CMPT 450/750: Computer Architecture
Fall 2024

Memory Ordering

Alaa Alameldeen & Arrvindh Shriraman

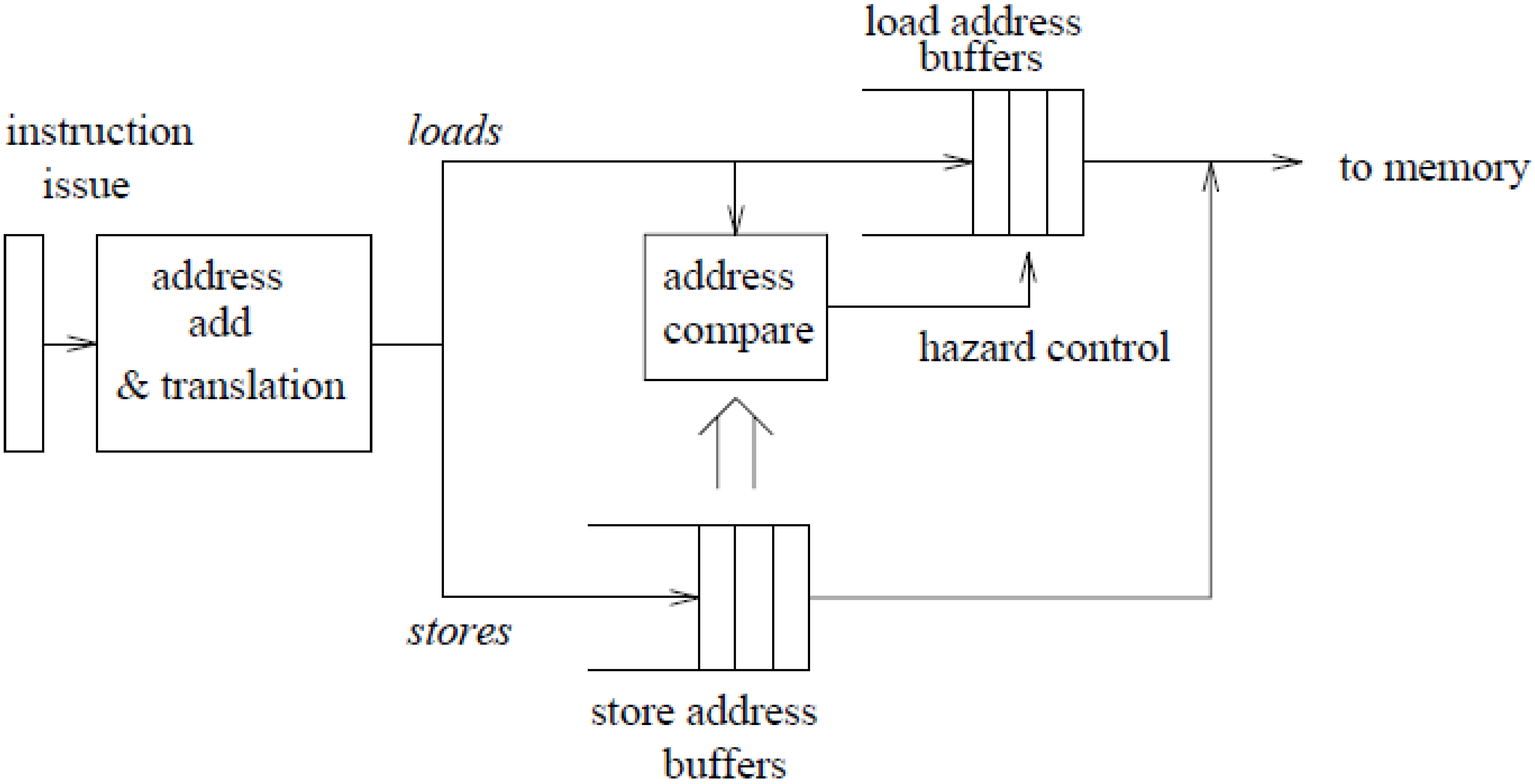
Handling Memory Operations

- **Some instructions (loads, stores) have memory operands, and need to access the memory hierarchy**
    - ➢ Accesses performed in the “memory” stage of the superscalar processor pipeline
- **To maximize ILP, loads/stores may execute out of order**
- **Memory operations require special handling**
    - ➢ Recall that Register dependences are identified at decode time
        - ❑ Allows early renaming to remove false dependences
        - ❑ Maximizes ILP
    - ➢ But Memory dependences cannot be determined before execution since memory addresses need to be computed first
    - ➢ False dependences exist in memory execution stream
        - ❑ For example, multiple stores to the same bytes (WAW)
        - ❑ Frequent due to stack pushes and pops

How Processors Execute Memory Operations

- **Functions that a processor has to do for memory operations:**
  - ➢Enforcing memory dependences among loads and stores
  - ➢Stores/Writes to memory need to be **ordered and non-speculative**
  - ➢Stores issue to the data cache after retirement
- **Loads and stores ordering is enforced using load and store buffers while allowing out of order execution**
- **Stores consist of address and data uops. Store addresses are buffered in a queue**
- **Store addresses remain buffered until:**
  - ➢Store data is available AND Store instruction is committed in the reorder buffer
- **New load addresses are checked with the waiting store addresses. If there is a match:**
  - ➢The load waits OR Store data is bypassed to the matching load

Load and Store Buffers



Smith & Sohi 1995, Figure 11

Store Buffer

- **Store addresses are buffered in a queue**
  - ➢Entries allocated in program order at rename
- **Store addresses remain buffered until:**
  - ➢Store data is available
  - ➢Store instruction is retired in the reorder buffer
- **New load addresses are checked with the waiting older store addresses**
  - ➢If there is a match:
    - ❑The load waits OR
    - ❑Store data is bypassed to the matching load
  - ➢If there is no match:
    - ❑Load can execute speculatively OR
    - ❑Load waits till all prior store addresses are computed

Store Buffer Operation

- **To match loads with store addresses, the store buffer is typically organized as a fully-associative queue (could also be set-associative)**
  - ➢An address comparator in each buffer entry compares each store address to the address of a load issued to the data cache
  - ➢Multiple stores to same address may be present
  - ➢Load-store address match is qualified with “age” information to match a load to the last preceding store in program order
    - ❑Age is typically checked by attaching the number of the preceding store entry to each load at rename
    - ❑Effectively provides a form of “memory renaming”

Load Buffer

- **Loads can speculatively issue to the data cache out-of-order**
- **But load issue may stall**
  - ➢Unavailable resources/ports in the data cache
  - ➢Unknown store addresses
  - ➢Delayed store data execution
  - ➢Memory mapped I/O
  - ➢Lock operations
  - ➢Misaligned loads

Load Buffer Operation

- **Load buffer is provided for stalled loads to wait**
- **Load entries are allocated in program order at rename**
- **A stalled load simply waits in its buffer entry until stall condition is removed**
  - ➢Scheduling logic checks for awakened loads and re-issues them to the data cache, typically in program order

Load-Store Dependence Speculation

- **Load buffers are sometimes used for load-store dependence speculation**
  - ➢When a load issues ahead of a preceding store, it is impossible to perform address match
  - ➢Option 1: Stall the load until all prior store addresses are computed
    - ❑Significant performance impact in machines with deep, wide pipelines
  - ➢Option 2: Speculate that the load does not depend on the previous unknown stores
    - ❑Needs misprediction detection and recovery mechanism
- **One detection mechanism is to make the load buffer a fully associative queue**
  - ➢Store addresses are checked against all previously issued load addresses
  - ➢Only younger loads need to be checked

Memory Consistency

- **Load buffer snoops other processors' stores to maintain memory consistency on some processors**
- **Stores from other threads on the same processor need to be snooped in the load buffer to maintain memory consistency**
- **Memory consistency models define ordering requirements of loads and stores to different addresses and from multiple processors**
  - ➢Discussed later in the course
- **Examples for memory consistency models**
  - ➢Sequential Consistency (SC)
  - ➢Total Store Order (TSO)

Memory Dependence Prediction

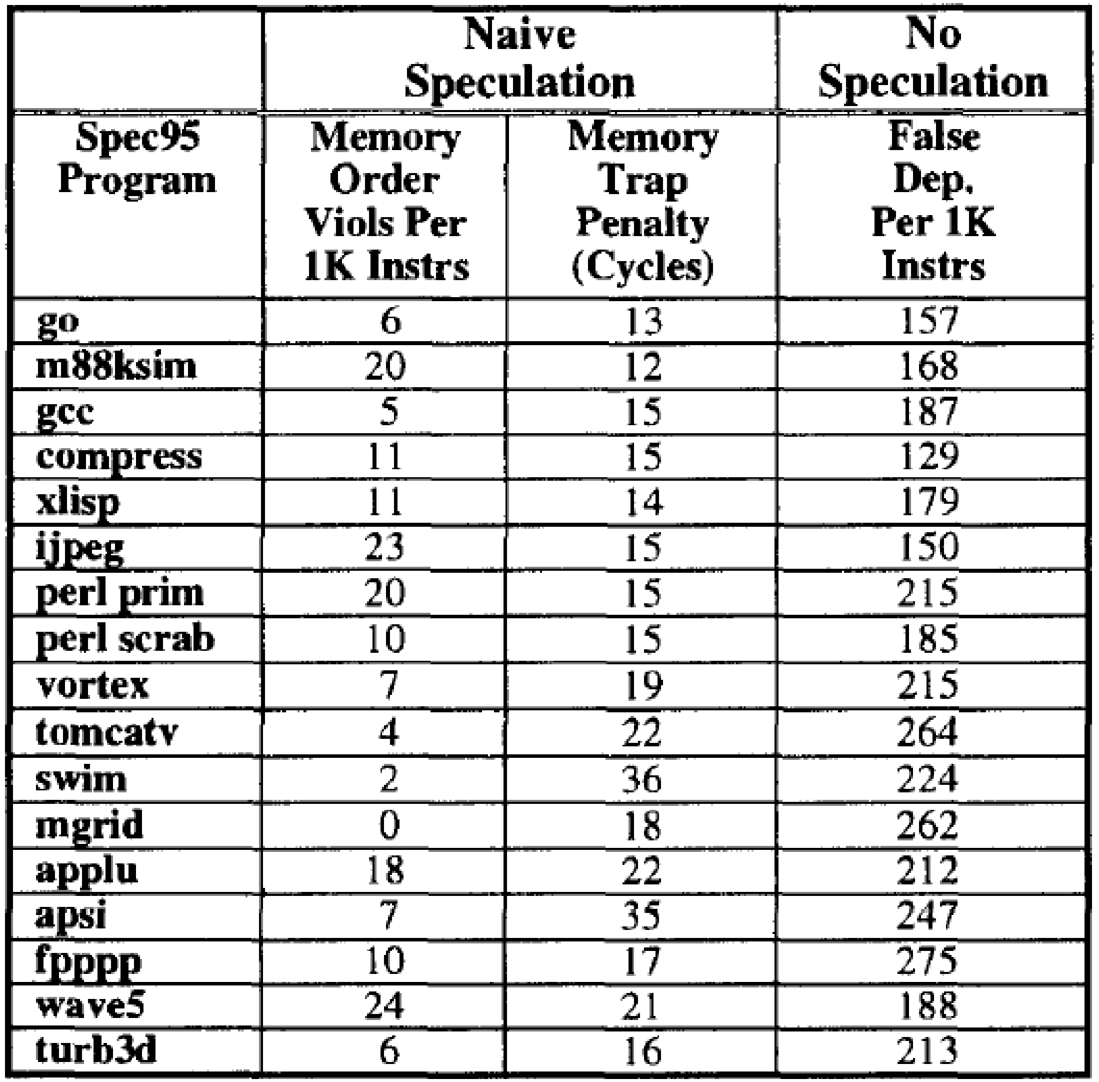
- **Memory Order Violation: A load is executed before an older store, reads the wrong value**
- **False Dependence: Loads wait unnecessarily for stores to different addresses**
- **Goals of Memory Dependence Prediction:**
  - ➢Predict the load instructions that would cause a memory-order violation
  - ➢Delay execution of these loads only as long as necessary to avoid violations

Memory Dependence Prediction

- **Memory misprediction recovery is expensive**
  - ➢Requires a pipeline flush if a store address matches a younger issued load
  - ➢Compare to branch mispredictions
- **Memory dependence prediction minimizes such mispredictions**
  - ➢Issue younger loads if predictor predicts “no dependence”
  - ➢Stall younger loads if predictor predicts “dependence”
  - ➢With a good predictor, this approach minimizes unnecessary stalls (false dependences) as well as pipeline flushes from mispredictions

Alternatives to Memory Dependence Prediction

- **No Speculation**
  - ➢Issue for any load waits till prior stores have issued
- **Naïve Speculation**
  - ➢Always issue and execute loads when their register dependences are satisfied, regardless of memory dependences
- **Statistics for some benchmarks:**

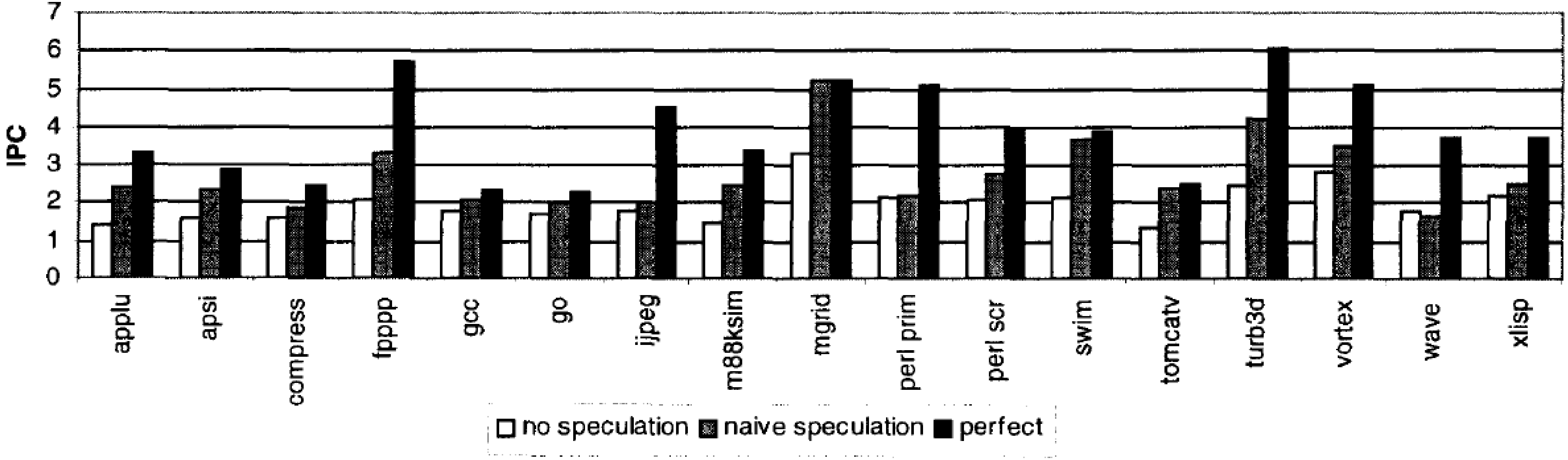


	Naive Speculation		No Speculation
Spec95 Program	Memory Order Viols Per 1K Instrs	Memory Trap Penalty (Cycles)	False Dep. Per 1K Instrs
go	6	13	157
m88ksim	20	12	168
gcc	5	15	187
compress	11	15	129
xlisp	11	14	179
ijpeg	23	15	150
perl prim	20	15	215
perl scrab	10	15	185
vortex	7	19	215
tomcatv	4	22	264
swim	2	36	224
mgrid	0	18	262
applu	18	22	212
apsi	7	35	247
fpppp	10	17	275
wave5	24	21	188
turb3d	6	16	213

Chrysos&Emer, 1998, Table 3.1

Perfect Memory Dependence Prediction

- Does not cause memory order violations
- Avoids all false dependences



Chrysos&Emer, 1998, Figure 3.1

Store Sets

- **Based on the assumption that future dependences can be predicted from past behavior**
- **Each load has a store set consisting of all stores upon which it has ever depended**
    - ➢Store is identified by its PC
- **When program starts, all loads have empty store sets**
- **When a memory order violation happens, store is added to load’s store set**

Store Set Example

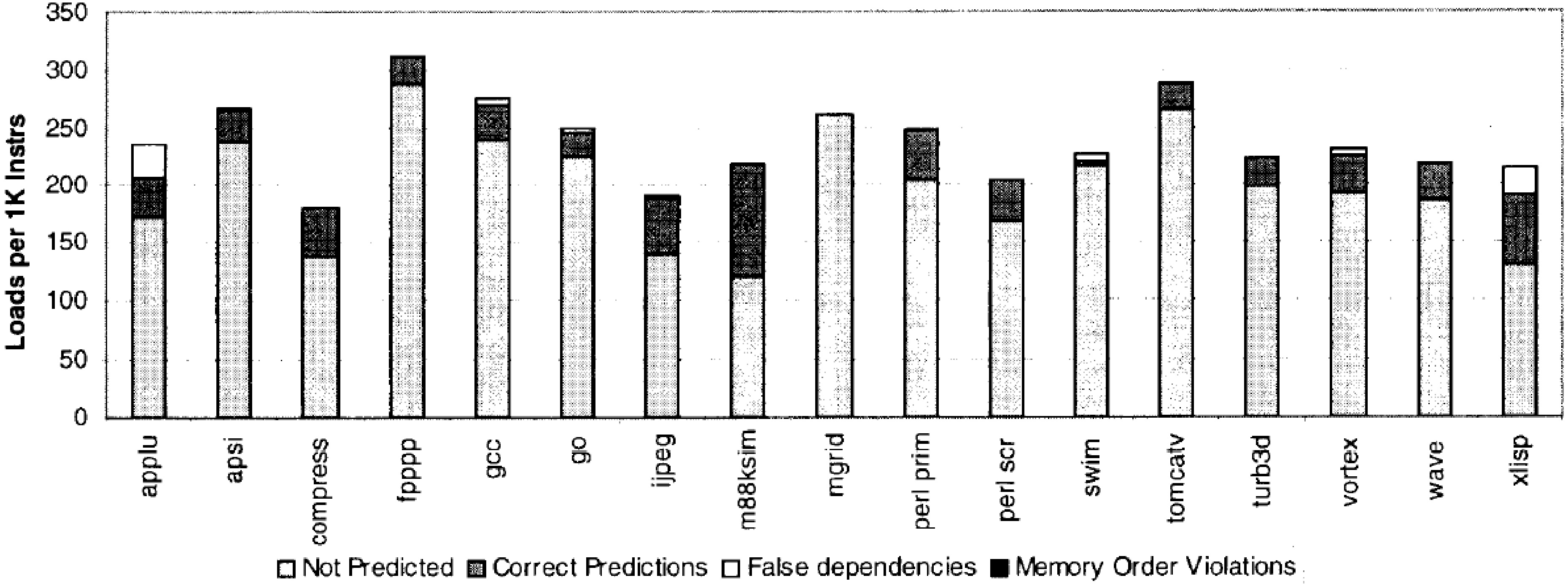
PC	Inst	
0	Store C	
4	Store A	
8	Store B	
12	Store C	
…		
28	Load B	SS = {PC 8}
32	Load D	SS = { }
36	Load C	SS = {PC 0, PC 12}
40	Load A	SS = {PC 4}

Store Set Performance

- **Infinite SS configuration (#sets, #elements/set are not limited)**
- **Each dynamic load is classified as:**
  - Not predicted (loads with empty store sets)
  - Correctly predicted
  - False dependence (unnecessary wait)
  - Memory order violation (dependence not predicted)

Chrysos&Emer, 1998, Figure 5.2

Figure 5.2: Infinite Store Sets Predictor - Dynamic Load Breakdown

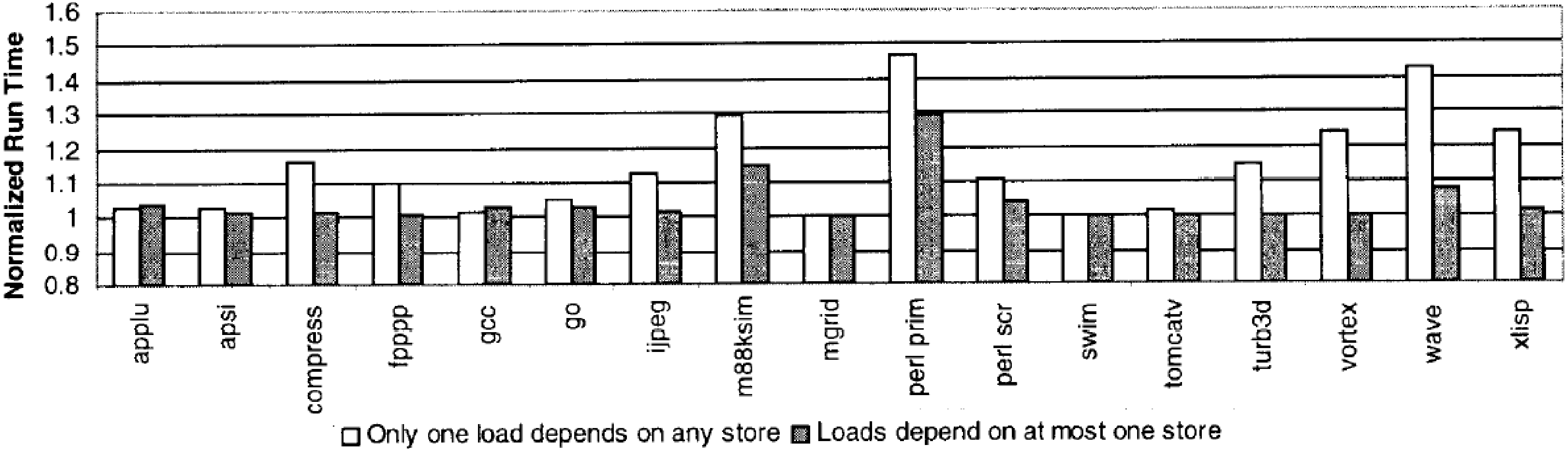


More Practical Store Set Performance

- **Hardware resources are not infinite, so we cannot allow infinitely large store sets per load**
- **Results when limiting store sets:**
  - ➢At most one load can depend on any store
  - ➢Each load depends on at most one store

Chrysos&Emer, 1998, Figure 5.1

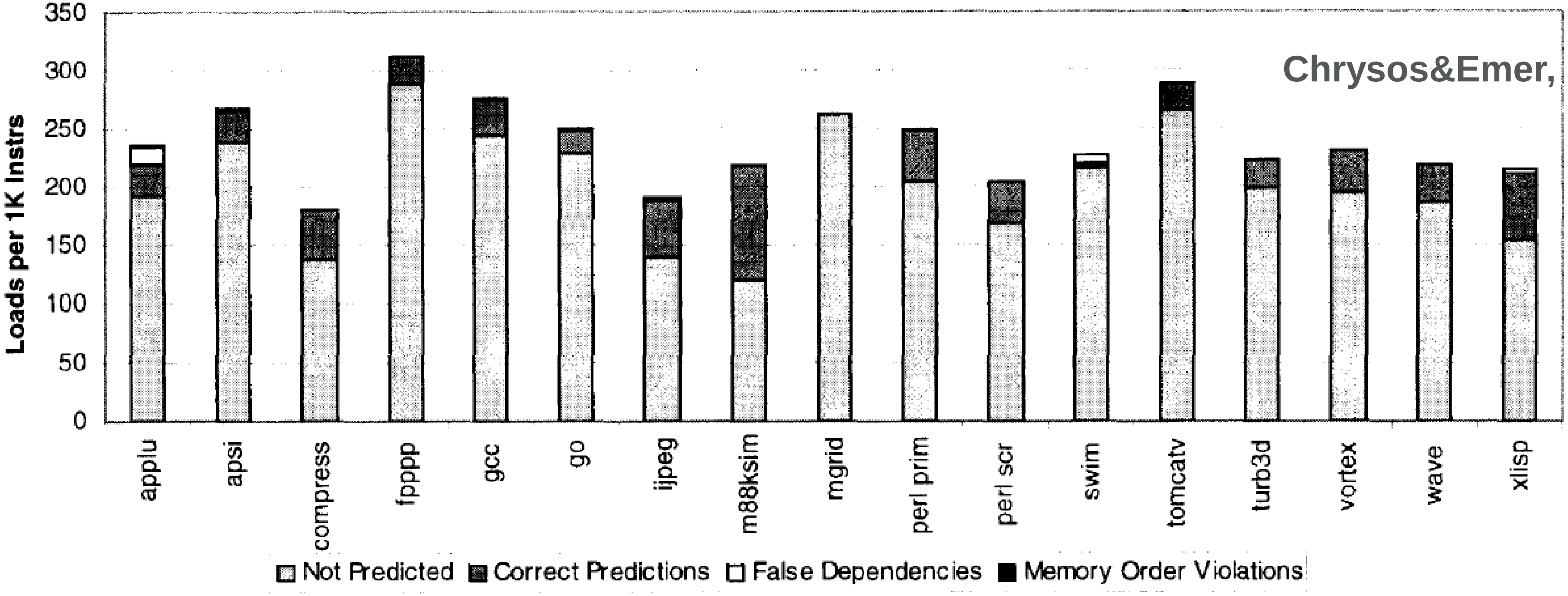
Figure 5.1: Effect on Run Time When Not Allowing Multiple Dependence Flexibility



Reducing False Dependences

- **With infinite SS, a store dependence remains in a load's store set forever, even if some dynamic instances of the load are independent**
- **To reduce false dependences, we can use 2-bit saturating counters**
  - ➢ Set to max value (3) on a memory order violation
  - ➢ Decremented if real dependence doesn't exist, incremented if real dependence exists
  - ➢ Counter values of 2 or 3 cause load to wait; otherwise no dependence is assumed

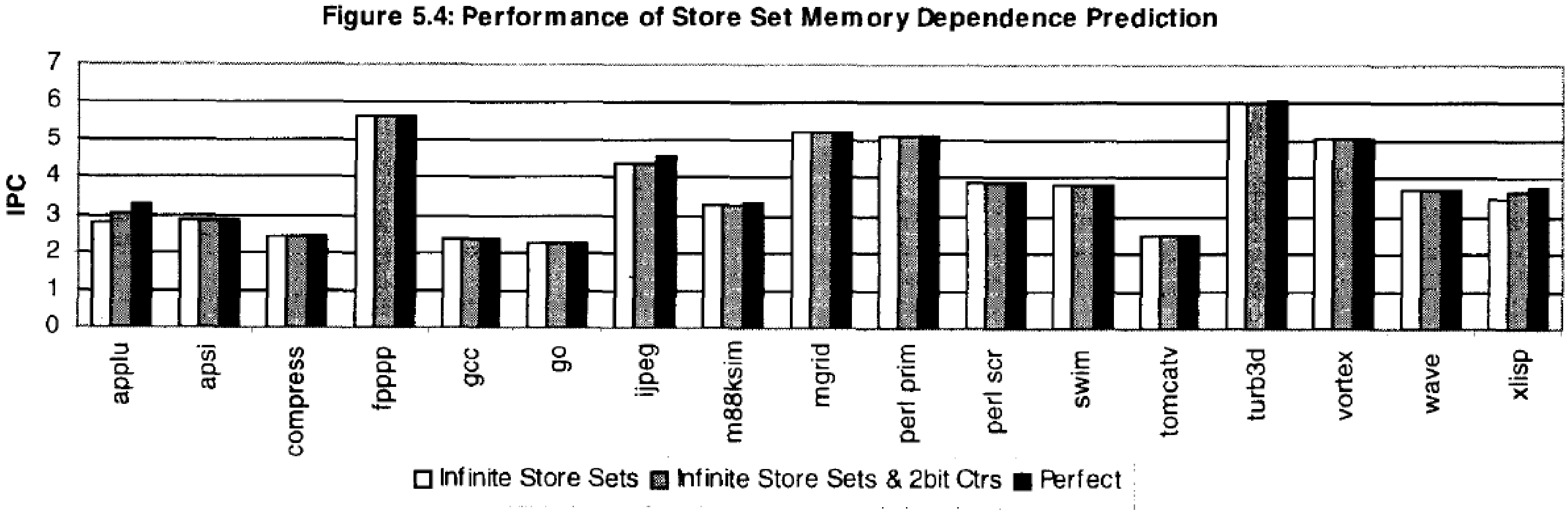
Figure 5.3: Infinite Store Sets Predictor with 2 Bit Counters - Dynamic Load Breakdown



Chrysos&Emer, 1998, Figure 5.3

Store Set Comparison to Perfect Prediction

Chrysos&Emer, 1998, Figure 5.4



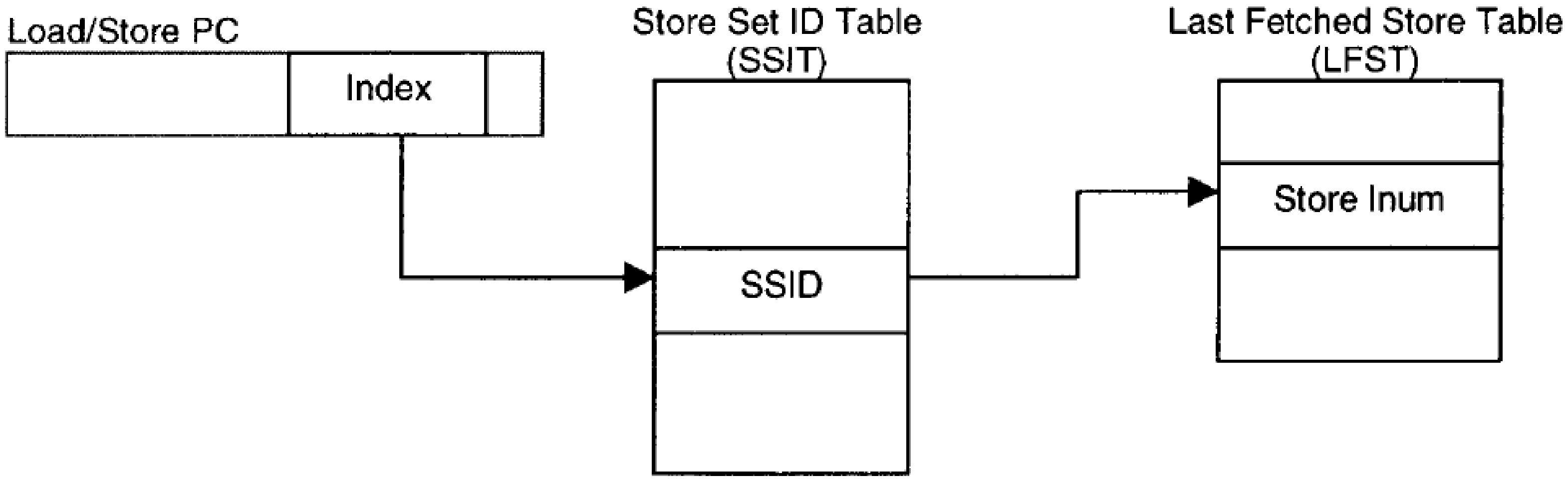
- Infinite SS with 2-bit saturating counters are very close to perfect memory dependence prediction

Practical Store Set Implementation

- **Store Set Identifier Table (SSIT): PC-indexed, maintains store sets**
- **Last Fetched Store Table (LFST) maintains dynamic inst. count about most recently fetched store for each store set**

Chrysos&Emer, 1998, Figure 6.1

Figure 6.1: Implementation of Store Sets Memory Dependence Prediction



Loads and stores index into the SSIT to get their store set identifiers, which are used to access and update the LFST. The store inums that are found in the LFST indicate the memory dependence prediction.

- **Limitations:**
  - Store PCs exist in one store set at a time
  - Two loads depending on the same store can share a store set
  - All stores in a store set are executed in order

Implementation Details

- **Recently fetched loads**
  - ➢Access SSIT based on their PC, get their SSID
  - ➢If SSID is valid, LFST is accessed to get most recent store in the load’s store set
- **Recently fetched stores**
  - ➢Access SSIT based on their PC
  - ➢If SSID is valid, then store belongs to a valid store set
    - ❑Access LFST to get most recently fetched store information in its store set
    - ❑Update LFST inserting its own dynamic inst. count since it is now the last fetched store in that store set
    - ❑After store is issued, it invalidates the LFST entry if it refers to itself to ensure loads & stores are only dependent on stores that haven’t been issued

Store Set Interference

- **Destructive interference happens because stores can belong to only one store set**

Example:

Load PC 1 → Store Set 1 { Store PC X, Store PC Y, Store PC Z }

Load PC 2 → Store Set 2 { Store PC J, Store PC K }

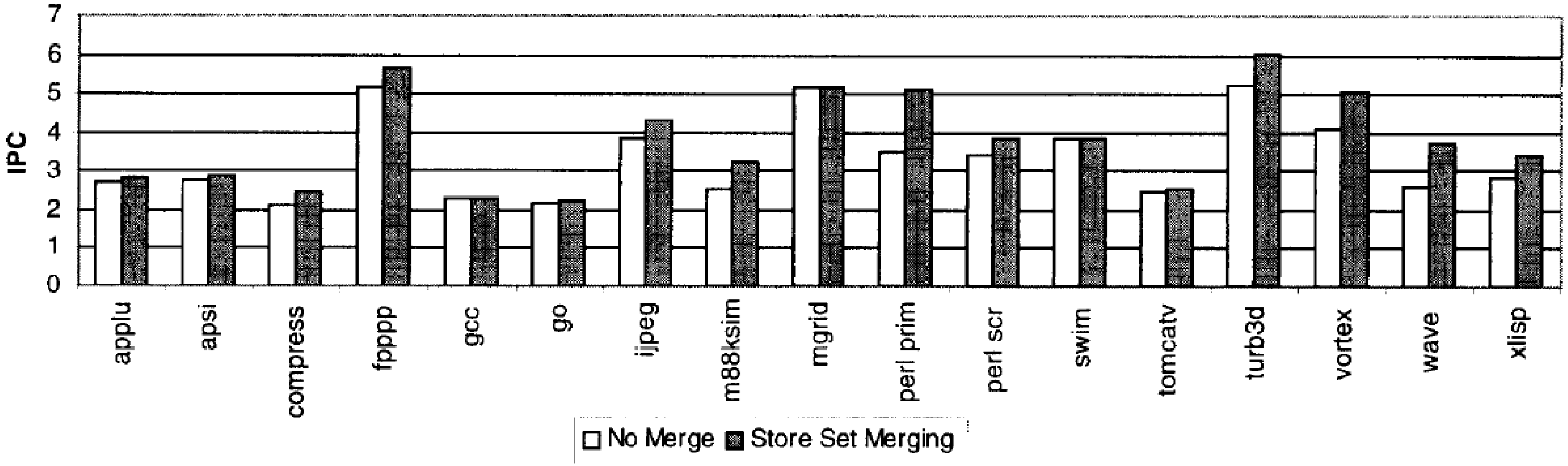
- **Assume that Load PC 1 has a memory order violation with Store PC J**
  - Each store can exist in one SS, so we need to remove Store PC J from SS 2 and add it to SS 1
  - But this causes future memory order violation between Load PC 2 and Store PC J
- **Store set merging avoids the problem**

Store Set Merging

- **When a store-load pair causes a memory order violation:**
  - ➢ If neither has been assigned a store set, a store set is allocated and assigned to both instructions
  - ➢ If load has been assigned a store set but the store hasn’t, the store is assigned the load’s store set
  - ➢ If store has been assigned a store set but the load hasn’t, the load is assigned the store’s load set
  - ➢ If both have store sets, one of them is declared the winner, and the instruction belonging to the loser’s store set is assigned the winner’s store set

Chrysos&Emer, 1998, Figure 6.2

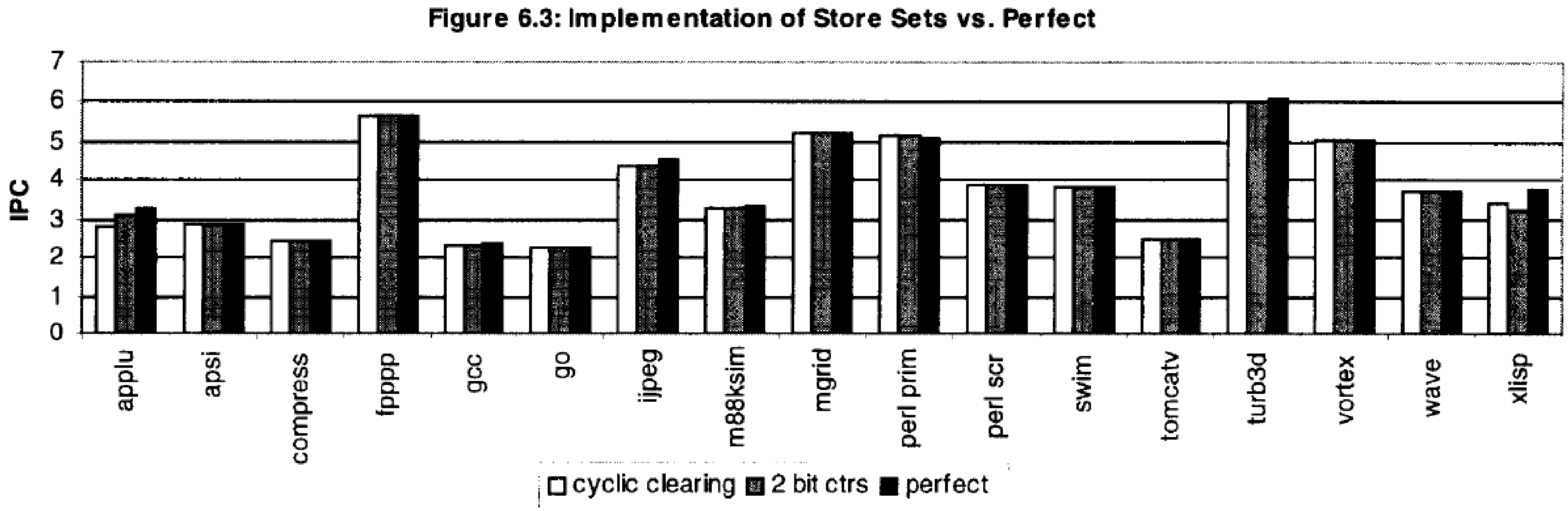
Figure 6.2: Performance Improvement Due to Store Set Merging



Store Set Performance

- For the practical SS implementation, cyclic clearing of valid bits (every ~1M cycles) is almost the same as 2-bit saturating counters
- With sufficiently large structures, performance very close to perfect prediction

Chrysos&Emer, 1998, Figure 6.3



Announcements

- **Reading Assignment**
  - ➢G.Z. Chrysos and J.S. Emer, “Memory Dependence Prediction using Store Sets,” ISCA 1998 (Read).
- **Assignment 1 due Friday Sep 27 @11:59 PM**
- **Exam Logistics**
  - ➢Exam 1 is on Tuesday Oct 1 during class time (1:30-2:20 PM)
  - ➢Open book, notes, calculator
  - ➢Exam will be available on the course canvas page. Link active during class time
  - ➢You need to join the zoom link and turn your camera on
    - ❑Zoom link will be sent on Piazza the day of the exam
  - ➢Attendance will be taken on exam days. You need to be on zoom for your exam to count.