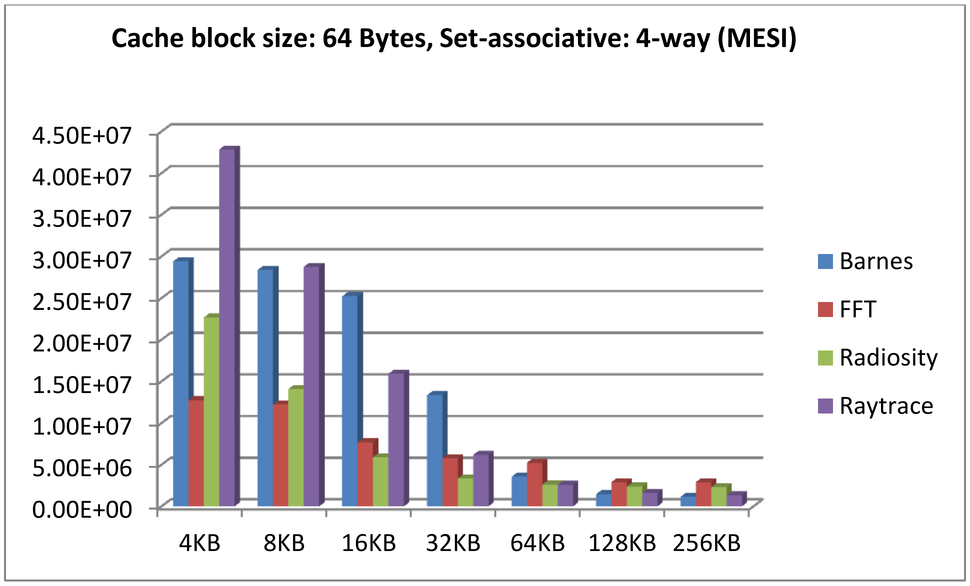


Figure 5.50: # Bus Traffic MESI (Block size 64 Bytes, Set-associative 4-Way, Core-4)

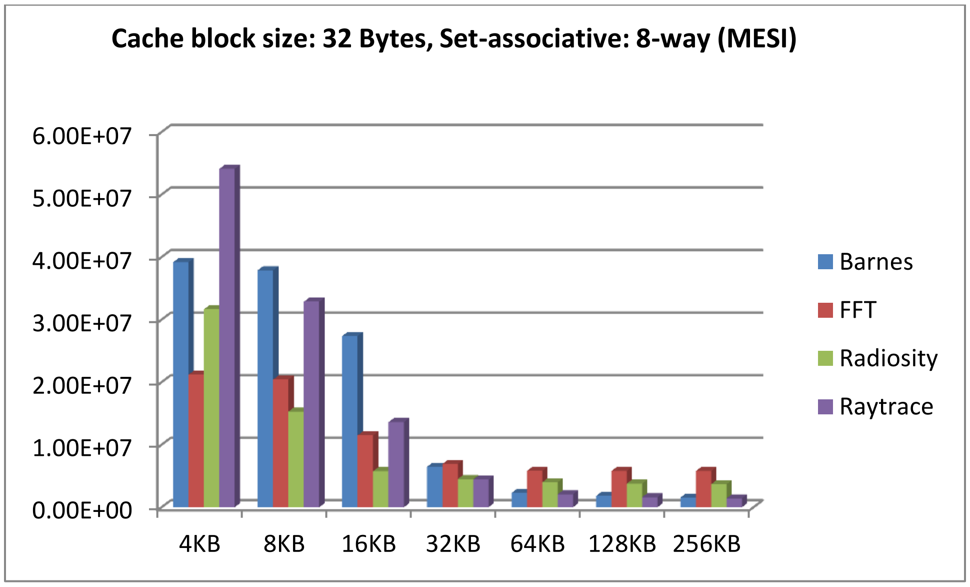


Figure 5.51: # Bus Traffic MESI (Block size 32 Bytes, Set-associative 8-Way, Core-4)

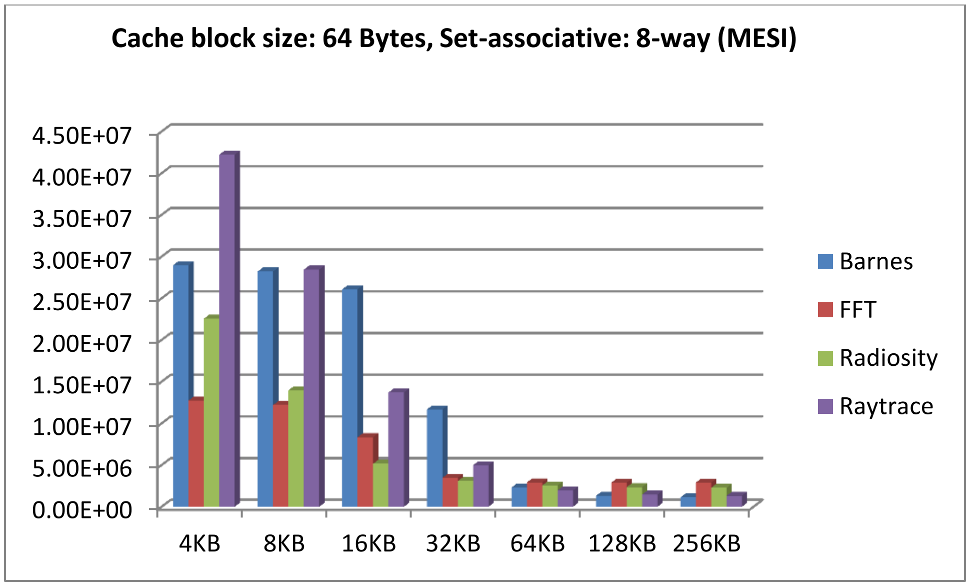


Figure 5.52: # Bus Traffic MESI (Block size 64 Bytes, Set-associative 8-Way, Core-4)

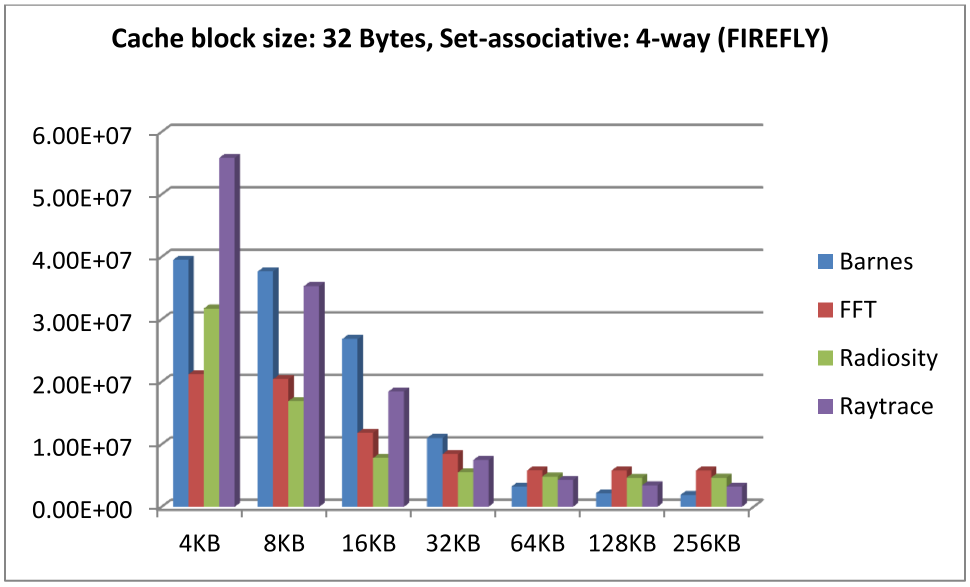


Figure 5.53: # Bus Traffic FIREFLY (Block size 32 Bytes, Set-associative 4-Way, Core-4)

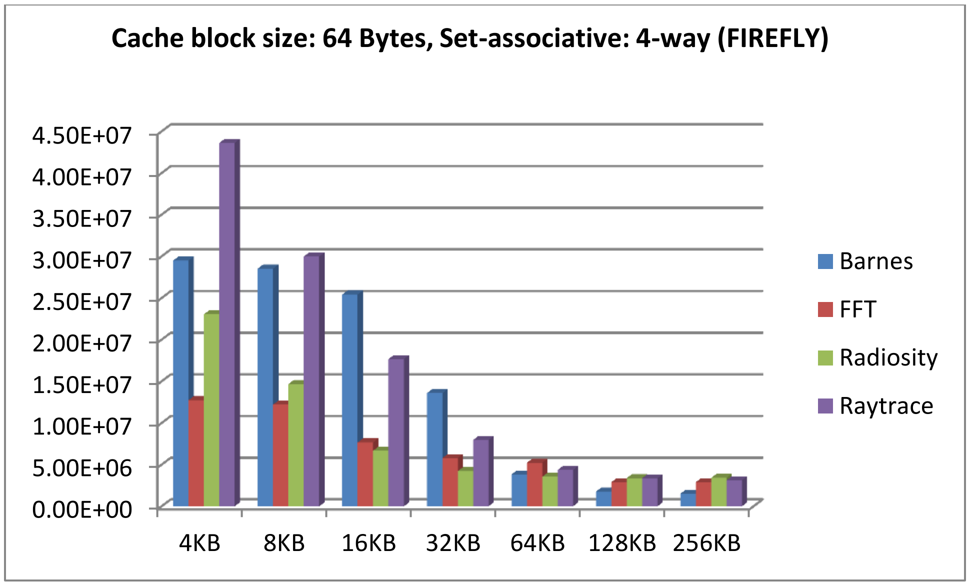


Figure 5.54: # Bus Traffic FIREFLY (Block size 64 Bytes, Set-associative 4-Way, Core-4)

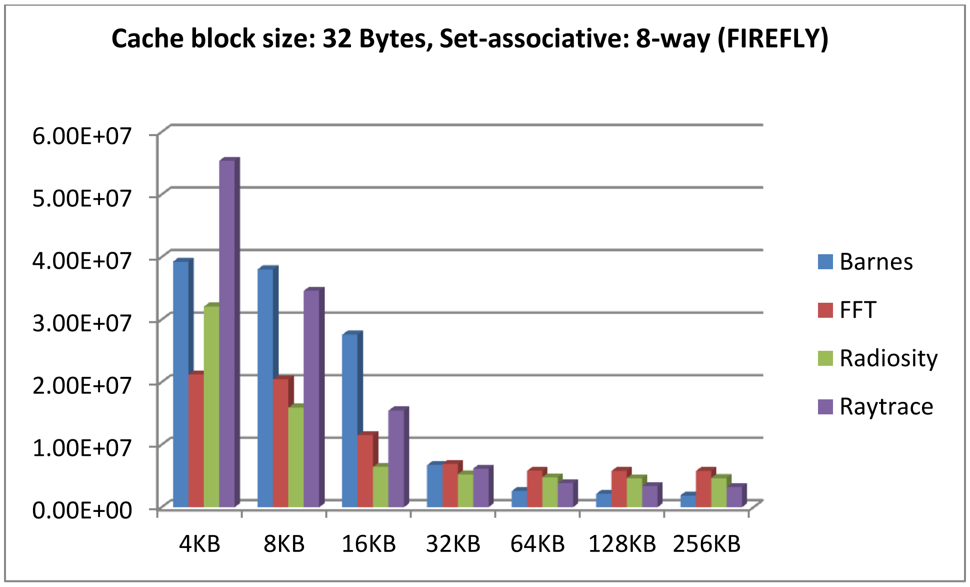


Figure 5.55: # Bus Traffic FIREFLY (Block size 32 Bytes, Set-associative 8-Way, Core-4)

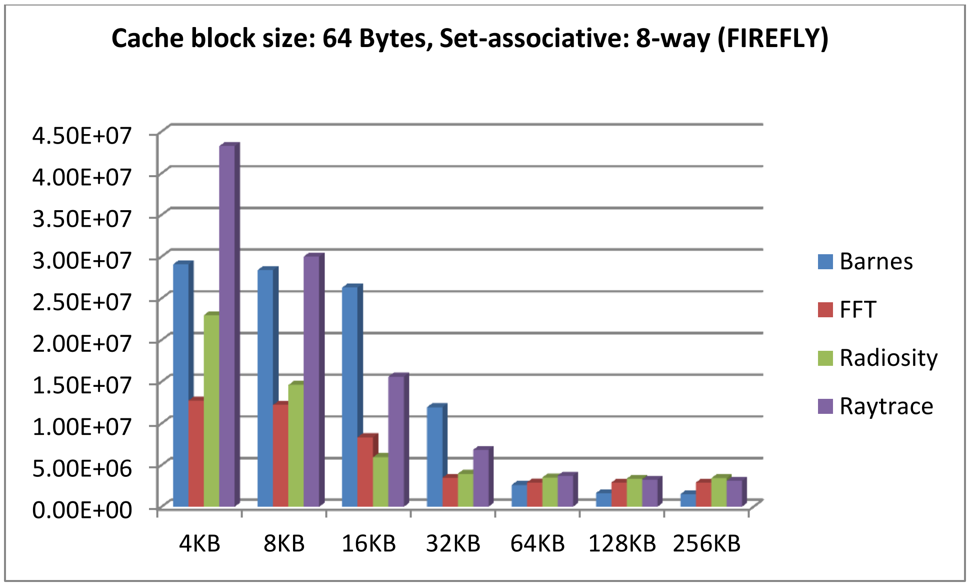


Figure 5.56: # Bus Traffic FIREFLY (Block size 64 Bytes, Set-associative 8-Way, Core-4)

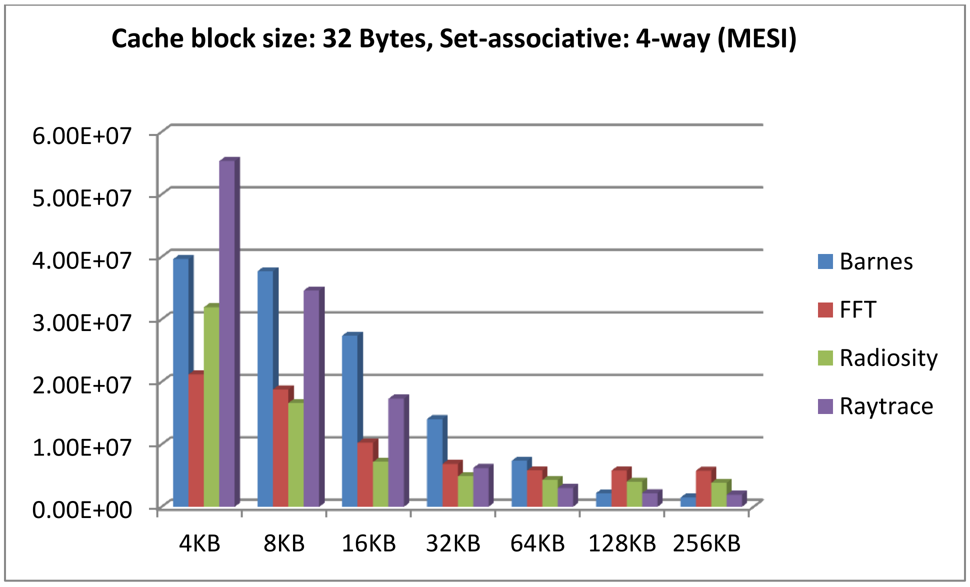


Figure 5.57: # Bus Traffic MESI (Block size 32 Bytes, Set-associative 4-Way, Core-8)

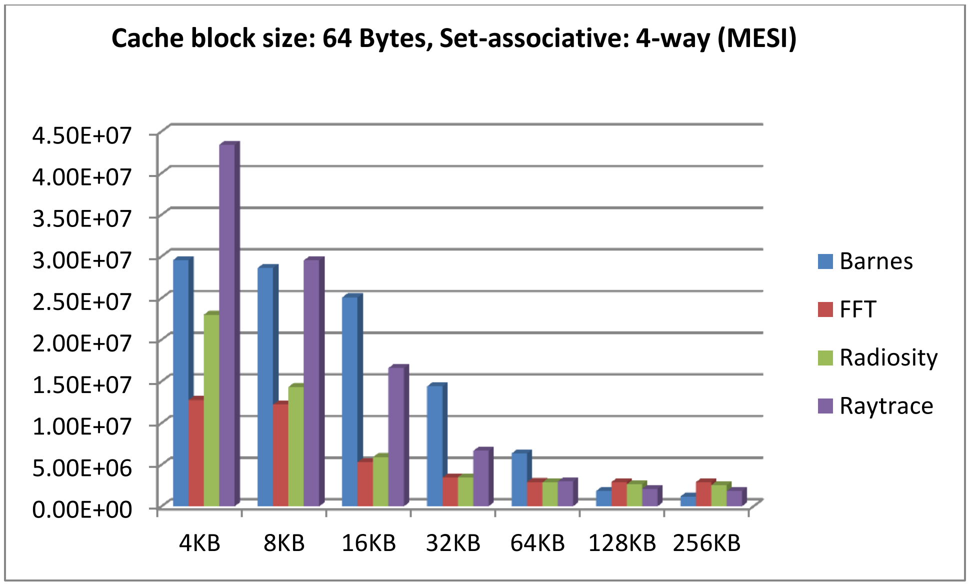


Figure 5.58: # Bus Traffic MESI (Block size 64 Bytes, Set-associative 4-Way, Core-8)

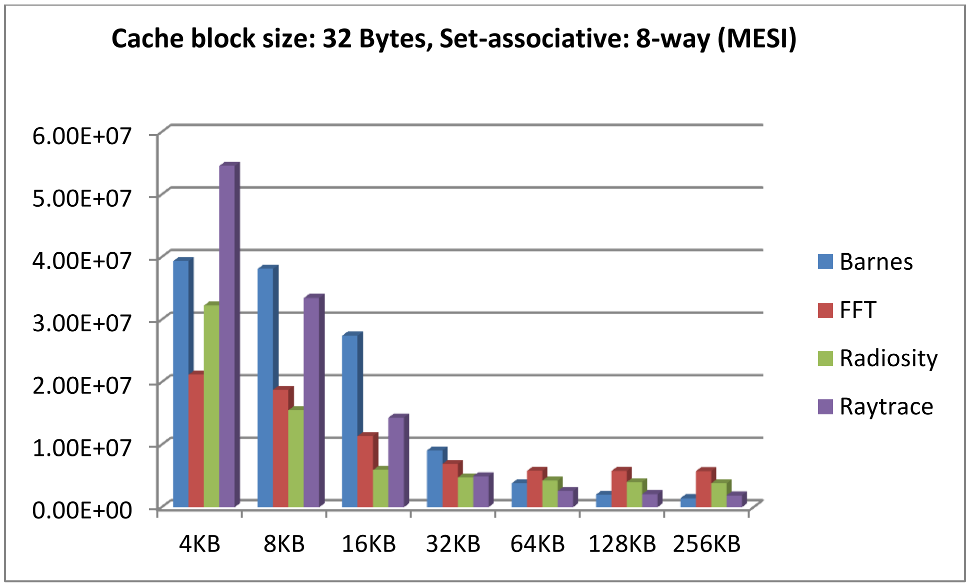


Figure 5.59: # Bus Traffic MESI (Block size 32 Bytes, Set-associative 8-Way, Core-8)

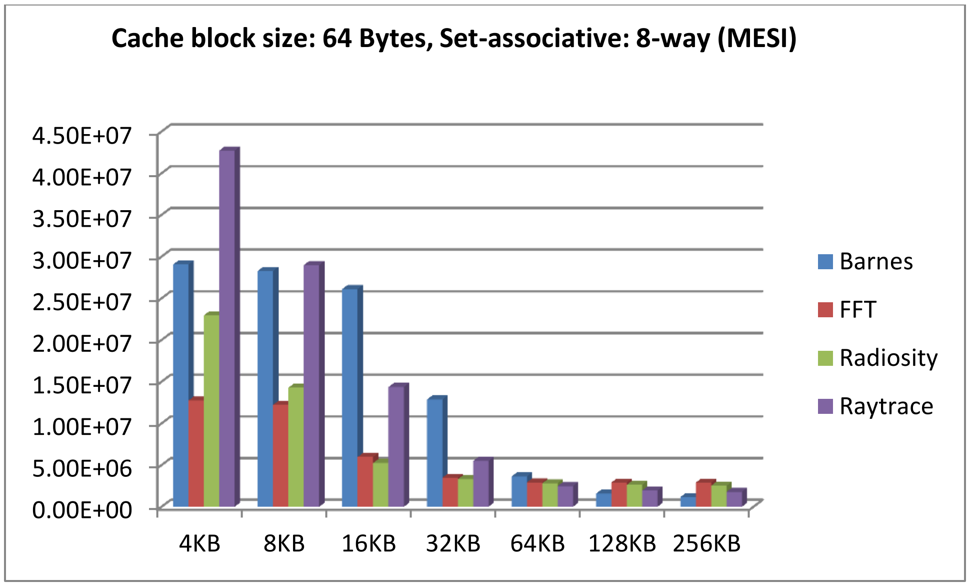


Figure 5.60: # Bus Traffic MESI (Block size 64 Bytes, Set-associative 8-Way, Core-8)

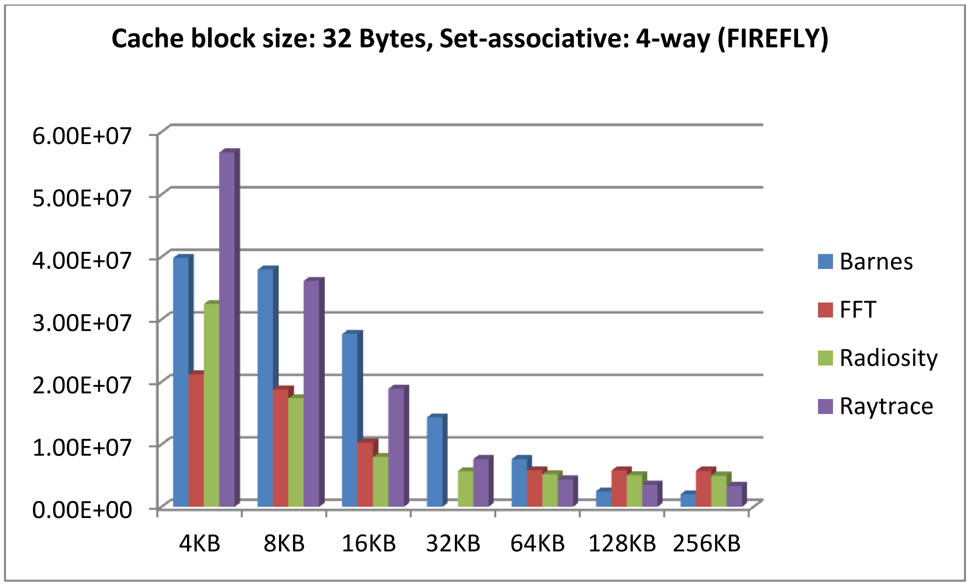


Figure 5.61: # Bus Traffic FIREFLY (Block size 32 Bytes, Set-associative 4-Way, Core-8)

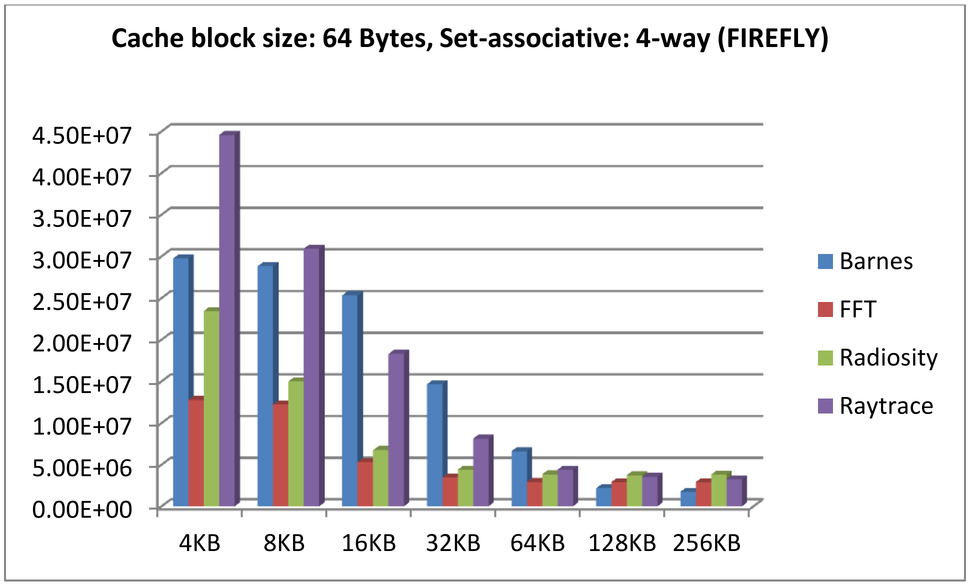


Figure 5.62: # Bus Traffic FIREFLY (Block size 64 Bytes, Set-associative 4-Way, Core-8)

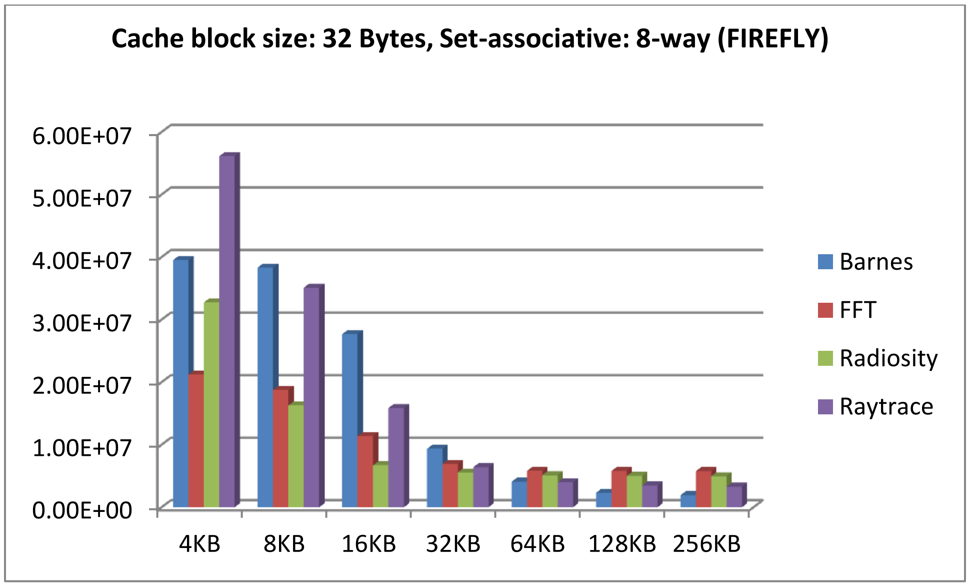


Figure 5.63: # Bus Traffic FIREFLY (Block size 32 Bytes, Set-associative 8-Way, Core-8)

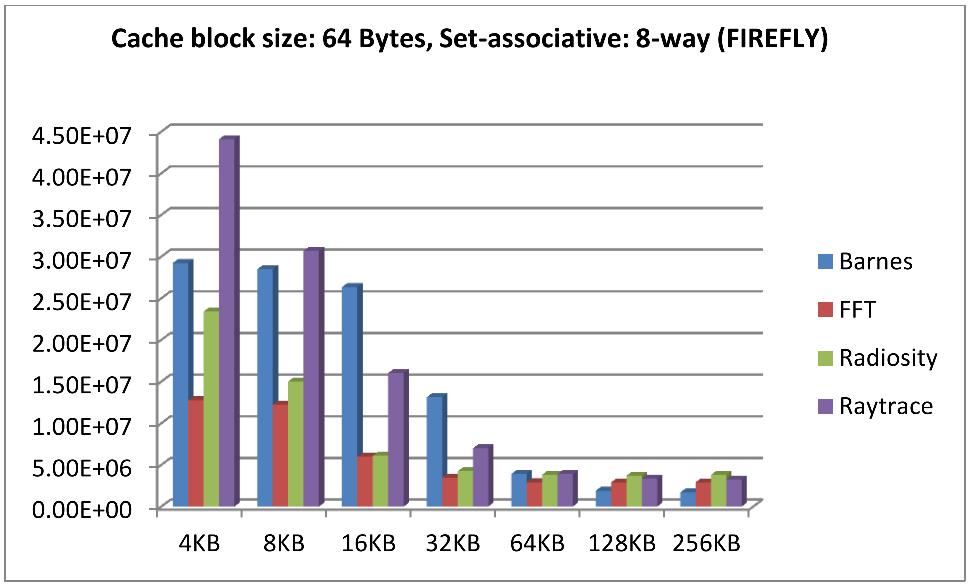


Figure 5.64: # Bus Traffic FIREFLY (Block size 64 Bytes, Set-associative 8-Way, Core-8)

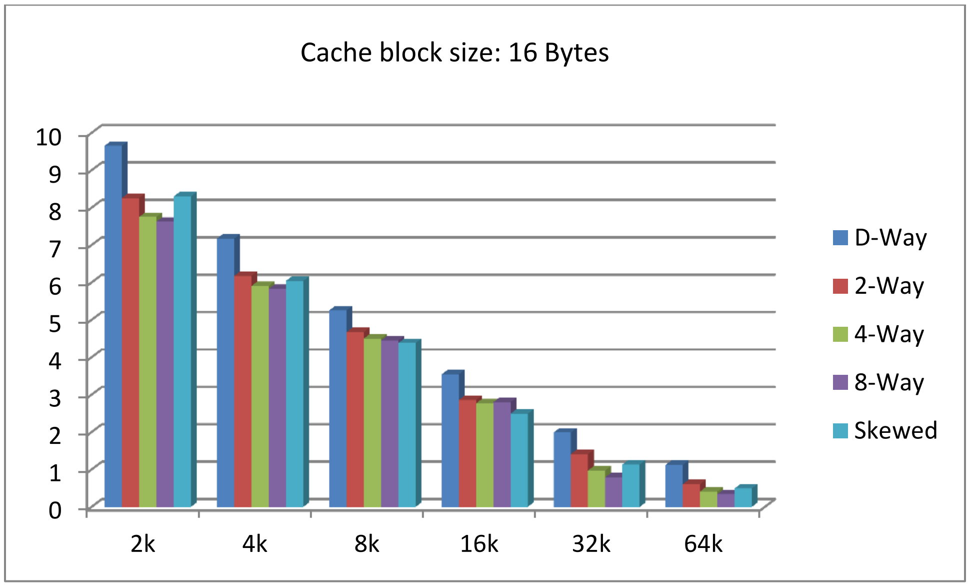


Figure 5.65: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 416.Gamess (Split Cache, Block size 16 Bytes)

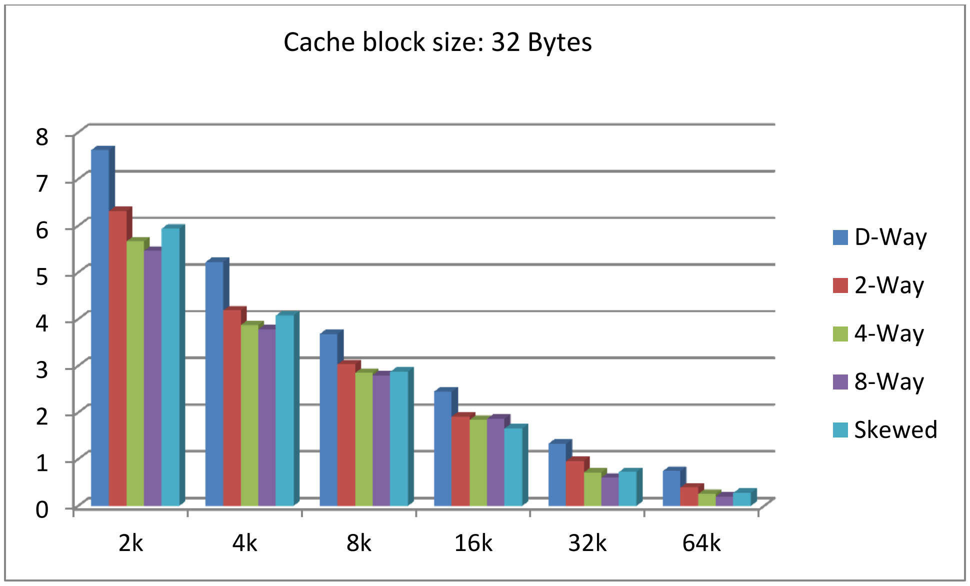


Figure 5.66: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 416.Gamess (Split Cache, Block size 32 Bytes)

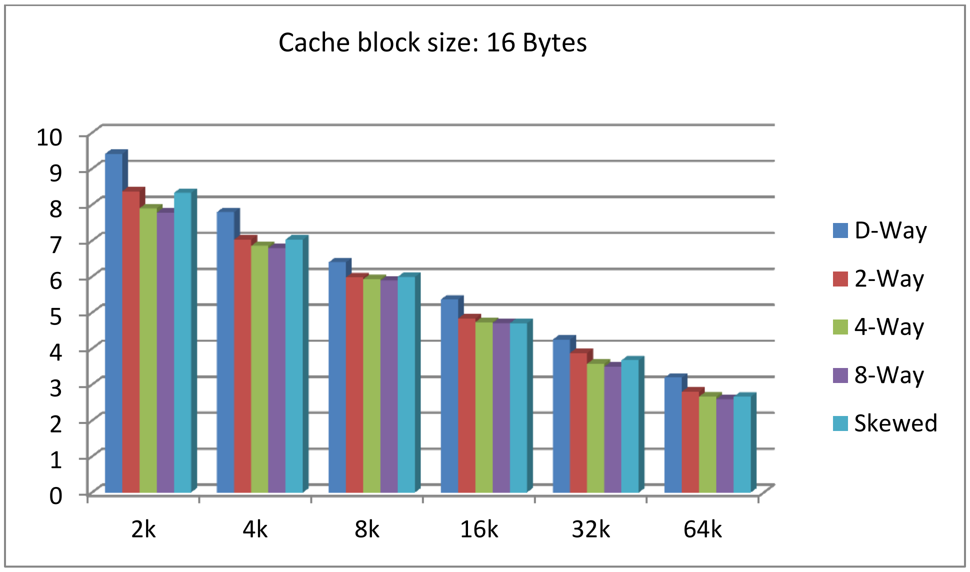


Figure 5.67: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 450.Soplex (Split Cache, Block size 16 Bytes)

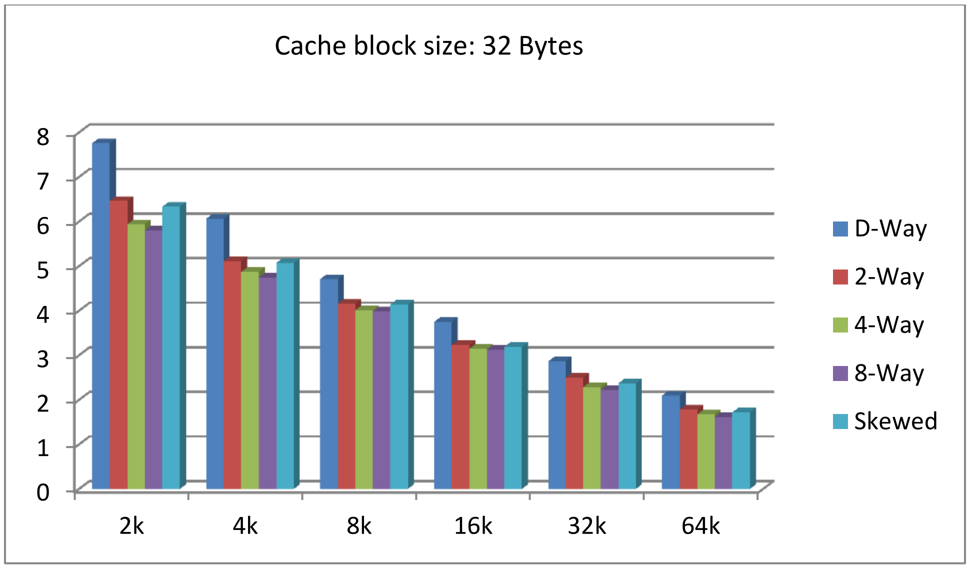


Figure 5.68: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 450.Soplex (Split Cache, Block size 32 Bytes)

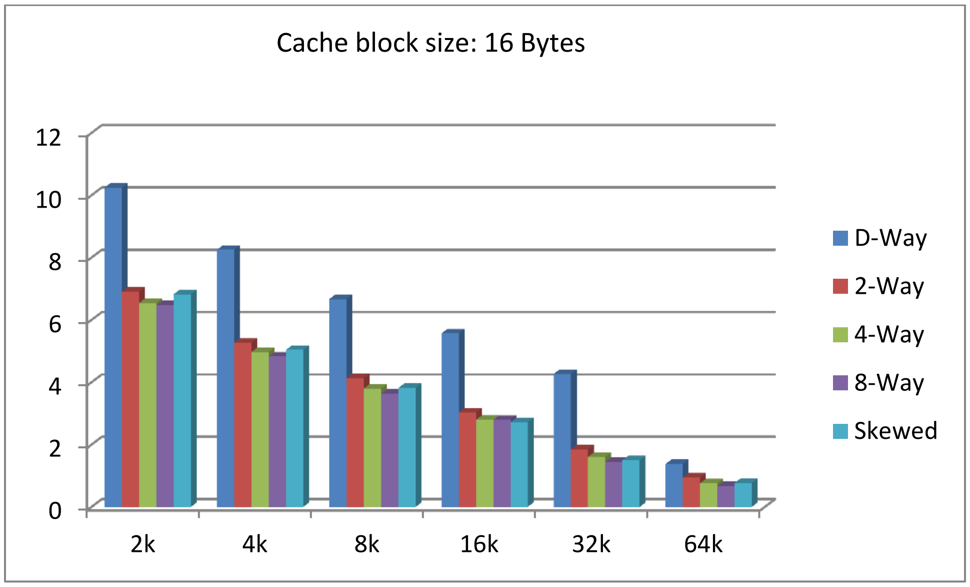


Figure 5.69: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 454.Calculix (Split Cache, Block size 16 Bytes)

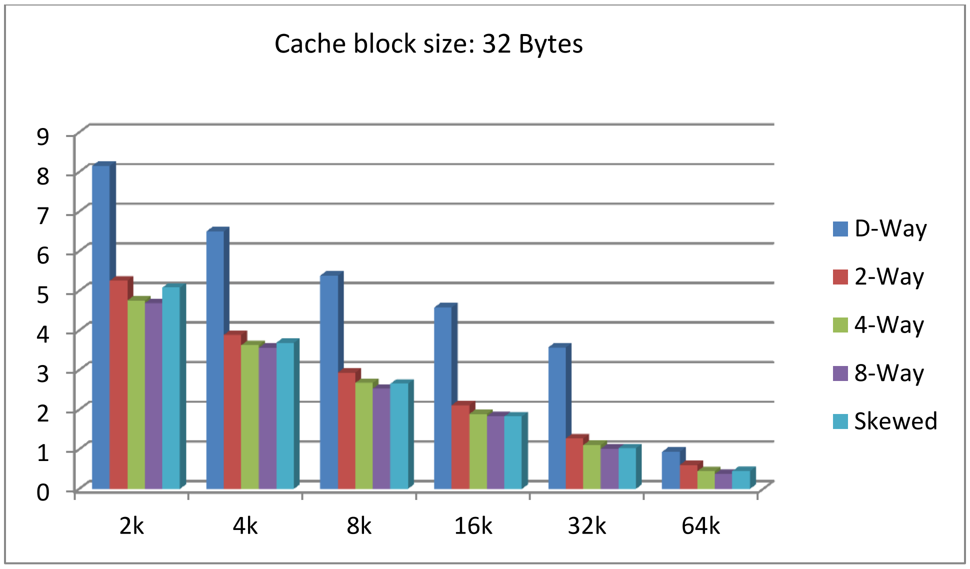


Figure 5.70: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 454.Calculix (Split Cache, Block size 32 Bytes)

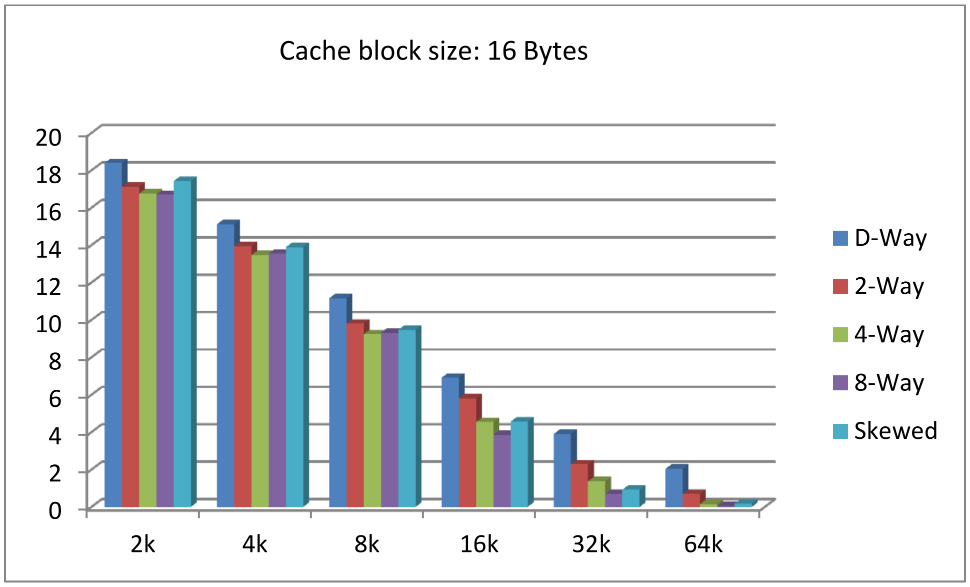


Figure 5.71: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 471.Omnetpp (Split Cache, Block size 16 Bytes)

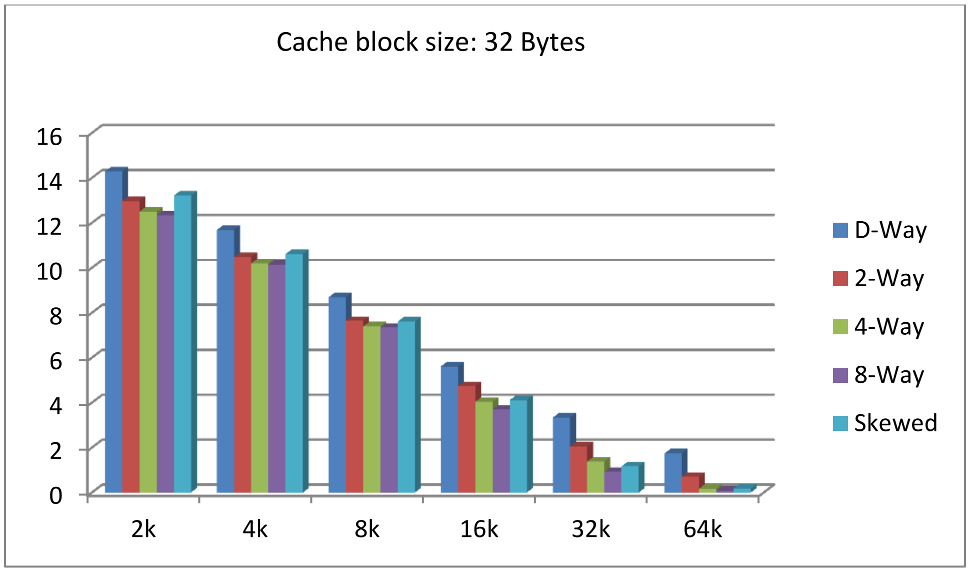


Figure 5.72: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 471.Omnetpp (Split Cache, Block size 32 Bytes)

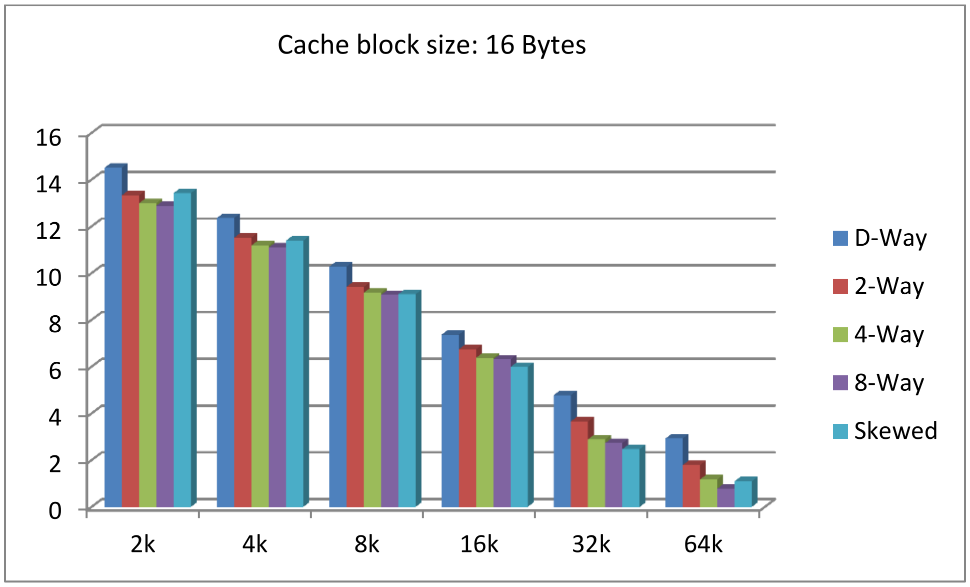


Figure 5.73: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 483.Xalancbmk (Split Cache, Block size 16 Bytes)

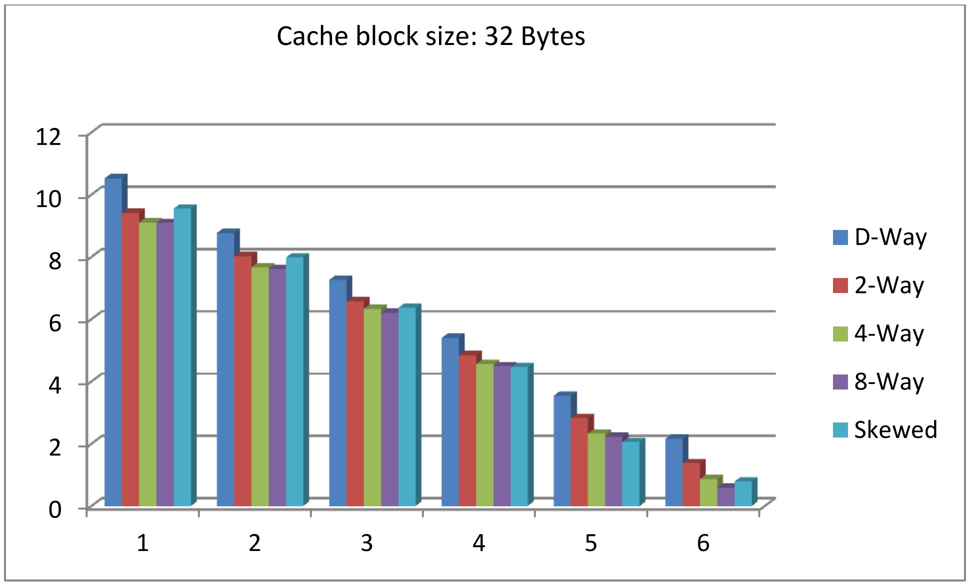


Figure 5.74: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 483.Xalancbmk (Split Cache, Block size 32 Bytes)

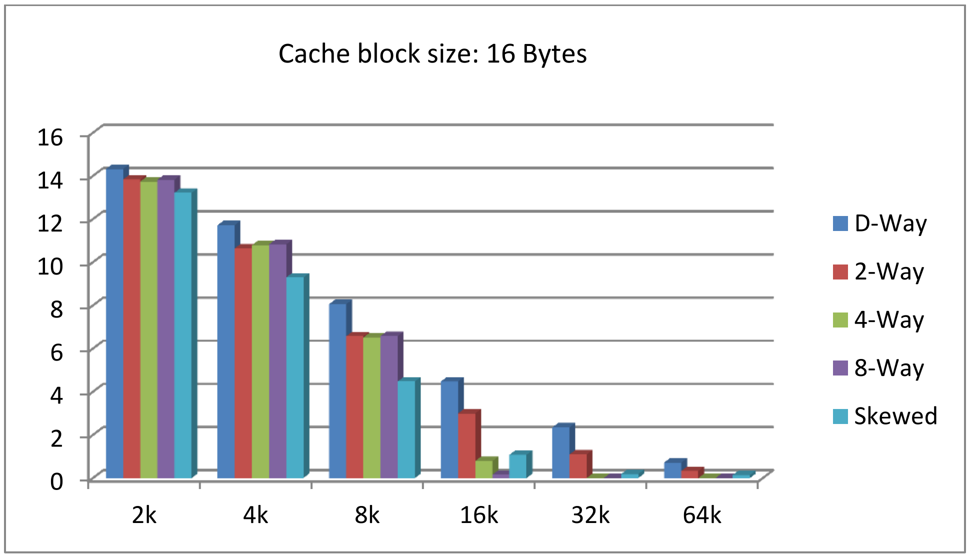


Figure 5.75: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 998.Specrand (Split Cache, Block size 16 Bytes)

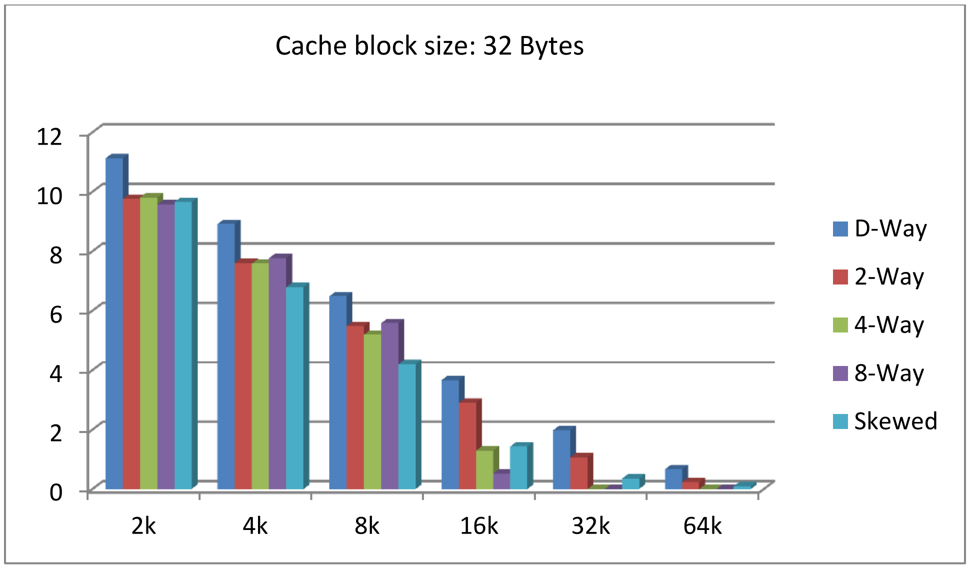


Figure 5.76: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 998.Specrand (Split Cache, Block size 32 Bytes)

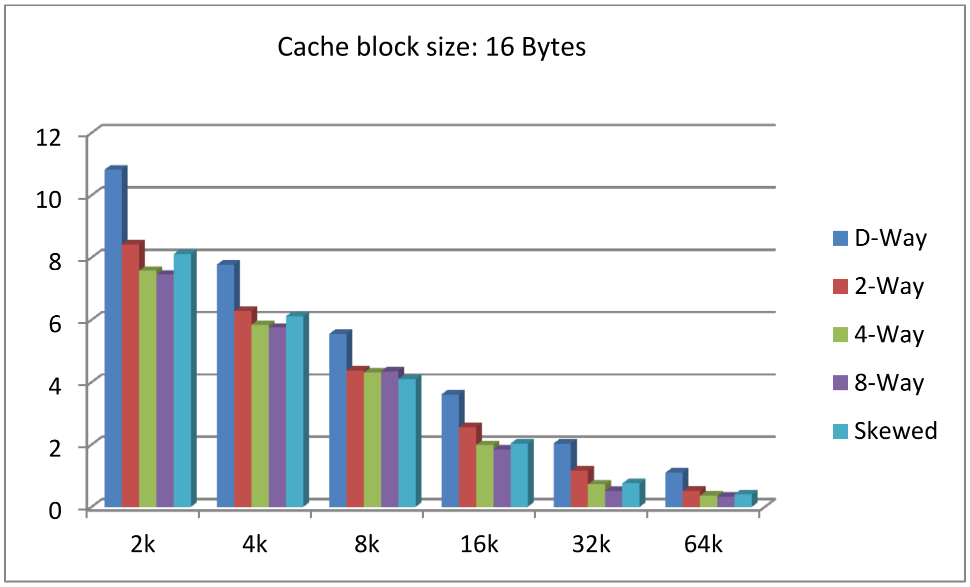


Figure 5.77: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 416.Gamess (Unified Cache, Block size 16 Bytes)

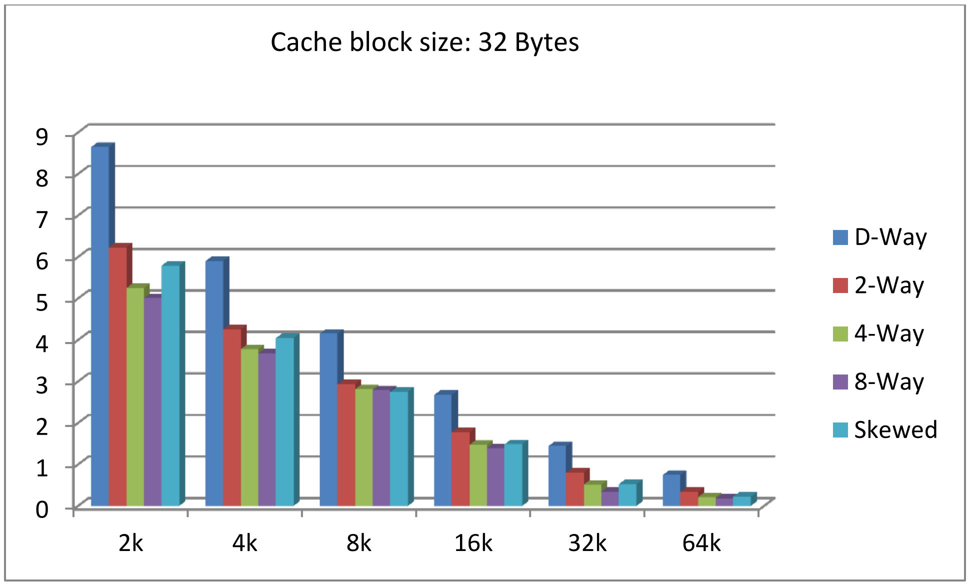


Figure 5.78: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 416.Gamess (Unified Cache, Block size 32 Bytes)

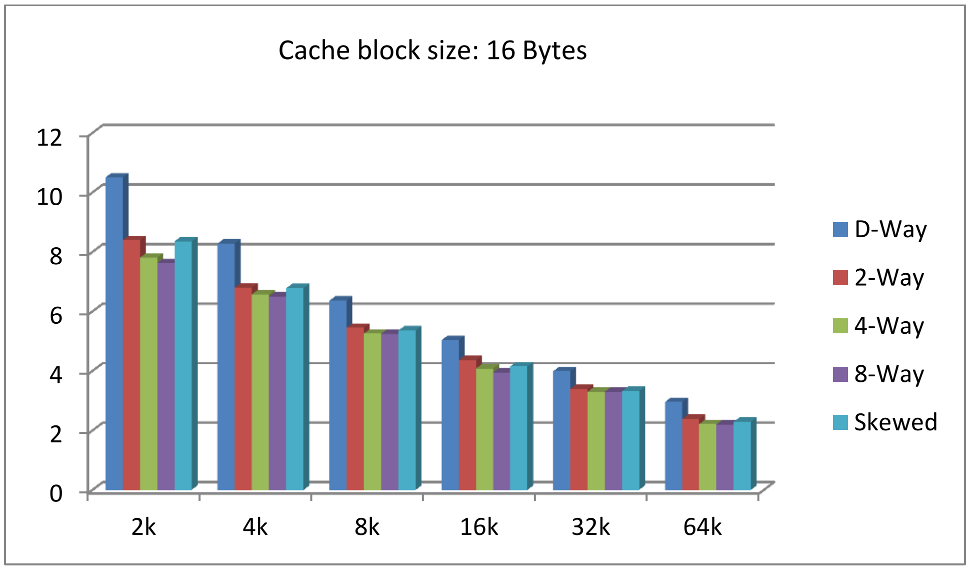


Figure 5.79: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 450.Soplex (Unified Cache, Block size 16 Bytes)

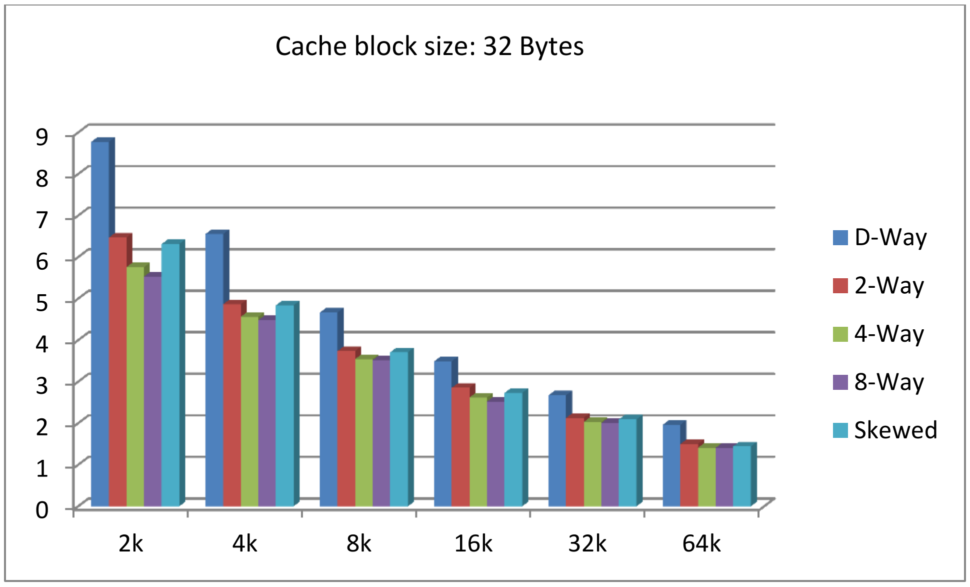


Figure 5.80: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 450.Soplex (Unified Cache, Block size 32 Bytes)

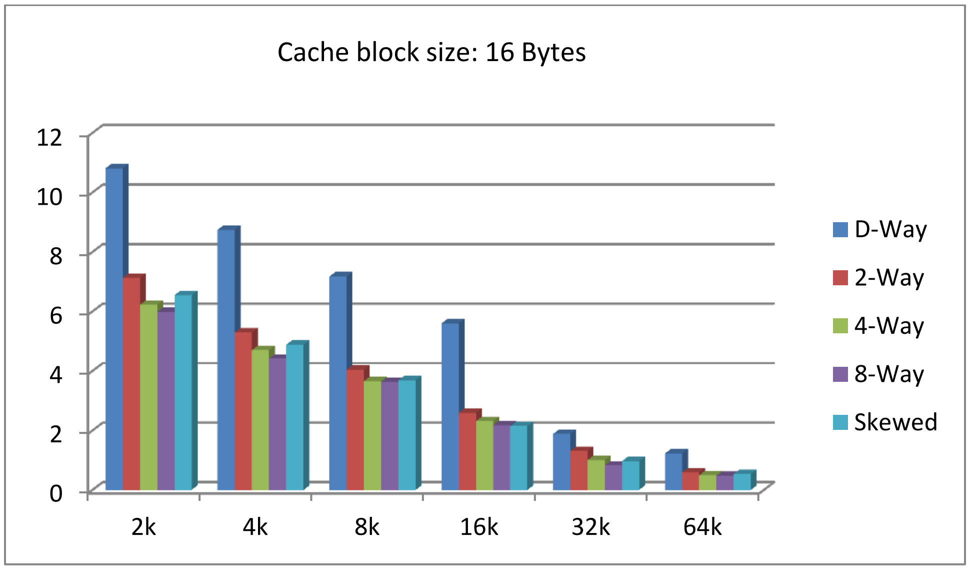


Figure 5.81: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 454.Calculix (Unified Cache, Block size 16 Bytes)

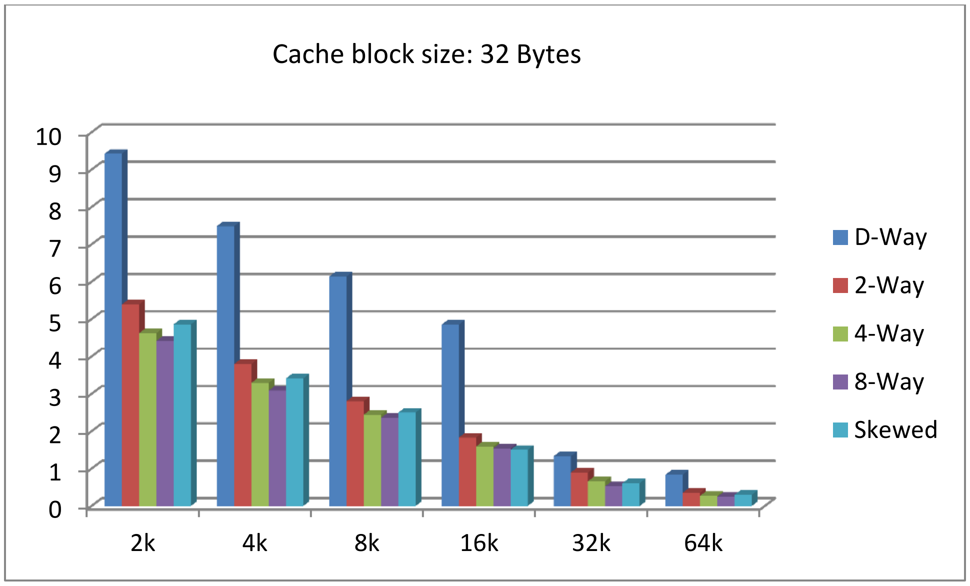


Figure 5.82: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 454.Calculix (Unified Cache, Block size 32 Bytes)

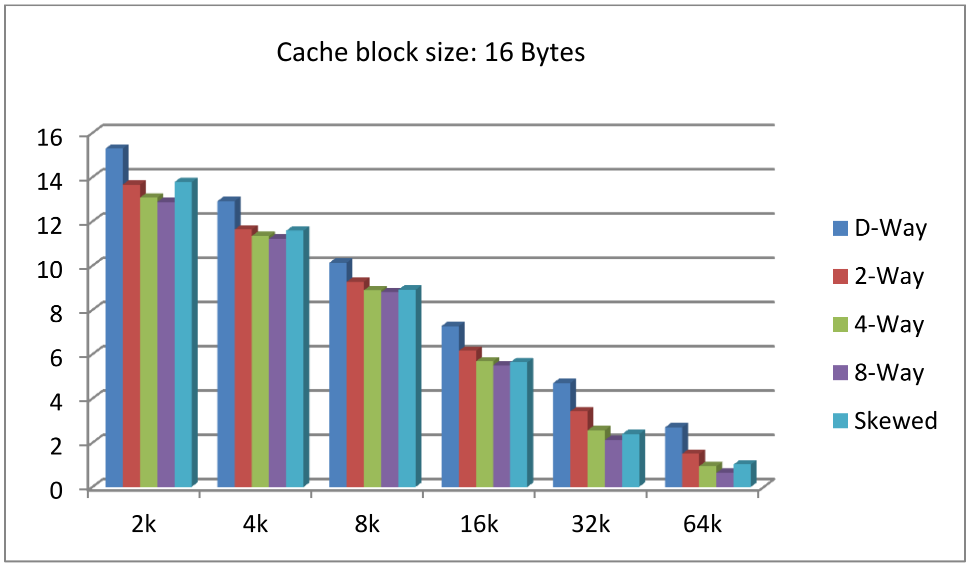


Figure 5.83: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 483.Xalancbmk (Unified Cache, Block size 16 Bytes)

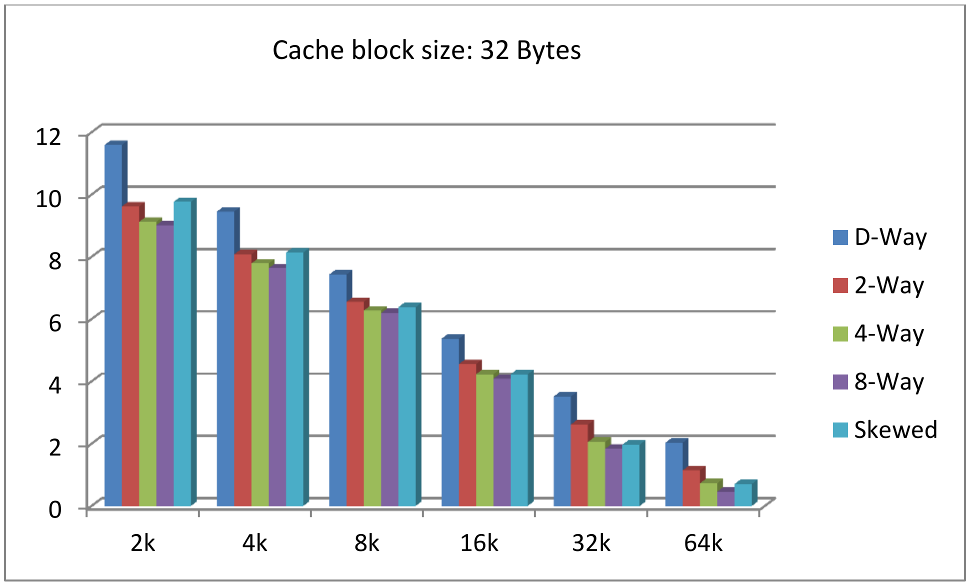


Figure 5.84: Cache Miss Rate for Conventional and 2-Way Skewed Cache for 483.Xalancbmk (Unified Cache, Block size 32 Bytes)

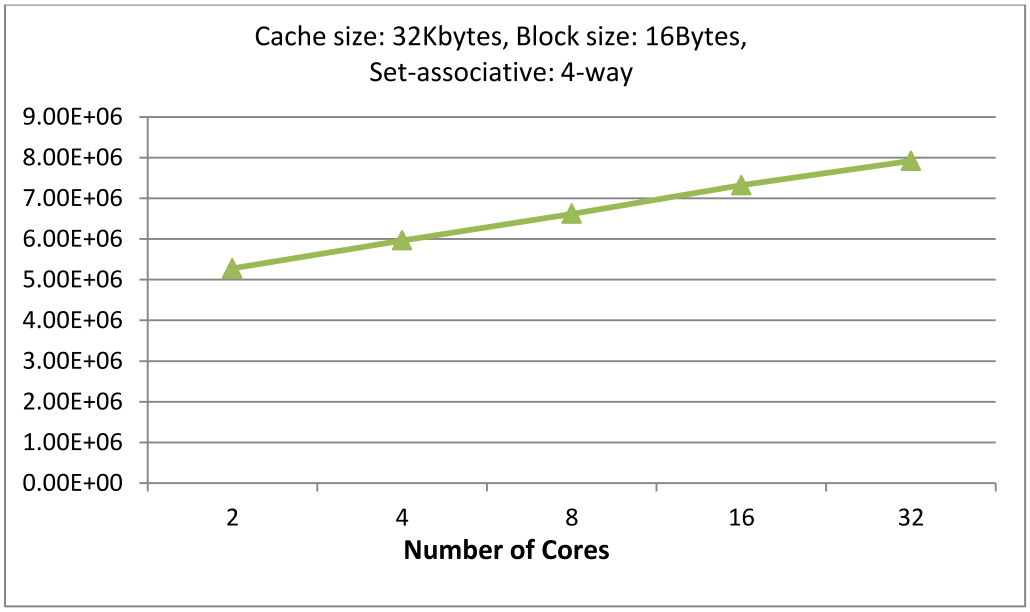


Figure 5.85: Bus Traffic (Raytrace)

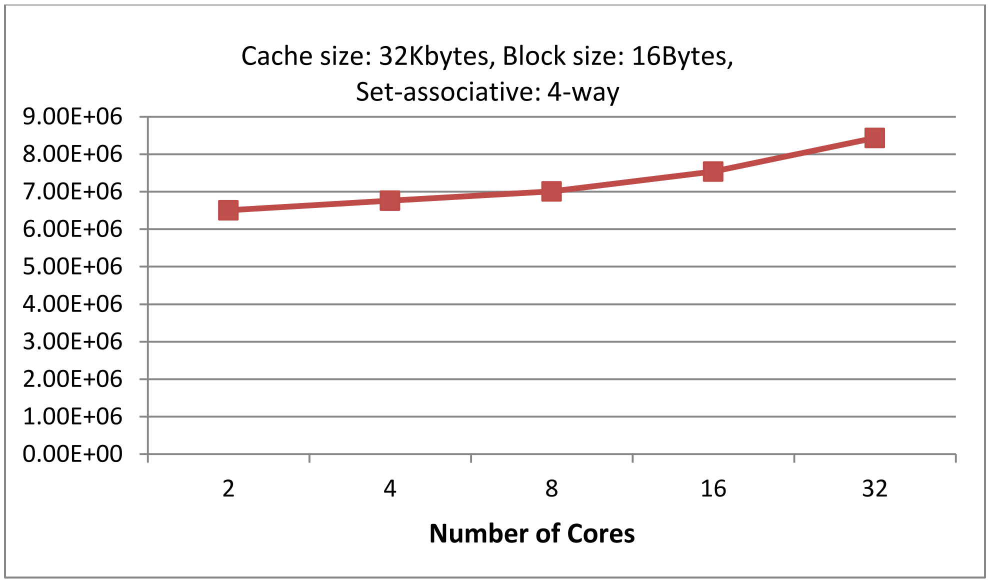


Figure 5.86: Bus Traffic (Radiosity)

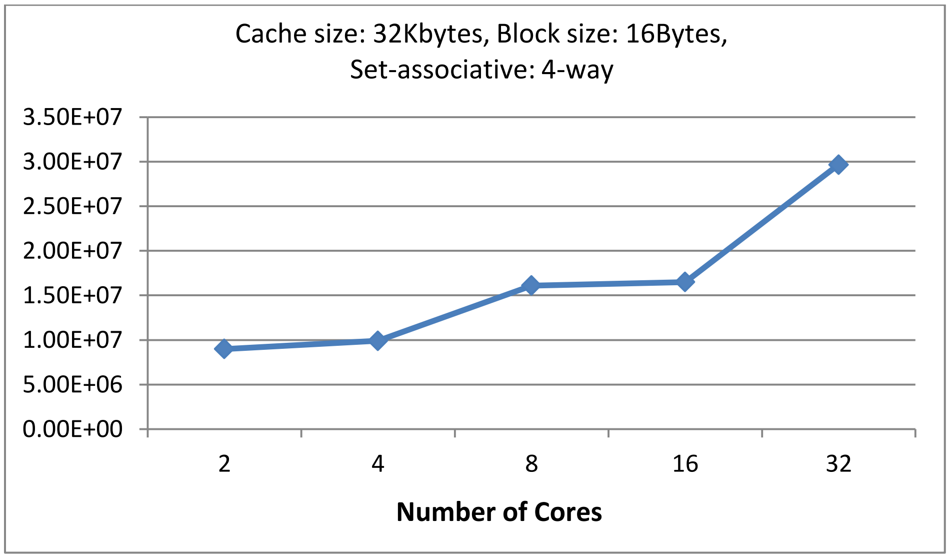


Figure 5.87: Bus Traffic (Barnes)

CHAPTER VI

CONCLUSION

6.1 Conclusion

We proposed a flexible multi-core functional cache simulator, FM-SIM, for modern general-purpose CPU or embedded processors. The proposed simulator needs to work with Pin, which is an open-source dynamic instrumentation tool, to provide a trace file for benchmark programs to feed into the FM-SIM.

We are hoping that this simulator is suitable for developing various cache memories for current multi-core processors since it is easy to modify and port any designed cache schemes into the simulator to evaluate. In the experimental results, eleven different benchmark programs were used to verify and analyze single-core and multi-core cache schemes. The results show that the proposed cache simulator works accurately.

We have a plan to release the FM-SIM as an open-source (like Pin) to share it with research and/or academic community. In addition, we expect more functions can be added to the FM-SIM through the active contributions from the academic and research community in the near future as a public domain.

CHAPTER VII

SOURCE CODE

FM-SIM.cpp

```
# include "headerfile.h"

int main(int argc, char ** argv)

{

        ReadCommand(argc, argv);

        Get_Index_Tag_Block_Bits();

        Design_Cache();

        Set_Initial_Value();

        ifstream trace(argv[argc-1]);

        if(protocol==0)//Singal core

        {

                cout<<"PROTOCOL: \t\tSINGLECORE\n\t***SIMLATION IS
RUNNING***\n";

                while(trace >> PID >> types >> line)

                {

                        Fetch_MemoryAddress();

                        Calc_TagValue_IndexNumber_BlockNumber();

                        Find_Hit_or_Miss_Singlecore();
                        Path_After_Hit_Or_Miss_Singlecore();
                }
                Result_SINGLE();
        }
        else if(protocol==1)//SNOOPY
        {
                cout<<"PROTOCOL: \t\tSNOOPY\n\t***SIMLATION IS RUNNING***\n";
                while(trace >> PID >> types >> line)
                {
                        Fetch_MemoryAddress();
                        Calc_TagValue_IndexNumber_BlockNumber();
                        Find_Hit_or_Miss_Multicore();
                        Path_After_Hit_Or_Miss_SNOOPY();
                }
                Result_MESI_Or_SNOOPY();
        }
        else if(protocol==2)//MESI
        {cout<<"PROTOCOL: \t\tMESI\n\t***SIMLATION IS RUNNING***\n";
                while(trace >> PID >> types >> line)
                {
                        Fetch_MemoryAddress();
                        Calc_TagValue_IndexNumber_BlockNumber();
                        Find_Hit_or_Miss_Multicore();
                Path_After_Hit_Or_Miss_MESI();
            }
            Result_MESI_Or_SNOOPY();
        }
        else if(protocol==3)//FIREFLY
        {
            cout<<"PROTOCOL: \t\tFIREFLY\n\t***SIMLATION IS RUNNING***\n";
            while(trace >> PID >> types >> line)
            {
                Fetch_MemoryAddress();
                Calc_TagValue_IndexNumber_BlockNumber();
                Find_Hit_or_Miss_Multicore();
                Path_After_Hit_Or_Miss_FIREFLY();
            }
            Result_FIREFLY();
        }
        else
            cout<<"\n Invalid Protocol Selected\n";
}
```

Header.h

```
# include <stdio.h>
# include <stdlib.h>
# include <iomanip>
# include <iostream>

# include <fstream>

# include <string>

# include <bitset>

using namespace std;

#include "GlobalVariable.h"

#include"create.h"

#include"Function.h"
```

<u>GlobalVariable.h</u>

```
int cachesize,blocksize,asc,threadno,protocol,bkbit,idbit;

int tagbit,*RamValid,*RamDirty,PID=-1,HitBankN=-1,HoM=-1,wp=0,LRUPolicy=0;

unsigned int Mem_Ad,AddrTagV,AddrIndexN,AddrBlockN;

string line;

char types;

string input,commandl1,commandl2;

# define MAX_CACHE 128
```

<u>Create.h</u>

```
struct Indexn

{

        int Tag,Valid,LRU,DB;

        char Dirtybit;

};

struct Indexn **cache[MAX_CACHE];
struct record
{
	double RHcount,WHcount,RMcount,WMcount,noload,nowrite,EXTRAMISS;
	double EXTRAHIT,IHcount,IMcount,noinst,MUPL,BusRds,BusWrs,cache2cache;
	double RHS,RHP,WHP,WHS,MemTransection,eee,BusRdxs,BusUpgrs;
	double Flush,WB,ShrHitR,ExcHitR,ModHitR,ShrHitW,ExcHitW,ModHitW;
}*cacherecord;
struct lookthreadVar
{
	int HTN,HTB;
	char HTStatus;
}*lookthreadrecord;
```

Function.h

```
int chartoint(int x);
void ReadCommand(int argc, char ** argv);
void inputf(string s);
void Get_Cache_parameter(string s);
void Get_Index_Tag_Block_Bits();
void Fetch_MemoryAddress();
void Calc_TagValue_IndexNumber_BlockNumber();
void Find_Hit_or_Miss_Multicore();
void Find_Hit_or_Miss_Singlecore();
void Path_After_Hit_Or_Miss_Singlecore();
void Path_After_Hit_Or_Miss_MESI();
void Path_After_Hit_Or_Miss_FIREFLY();
void Path_After_Hit_Or_Miss_SNOOPY();
int Find_HighestLRU_Bank();
int lookotherthread(int x);
void Update_LRU(int threadnoo,int bank);
void LRUF2(int threadnoo,int bank);
void Result_SINGLE();
void Result_MESI_Or_SNOOPY();
void Result_FIREFLY();
void Design_Cache();
void Set_Initial_Value();
int chartoint(int x) // convert char into integer value
{
        int value;
        if(x==48)
        {
                value=0;
        }
        if(x==49)
        {
                value=1;
        }
```

```
if(x==50)
{
        value=2;
}
if(x==51)
{
        value=3;
}
if(x==52)
{
        value=4;
}
if(x==53)
{
        value=5;
}
if(x==54)
{
        value=6;
}
if(x==55)
{
        value=7;
```

```
}
if(x==56)
{
        value=8;
}
if(x==57)
{
        value=9;
}
if(x==97)
{
        value=10;
}
if(x==98)
{
        value=11;
}
if(x==99)
{
        value=12;
}
if(x==100)
{
```

```
                value=13;
        }
        if(x==101)
        {
                value=14;
        }
        if(x==102)
        {
                value=15;
        }
        return value;
}
void ReadCommand(int argc, char ** argv)
{
string line;
        for (int i=1; i<argc-1; i++)
        {
                line=argv[i];
                if(line[0]==45) //45 is ASCII value of '-'
                {
                        if (line[1]==108)  // 108 is ASCII value of 'l'
                        {
                                if(line[2]==49) // 49 is ASCII value of '1'
                                {
                                    i++;
                                    commandl1=argv[i];
                                    Get_Cache_parameter(argv[i]);
                                }
                        }
                        else if (line[1]==67)// 67 is ASCII value of 'C'
                        {
                                threadno=chartoint(line[2]);
                        }
                        else if (line[1]==80)// 80 is ASCII value of 'P'
                        {
                                protocol=chartoint(line[2]);
                        }
                }
        }
cout<<"\nCacheSize\t\t"<<(1<<cachesize);
cout<<"\nBlockSize\t\t"<<(1<<blocksize);
cout<<"\nAssoc:\t\t\t"<<(1<<asc);
cout<<"\nTrace File:\t\t"<<argv[argc-1]<<"\n";
}
void Get_Cache_parameter(string s) // with this function we split command into parts to get
value of index, tag and block
```

```
{
   int c,j=0,valu[5];
         char e;
         valu[0]=0;
         valu[1]=0;
         valu[2]=0;
         valu[3]=0;
         valu[4]=0;
         for(int i=0; i<=4; i++)
         {
                  e=s[j];
                  valu[i]=chartoint(e);
                  j=j+1;
                  e=s[j];
                  while(e!=58)
                  {
                           valu[i]=valu[i]*10;
                           valu[i]=valu[i]+chartoint(e);
                           j=j+1;e=s[j];
                  }
         j=j+1;
         }
         cachesize=valu[0];
```

```
        blocksize=valu[1];

        asc=valu[2];

        wp=valu[3];

        LRUPolicy=valu[4];

}

void Get_Index_Tag_Block_Bits() // calculate the bits of index, tag, and block size

{

        bkbit=blocksize;

        idbit=cachesize-bkbit-asc;

        tagbit=32-idbit-bkbit;

}

void Design_Cache()

{

        for(int i=0; i<threadno; i++)

        {

                cache[i]=new struct Indexn *[1<<asc];

                for(int j=0; j<(1<<asc); j++)

                {

                        cache[i][j]=new struct Indexn [1<<idbit];

                }

        }

        cacherecord=new struct record [threadno];

        lookthreadrecord=new struct lookthreadVar [threadno];
```

```
}
void Set_Initial_Value()
{
        for(int i=0; i<threadno; i++)
        {
                for(int j=0; j<(1<<asc); j++)
                {
                        for(int k=0; k<(1<<idbit); k++)
                        {
                                cache[i][j][k].Valid=0;
                                cache[i][j][k].Tag=-1;
                                cache[i][j][k].Dirtybit='I';
                                cache[i][j][k].LRU=j;
                                cache[i][j][k].DB=1;
                        }
                }
                cacherecord[i].RHcount=0;
                cacherecord[i].RMcount=0;
                cacherecord[i].WHcount=0;
                cacherecord[i].WMcount=0;
                cacherecord[i].noload=0;
                cacherecord[i].nowrite=0;
                cacherecord[i].EXTRAMISS=0;
```

```
cacherecord[i].EXTRAHIT=0;

cacherecord[i].BusRds=0;

cacherecord[i].BusWrs=0;

cacherecord[i].cache2cache=0;

cacherecord[i].RHS=0;

cacherecord[i].RHP=0;

cacherecord[i].WHP=0;

cacherecord[i].WHS=0;

cacherecord[i].eee=0;

cacherecord[i].BusRdxs=0;

cacherecord[i].BusUpgrs=0;

cacherecord[i].Flush=0;

cacherecord[i].WB=0;

cacherecord[i].ShrHitR=0;

cacherecord[i].ExcHitR=0;

cacherecord[i].ModHitR=0;

cacherecord[i].ShrHitW=0;

cacherecord[i].ExcHitW=0;

cacherecord[i].ModHitW=0;

cacherecord[i].IHcount=0;

cacherecord[i].IMcount=0;

cacherecord[i].noinst=0;
```

```
        }
}
void Fetch_MemoryAddress() // Fetch memory address of instrction
{
        int b1,c1;
        b1=line[2];
        Mem_Ad=chartoint(b1);
        for( int a = 3; a < line.size(); a = a + 1 )
        {
                Mem_Ad=Mem_Ad*16;b1=line[a];
                c1=chartoint(b1);
                Mem_Ad=Mem_Ad+c1;
        }
}
void Calc_TagValue_IndexNumber_BlockNumber() //Calculating Index number, Tag value and
Block number from memory address
{
        AddrBlockN=Mem_Ad&((1<<bkbit)-1);
        AddrIndexN=(Mem_Ad>>bkbit)&((1<<idbit)-1);
        AddrTagV=(Mem_Ad>>(idbit+bkbit))&((1<<tagbit)-1);
}
void Find_Hit_or_Miss_Singlecore()
{
```

```
int a=2,b;
b=1<<asc;
for(int i=0; i<b; i++)
{
        if(cache[PID][i][AddrIndexN].Valid==1)
        {
                if(cache[PID][i][AddrIndexN].Tag==AddrTagV)
                {
                        a=1;
                        HoM=1;// it mean instruction is hit
                        HitBankN=i;
                        if(types=='O')
                        {
                                cacherecord[PID].IHcount++;cacherecord[PID].noinst++;
                        }
                        if(types=='r')
                        {
                                cacherecord[PID].RHcount++;cacherecord[PID].noload++;
                        }
                        if(types=='w')
                        {

cacherecord[PID].WHcount++;cacherecord[PID].nowrite++;
```

```
                }
                break;
            }
        }
    }
    if(a==2)
    {
        HoM=0;//it mean instruction is miss
        if(types=='O')
        {
            cacherecord[PID].IMcount++;cacherecord[PID].noinst++;
        }
        if(types=='r')
        {
            cacherecord[PID].RMcount++;cacherecord[PID].noload++;
        }
        if(types=='w')
        {
            cacherecord[PID].WMcount++;cacherecord[PID].nowrite++;
        }
    }
}
void Path_After_Hit_Or_Miss_Singlecore()
```

```
{
        if(HoM==1)// In case of Hit
        {
                if(types=='r'&&wp==1)
                {
                        Update_LRU(PID,HitBankN);
                }
                if(types=='r'&&wp==2)
                {
                        Update_LRU(PID,HitBankN);
                }
                if(types=='w'&&wp==1)
                {
                        Update_LRU(PID,HitBankN);
                        cacherecord[PID].MUPL++;
                }
                if(types=='w'&&wp==2)
                {
                        Update_LRU(PID,HitBankN);
                        cache[PID][HitBankN][AddrIndexN].DB=1;
                }
                if(types=='O'&&wp==1)
                {
```

```
                Update_LRU(PID,HitBankN);
        }
        if(types=='O'&&wp==2)
        {
                Update_LRU(PID,HitBankN);
        }
}
if(HoM==0)// In case of miss
{
        int replacebankNo,nohitthreads;
        if(types=='r'&&wp==1) // for load with write through
        {
                replacebankNo=Find_HighestLRU_Bank();
                cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                cache[PID][replacebankNo][AddrIndexN].Valid=1;
                Update_LRU(PID,replacebankNo);
        }
        if(types=='r'&&wp==2) // for load hit with Write-Back
        {
                replacebankNo=Find_HighestLRU_Bank();
                if(cache[PID][replacebankNo][AddrIndexN].DB==0) // if dirtybit=0
                {
                        cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                        cache[PID][replacebankNo][AddrIndexN].Valid=1;

                        Update_LRU(PID,replacebankNo);

                }

                if(cache[PID][replacebankNo][AddrIndexN].DB==1) // if dirtybit=0

                {

                        cacherecord[PID].MUPL++;

                        cache[PID][replacebankNo][AddrIndexN].DB=0;

                        cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;

                        cache[PID][replacebankNo][AddrIndexN].Valid=1;

                        Update_LRU(PID,replacebankNo);

                }

        }

        if(types=='w'&&wp==1) // for store hit with write through

        {

                cacherecord[PID].MUPL++;

        }

        if(types=='w'&&wp==2) // for store with Write-Back

        {

                replacebankNo=Find_HighestLRU_Bank();

                if(cache[PID][replacebankNo][AddrIndexN].DB==1) // if dirtybit=0

                {

                        cacherecord[PID].MUPL++;

                        cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                        cache[PID][replacebankNo][AddrIndexN].Valid=1;
                        Update_LRU(PID,replacebankNo);
                }
                if(cache[PID][replacebankNo][AddrIndexN].DB==0) // if dirtybit=0
                {
                        cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                        cache[PID][replacebankNo][AddrIndexN].Valid=1;
                        cache[PID][replacebankNo][AddrIndexN].DB=1;
                        Update_LRU(PID,replacebankNo);
                }
        }
        if(types=='O'&&wp==1) // just for instruciton hit with write through
        {
                replacebankNo=Find_HighestLRU_Bank();
                cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                cache[PID][replacebankNo][AddrIndexN].Valid=1;
                Update_LRU(PID,replacebankNo);
        }
        if(types=='O'&&wp==2) // just for instruciton hit with Write-Back
        {
                replacebankNo=Find_HighestLRU_Bank();
                if(cache[PID][replacebankNo][AddrIndexN].DB==0) // if dirtybit=0
                {
```

```
                        cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                        cache[PID][replacebankNo][AddrIndexN].Valid=1;
                        Update_LRU(PID,replacebankNo);
                    }
                    if(cache[PID][replacebankNo][AddrIndexN].DB==1) // if dirtybit=0
                    {
                        cacherecord[PID].MUPL++;
                        cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                        cache[PID][replacebankNo][AddrIndexN].Valid=1;
                        cache[PID][replacebankNo][AddrIndexN].DB=0;
                        Update_LRU(PID,replacebankNo);
                    }
                }
        }


}
void Find_Hit_or_Miss_Multicore()
{
        int a=2,b;
        b=1<<asc;
        for(int i=0; i<b; i++)
        {
                if(cache[PID][i][AddrIndexN].Valid==1)
        {
                if(cache[PID][i][AddrIndexN].Tag==AddrTagV)
                {
                        a=1;
                        HoM=1;// it mean instruction is hit
                        HitBankN=i;
                        if(types=='r')
                        {
                                cacherecord[PID].RHcount++;cacherecord[PID].noload++;
                        }
                        if(types=='w')
                        {

cacherecord[PID].WHcount++;cacherecord[PID].nowrite++;
                        }
                        break;
                }
        }
}
if(a==2)
{
        HoM=0;//it mean instruction is miss
        if(types=='r')
```

```
                {
                        cacherecord[PID].RMcount++;cacherecord[PID].noload++;
                }
                if(types=='w')
                {
                        cacherecord[PID].WMcount++;cacherecord[PID].nowrite++;
                }
        }
}
void Path_After_Hit_Or_Miss_MESI() // MESI
{
        if(HoM==1)// In case of Hit
        {
                Update_LRU(PID,HitBankN);
                if(types=='r')
                {
                        cacherecord[PID].RHP++;
                        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='E')
                        {
                                cacherecord[PID].ExcHitR++;
                        }
                        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='S')
                        {
```

```
                cacherecord[PID].ShrHitR++;
        }
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='M')
        {
                cacherecord[PID].ModHitR++;
        }
}
if(types=='w')
{
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='E')
        {
                cacherecord[PID].ExcHitW++;
        }
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='S')
        {
                cacherecord[PID].ShrHitW++;
        }
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='M')
        {
                cacherecord[PID].ModHitW++;
        }
        cacherecord[PID].WHP++;
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='S')
```

```
                {
                        int hitno;
                        hitno=lookotherthread(PID);
                        if(hitno!=0)
                        {
                                cacherecord[PID].BusUpgrs++;
                        }
                        for(int i=0; i<hitno;i++)
                        {

cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Dirtybit='I';

cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Valid=0;

LRUF2(lookthreadrecord[i].HTN,lookthreadrecord[i].HTB);
                        }
                }
                cache[PID][HitBankN][AddrIndexN].Dirtybit=='M';
        }
}
if(HoM==0)// In case of miss
{
        int replacebankNo,nohitthreads;
```

```
replacebankNo=Find_HighestLRU_Bank();
nohitthreads=lookotherthread(PID);
if(nohitthreads==0)
{
        if(types=='r')
        {
                cacherecord[PID].BusRds++;
                cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                cache[PID][replacebankNo][AddrIndexN].Valid=1;
                cache[PID][replacebankNo][AddrIndexN].Dirtybit='E';
                Update_LRU(PID,replacebankNo);
        }
        if(types=='w')
        {
                cacherecord[PID].BusRdxs++;
                cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                cache[PID][replacebankNo][AddrIndexN].Valid=1;
                cache[PID][replacebankNo][AddrIndexN].Dirtybit='M';
                Update_LRU(PID,replacebankNo);
        }
}
if(nohitthreads!=0)
{ cacherecord[PID].cache2cache++;
```

```
        if(types=='r')
        {
                cacherecord[PID].RHS++;
                cacherecord[PID].BusRds++;
                cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                cache[PID][replacebankNo][AddrIndexN].Valid=1;
                cache[PID][replacebankNo][AddrIndexN].Dirtybit='S';
                Update_LRU(PID,replacebankNo);
                for(int i=0; i<nohitthreads;i++)
                {
                        if(lookthreadrecord[i].HTStatus!='S')
                        {

cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Dirtybit='S';
                        }
                }
        }
        if(types=='w')
        {
                cacherecord[PID].WHS++;
                cacherecord[PID].BusRdxs++;
                cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                cache[PID][replacebankNo][AddrIndexN].Valid=1;
```

```
                                        cache[PID][replacebankNo][AddrIndexN].Dirtybit='M';
                                        Update_LRU(PID,replacebankNo);
                                        for(int i=0; i<nohitthreads;i++)
                                        {

        cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Dirtybit='I';

        cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Valid=0;

        LRUF2(lookthreadrecord[i].HTN,lookthreadrecord[i].HTB);
                                        }
                                }
                        }

                }
}
void Path_After_Hit_Or_Miss_FIREFLY() // FIREFLY
{
        if(HoM==1)// In case of hit
        {
                Update_LRU(PID,HitBankN);
                if(types=='r')
                {
```

```
        cacherecord[PID].RHP++;
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='V')
        {
                cacherecord[PID].ExcHitR++;
        }
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='S')
        {
                cacherecord[PID].ShrHitR++;
        }
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='D')
        {
                cacherecord[PID].ModHitR++;
        }
}

if(types=='w')
{
        cacherecord[PID].WHP++;
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='V')
        {
        cache[PID][HitBankN][AddrIndexN].Dirtybit=='D';
        }
        else if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='S')
```

```
                {
                        int hitno=0;
                        hitno=lookotherthread(PID);
                        if(hitno!=0)
                                {
                                        cache[PID][HitBankN][AddrIndexN].Dirtybit=='S';
                                        cacherecord[PID].BusWrs++;
                                }
                        if(hitno==0){cache[PID][HitBankN][AddrIndexN].Dirtybit=='D';}
                }
        }
}
if(HoM==0)// In case of miss
{
        int replacebankNo,nohitthreads;
        replacebankNo=Find_HighestLRU_Bank();
        nohitthreads=lookotherthread(PID);
        if(nohitthreads==0)
        {
                if(types=='r')
                {
                        cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                        cache[PID][replacebankNo][AddrIndexN].Valid=1;
                cache[PID][replacebankNo][AddrIndexN].Dirtybit='V';
                Update_LRU(PID,replacebankNo);
                cacherecord[PID].BusRds++;
        }
        if(types=='w')
        {
                cacherecord[PID].BusRds++;
                cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                cache[PID][replacebankNo][AddrIndexN].Valid=1;
                cache[PID][replacebankNo][AddrIndexN].Dirtybit='D';
                Update_LRU(PID,replacebankNo);
        }
}
if(nohitthreads!=0)
{cacherecord[PID].cache2cache++;
        if(types=='r')
        {
                cacherecord[PID].RHS++;
                cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                cache[PID][replacebankNo][AddrIndexN].Valid=1;
                cache[PID][replacebankNo][AddrIndexN].Dirtybit='S';
                Update_LRU(PID,replacebankNo);
                cacherecord[PID].BusRds++;
```

```
                    for(int i=0; i<nohitthreads;i++)
                    {
                        if(lookthreadrecord[i].HTStatus!='S')
                        {

cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Dirtybit='S';
                        }
                    }
                }
                if(types=='w')
                {
                    cacherecord[PID].WHS++;
                    cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                    cache[PID][replacebankNo][AddrIndexN].Valid=1;
                    cache[PID][replacebankNo][AddrIndexN].Dirtybit='S';
                    Update_LRU(PID,replacebankNo);
                    for(int i=0; i<nohitthreads;i++)
                    {

cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Dirtybit='S';

cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Valid=1;
```

```
        LRUF2(lookthreadrecord[i].HTN,lookthreadrecord[i].HTB);
                        }
                    }
                }

        }
}
void Path_After_Hit_Or_Miss_SNOOPY() //SNOOPY
{
        if(HoM==1)// In case of Hit
        {
                Update_LRU(PID,HitBankN);
                if(types=='r')
                {
                        cacherecord[PID].RHP++;
                        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='S')
                        {
                                cacherecord[PID].ShrHitR++;
                        }
                        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='M')
                        {
                                cacherecord[PID].ModHitR++;
        }
}
if(types=='w')
{
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='S')
        {
                cacherecord[PID].ShrHitW++;
        }
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='M')
        {
                cacherecord[PID].ModHitW++;
        }
        cacherecord[PID].WHP++;
        if(cache[PID][HitBankN][AddrIndexN].Dirtybit=='S')
        {
                int hitno;
                hitno=lookotherthread(PID);
                if(hitno!=0)
                {
                        cacherecord[PID].BusUpgrs++;
                }
                for(int i=0; i<hitno;i++)
                {
```

```
cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Dirtybit='I';

cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Valid=0;

LRUF2(lookthreadrecord[i].HTN,lookthreadrecord[i].HTB);
                        }
                        cache[PID][HitBankN][AddrIndexN].Dirtybit=='M';
                }
        }
}
if(HoM==0)// In case of miss
{
        int replacebankNo,nohitthreads;
        replacebankNo=Find_HighestLRU_Bank();
        nohitthreads=lookotherthread(PID);
        if(nohitthreads==0)
        {
                if(types=='r')
                {
                        cacherecord[PID].BusRds++;
                        cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                        cache[PID][replacebankNo][AddrIndexN].Valid=1;
                cache[PID][replacebankNo][AddrIndexN].Dirtybit='S';
                Update_LRU(PID,replacebankNo);
        }
        if(types=='w')
        {
                cacherecord[PID].BusRdxs++;
                cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                cache[PID][replacebankNo][AddrIndexN].Valid=1;
                cache[PID][replacebankNo][AddrIndexN].Dirtybit='M';
                Update_LRU(PID,replacebankNo);
        }
}
if(nohitthreads!=0)
{ cacherecord[PID].cache2cache++;
        if(types=='r')
        {
                cacherecord[PID].RHS++;
                cacherecord[PID].BusRds++;
                cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                cache[PID][replacebankNo][AddrIndexN].Valid=1;
                cache[PID][replacebankNo][AddrIndexN].Dirtybit='S';
                Update_LRU(PID,replacebankNo);
                for(int i=0; i<nohitthreads;i++)
```

```
                        {
                                if(lookthreadrecord[i].HTStatus!='S')
                                {

cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Dirtybit='S';
                                }
                        }
                }
                if(types=='w')
                {
                        cacherecord[PID].WHS++;
                        cacherecord[PID].BusRdxs++;
                        cache[PID][replacebankNo][AddrIndexN].Tag=AddrTagV;
                        cache[PID][replacebankNo][AddrIndexN].Valid=1;
                        cache[PID][replacebankNo][AddrIndexN].Dirtybit='M';
                        Update_LRU(PID,replacebankNo);
                        for(int i=0; i<nohitthreads;i++)
                        {

cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Dirtybit='I';

cache[lookthreadrecord[i].HTN][lookthreadrecord[i].HTB][AddrIndexN].Valid=0;
```

```
        LRUF2(lookthreadrecord[i].HTN,lookthreadrecord[i].HTB);
                        }
                    }
                }


        }
}
int Find_HighestLRU_Bank() //Find the high LRU value Bank
{
        int a,c,bankno;
        a=cache[PID][0][AddrIndexN].LRU;
        c=1<<asc;
        for(int i=0; i<c; i++)
        {
                if(a<=cache[PID][i][AddrIndexN].LRU)
                {
                        bankno=i;
                        a=cache[PID][i][AddrIndexN].LRU;
                }
        }
        return bankno;
}
```

```
int lookotherthread(int x)
{
        int b,nohit=0;
        b=1<<asc;
        for(int j=0; j<threadno;j++)
        {
                if (j!=x)
                {
                        for(int i=0; i<b; i++)
                        {
                                if(cache[j][i][AddrIndexN].Valid==1)
                                {
                                        if(cache[j][i][AddrIndexN].Tag==AddrTagV)
                                        {
                                                lookthreadrecord[nohit].HTN=j;
                                                lookthreadrecord[nohit].HTB=i;

        lookthreadrecord[nohit].HTStatus=cache[j][i][AddrIndexN].Dirtybit;
                                                nohit++;
                                        }
                                }
                        }
                }
```

```
	}
	return nohit;
}
void Update_LRU(int threadnoo,int bank) //updating LRU value
{
	int a,c;
	a=cache[threadnoo][bank][AddrIndexN].LRU;
	c=1<<asc;
	for(int i=0; i<c; i++)
	{
		if(cache[threadnoo][i][AddrIndexN].LRU<a)
		{
			cache[threadnoo][i][AddrIndexN].LRU++;
		}
	}
	cache[threadnoo][bank][AddrIndexN].LRU=0;
}
void LRUF2(int threadnoo,int bank) //Decrease the LRU value
{
	int a,c;
	a=cache[threadnoo][bank][AddrIndexN].LRU;
	c=1<<asc;
	for(int i=0; i<c; i++)
```

```
        {
                if(cache[threadnoo][i][AddrIndexN].LRU>a)
                {
                        cache[threadnoo][i][AddrIndexN].LRU--;
                }
        }
        cache[threadnoo][bank][AddrIndexN].LRU=(c-1);
}
void Result_SINGLE()
{
        for(int j=0;j<threadno;j++)
        {
                cout<<"\n # Load  Instruction =  "<<cacherecord[j].noload;
                cout<<"\n # Store Instruction  =  "<<cacherecord[j].nowrite;
                cout<<"\n # Instruction                =  "<<cacherecord[j].noinst;
                cout<<"\n # AllInstruction      =
"<<(cacherecord[j].noinst+cacherecord[j].noload+cacherecord[j].nowrite);
                cout<<"\n # Data References   =
"<<(cacherecord[j].noload+cacherecord[j].nowrite);
                cout<<"\n # instruction misses         =  "<<cacherecord[j].IMcount;
                cout<<"\n # Data misses                =
"<<(cacherecord[j].RMcount+cacherecord[j].WMcount);
```

```
			cout<<"\n # inst miss Rate	=
"<<(cacherecord[j].IMcount/cacherecord[j].noinst*100);
			cout<<"\n # Data miss Rate	=
"<<((cacherecord[j].RMcount+cacherecord[j].WMcount)/(cacherecord[j].noload+cacherecord[j].
nowrite)*100);
			cout<<"\n # Global miss rate	=
"<<((cacherecord[j].RMcount+cacherecord[j].WMcount+cacherecord[j].IMcount)/(cacherecord[
j].noinst+cacherecord[j].noload+cacherecord[j].nowrite)*100);
			cout<<"\n";
	}
}
void Result_MESI_Or_SNOOPY()
{
	float totalall=0,totallmiss=0;
	float cachemissrate=0;
	double TC2C=0,TBSTRAFFIC=0,RHP=0,RHS=0,WHP=0,WHS=0,TEHR=0;
	double TSHR=0,TMHR=0,TEHW=0,TSHW=0,TMHW=0,totalmissrate=0;
	for(int j=0;j<threadno;j++)
	{
	totalmissrate=totalmissrate+(cacherecord[j].RMcount+cacherecord[j].WMcount)/(cacher
ecord[j].noload+cacherecord[j].nowrite)*100;
			cout<<"\n\tResult Core"<<j;
			cout<<"\n#READ\t\t\t"<<cacherecord[j].noload;
```

```
                cout<<"\nRDmiss\t\t\t"<<cacherecord[j].RMcount;
                cout<<"\n#Write\t\t\t"<<cacherecord[j].nowrite;
                cout<<"\nWTmiss\t\t\t"<<cacherecord[j].WMcount;
        cout<<"\nMissrate\t\t"<<(((cacherecord[j].RMcount+cacherecord[j].WMcount)/(cachere
cord[j].noload+cacherecord[j].nowrite))*100)<<" %";
                cout<<"\n#BusRds\t\t\t"<<cacherecord[j].BusRds;
                cout<<"\n#BusRdxs\t\t"<<cacherecord[j].BusRdxs;
                cout<<"\n#BusUpgrs\t\t"<<cacherecord[j].BusUpgrs;
                cout<<"\n#Cache-to-Cache\t\t"<<cacherecord[j].cache2cache;
                cout<<"\n#Read Exclusive Hit\t"<<cacherecord[j].ExcHitR;
                cout<<"\n#Read Shared Hit\t"<<cacherecord[j].ShrHitR;
                cout<<"\n#Read Modified Hit\t"<<cacherecord[j].ModHitR;
                cout<<"\n#Write Exclusive Hit\t"<<cacherecord[j].ExcHitW;
                cout<<"\n#Write Shared Hit\t"<<cacherecord[j].ShrHitW;
                cout<<"\n#Write Modified Hit\t"<<cacherecord[j].ModHitW;
        }
        cout<<"\n\tTotal";
        for(int j=0;j<threadno;j++)
        {
        TBSTRAFFIC=cacherecord[j].BusRds+cacherecord[j].BusRdxs+cacherecord[j].BusUpg
rs+TBSTRAFFIC;
                TC2C=TC2C+cacherecord[j].cache2cache;
                totalall=cacherecord[j].noload+cacherecord[j].nowrite+totalall;
                totallmiss=cacherecord[j].RMcount+cacherecord[j].WMcount+totallmiss;
                RHP=RHP+cacherecord[j].RHP;
                RHS=RHS+cacherecord[j].RHS;
                WHP=WHP+cacherecord[j].WHP;
                WHS=WHS+cacherecord[j].WHS;
                TEHR=TEHR+cacherecord[j].ExcHitR;
                TSHR=TSHR+cacherecord[j].ShrHitR;
                TMHR=TMHR+cacherecord[j].ModHitR;
                TEHW=TEHW+cacherecord[j].ExcHitW;
                TSHW=TSHW+cacherecord[j].ShrHitW;
                TMHW=TMHW+cacherecord[j].ModHitW;
        }
        cout<<"\nTotal BusTraffic=\t"<<TBSTRAFFIC;
        cout<<"\nTotal Cache-to-Cache=\t"<<TC2C;
        cout<<"\nTotal # Misses=\t\t"<<totallmiss;cachemissrate=totallmiss/totalall*100;
        cout<<"\nMisses/Accesses=\t"<<cachemissrate<<" %";
        cout<<"\nTotalMissRate=\t\t"<<totalmissrate<<" %";
        cout<<"\nTotal Exclusive Read Hit\t"<<TEHR;
        cout<<"\nTotal Shared Read Hit\t\t"<<TSHR;
        cout<<"\nTotal Modified Read Hit\t\t"<<TMHR;
        cout<<"\nTotal Exclusive Write Hit\t"<<TEHW;
        cout<<"\nTotal Shared Read Write\t\t"<<TSHW;
        cout<<"\nTotal Modified Read Write\t"<<TMHW;
```

```
	cout<<"\nRead-Hit(Private)\t\t"<<(TEHR+TMHR);
	cout<<"\nRead-Hit(Shared)\t\t"<<TSHR;
	cout<<"\nWrite-Hit(Private)\t\t"<<(TEHW+TMHW);
	cout<<"\nWrite-Hit(Shared)\t\t"<<TSHW;
	cout<<"\n\n";
}
void Result_FIREFLY()
{
	float totalall=0,totallmiss=0;
	float cachemissrate=0;
	double TC2C=0,TBSTRAFFIC=0,RHP=0,RHS=0,WHP=0,WHS=0,TEHR=0;
	double TSHR=0,TMHR=0,TEHW=0,TSHW=0,TMHW=0,totalmissrate=0;
	for(int j=0;j<threadno;j++)
	{
	totalmissrate=totalmissrate+(cacherecord[j].RMcount+cacherecord[j].WMcount)/(cacher
ecord[j].noload+cacherecord[j].nowrite)*100;
		cout<<"\n\tResult Core"<<j;
		cout<<"\n#READ\t\t\t"<<cacherecord[j].noload;
		cout<<"\nRDmiss\t\t\t"<<cacherecord[j].RMcount;
		cout<<"\n#Write\t\t\t"<<cacherecord[j].nowrite;
		cout<<"\nWTmiss\t\t\t"<<cacherecord[j].WMcount;
	cout<<"\nMissrate\t\t"<<(((cacherecord[j].RMcount+cacherecord[j].WMcount)/(cachere
cord[j].noload+cacherecord[j].nowrite))*100)<<" %";
```

```
        cout<<"\n#BusRds\t\t\t"<<cacherecord[j].BusRds;

        cout<<"\n#BusBusWrs\t\t"<<cacherecord[j].BusWrs;

        cout<<"\n#Cache-to-Cache\t\t"<<cacherecord[j].cache2cache;

        cout<<"\n#Read Valid Hit\t"<<cacherecord[j].ExcHitR;

        cout<<"\n#Read Shared Hit\t"<<cacherecord[j].ShrHitR;

        cout<<"\n#Read Dirty Hit\t"<<cacherecord[j].ModHitR;

        cout<<"\n#Write Valid Hit\t"<<cacherecord[j].ExcHitW;

        cout<<"\n#Write Shared Hit\t"<<cacherecord[j].ShrHitW;

        cout<<"\n#Write Dirty Hit\t"<<cacherecord[j].ModHitW;

}

cout<<"\n\tTotal";

for(int j=0;j<threadno;j++)

{

        TBSTRAFFIC=cacherecord[j].BusRds+cacherecord[j].BusWrs+TBSTRAFFIC;

        TC2C=TC2C+cacherecord[j].cache2cache;

        totalall=cacherecord[j].noload+cacherecord[j].nowrite+totalall;

        totallmiss=cacherecord[j].RMcount+cacherecord[j].WMcount+totallmiss;

        RHP=RHP+cacherecord[j].RHP;

        RHS=RHS+cacherecord[j].RHS;

        WHP=WHP+cacherecord[j].WHP;

        WHS=WHS+cacherecord[j].WHS;

        TEHR=TEHR+cacherecord[j].ExcHitR;

        TSHR=TSHR+cacherecord[j].ShrHitR;
```

```
                TMHR=TMHR+cacherecord[j].ModHitR;

                TEHW=TEHW+cacherecord[j].ExcHitW;

                TSHW=TSHW+cacherecord[j].ShrHitW;

                TMHW=TMHW+cacherecord[j].ModHitW;

        }

        cout<<"\nTotal BusTraffic=\t"<<TBSTRAFFIC;

        cout<<"\nTotal Cache-to-Cache=\t"<<TC2C;

        cout<<"\nTotal # Misses=\t\t"<<totallmiss;cachemissrate=totallmiss/totalall*100;

        cout<<"\nMisses/Accesses=\t"<<cachemissrate<<" %";

        cout<<"\nTotalMissRate=\t\t"<<totalmissrate<<" %";

        cout<<"\nTotal Valid Read Hit\t"<<TEHR;

        cout<<"\nTotal Shared Read Hit\t\t"<<TSHR;

        cout<<"\nTotal Dirty Read Hit\t\t"<<TMHR;

        cout<<"\nTotal Valid Write Hit\t"<<TEHW;

        cout<<"\nTotal Shared Read Write\t\t"<<TSHW;

        cout<<"\nTotal Dirty Read Write\t"<<TMHW;

        cout<<"\nRead-Hit(Private)\t\t"<<(TEHR+TMHR);

        cout<<"\nRead-Hit(Shared)\t\t"<<TSHR;

        cout<<"\nWrite-Hit(Private)\t\t"<<(TEHW+TMHW);

        cout<<"\nWrite-Hit(Shared)\t\t"<<TSHW;

}
```

REFERENCES

[1] Miguel A. Vega Rodriguez, et al., Simulation of Cache Memory Systems on Symmetric Multiprocessors with Educational Purposes, Proc. of the I International Congress in Quality and in Technical Education Innovation, vol. III, pp. 47-59. Donostia-San Sebastián, Spain. 4-6 September 2000.

[2] McCurdy, C. and Fischer, C. “Using Pin as a memory reference generator for multiprocessor simulation,” *SIGARCH Computer Architecture News* 33, 5 (Dec. 2005), 39-44.

[3] Aamer Jaleel, Robert S. Cohn, Chi-Keung Luk, and Bruce Jacob. "CMP$im: A Pin-Based On-The-Fly Multi-Core Cache Simulator”, In the Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA'2008.

[4] Paruj Ratanaworabhan, “Functional Cache Simulator for Multicore, Proceedings of the 9th Electrical Engineering /Electronics, Computer, Telecommunications, and Information Technology (ECTI-CON), May 16-18, 2012

[5] R. A. Uhlig. And T. N. Mudge. “Trace-driven Memory Simulation: A Survey”, In ACM Computing Surveys, Vol. 29, 1997.

[6] J. Edler and M. D. Hill. “Dinero IV Trace-Driven Uniprocessor Cache Simulator [online],” Available: https://scholar.google.com/citations?user=7lVfIWYAAAAJ

[7] D. Burger, and T. Austin, “The SimpleScalar Tool Set, Version 2.0,” University of Wisconsin-Madison Computer Sciences Department Technical Report #1242, June 1997.[8] Standard Performance Evaluation Corporation, “SPEC CPU 2006 [online],” Available: https://www.spec.org/cpu2006.

[9] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi and K. Hazelwood, Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation, ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Chicago, IL, USA, 2005.

[10] Rafael Ubal, et al., “Multi2Sim: A Simulation Framework for CPU-GPU Computing,” the 21st IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT), Minneapolis, MN, USA, pp. September 2012.

[11] Charles Price. *MIPS IV Instruction Set, revision 3.1*. MIPS Technologies, Inc., Mountain View, CA, January 1995.

[12] Naraig Manjikian, "Enhancements and Applications of the SimpleScalar Simulator for Undergraduate and Graduate Computer Architecture Education," Department of Electrical and Computer Engineering Queen's University Kingston, Ontario, Canada K7L 3N6.

[13] D. Madon, E. Sanchez, and S. Monnier. A Study of a Simultaneous Multithreaded Processor Implementation. In European Conference on Parallel Processing, pages 716–726, 1999.

[14] J. Sharkey. M-Sim: A Flexible, Multithreaded Architectural Simulation Environment. Technical Report CS-TR-05-DP01, Department of Computer Science, State University of New York at Binghamton, 2005.

[15] D. M. Tullsen. Simulation and Modeling of a Simultaneous Multithreading Processor. 22nd Annual Computer Measurement Group Conference, December 1996.

[16] Jianyun Cheng, Zhenzhou Ji, Binjie Zhang "an optimized method of memory simulation accuracy in multicore multithread processor" Department of Computer Science and Engineering Harbin Institute of Technology 150001 Harbin, China.

[17] B. Lee and D. Brooks. Effects of Pipeline Complexity on SMT/CMP Power-Performance Efficiency. Workshop on Complexity Effective Design, 2005.

[18] J. Hennessy and D. A. Patterson, *Computer Architecture – A Quantitative Approach*, Waltham, MA, USA: Morgan Kaufmann, Elsevier, 2012.

[19] C. Bienia, S. Kumar, and K. Li. PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors. In Proceedings of IISWC 2008, pages 47–56, Sept. 2008.

[20] Arun A. Nair and Lizy K. John , Simulation Points for SPEC CPU 2006, Proceedings of 2008 IEEE International Conference on Computer Design.

[21] Marco Elver and Vijay Nagarajan, TSO-CC: Consistency directed cache coherence for TSO, Proceedings of 2014 IEEE 20th International Symposium on high Performance Computer Architecture (HPCA), Orlando, Florida, U.S.A., 2014.

[22] Marco Elver and Vijay Nagarajan, RC3: Consistency directed cache coherence for x86-64 with RC extensions, Proceedings of 2015 IEEE International Symposium on Parallel Architecture and Compilation Techniques (PACT), Washington, DC, U.S.A., 2015.

[23] A. Seznec, A case for 2-way skewed-associativity cache, the 20th International Symposium on Computer Architecture (ISCA), San Diego, USA, May 1993.

[24] Hashemi, B. (2011-05-01). "Simulation and Evaluation Snoopy Cache Coherence Protocols with Update Strategy in Shared Memory Multiprocessor Systems". *2011 Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications*

Workshops (ISPAW).

BIOGRAPHICAL SKETCH

Rano Mal was born in 1989. He has done his undergraduate in Electrical (Telecommunication) Engineering from NUST (National University of Science & Technology), Rawalpindi Pakistan in July 2013. He finished his Master of Science in Electrical Engineering from The University of Texas Rio Grande Valley, Edinburg, Texas, US in July 2017. Rano has published one paper based on his thesis work. His research interests lie in high performance computer architecture, simulator design, and cluster architecture. His email address is rano.mal01@utrgv.edu

Permanent address: 735 Marsh Point RD, Evans, GA 30809, USA.