



# Frequent Value Locality and its Applications

**Rajiv Gupta**

**Jun Yang**

**The University of Arizona  
Department of Computer Science**



# Frequent Value Locality

- ❑ **Small number of values occupy a large fraction of memory locations in use**  
⇒ **frequent values.**
- ❑ **Frequent values remain the same for long periods of time during execution**  
⇒ **they can be identified and exploited.**
- ❑ **Frequent values are distributed quite uniformly in memory**  
⇒ **they are encountered in all parts of the memory.**

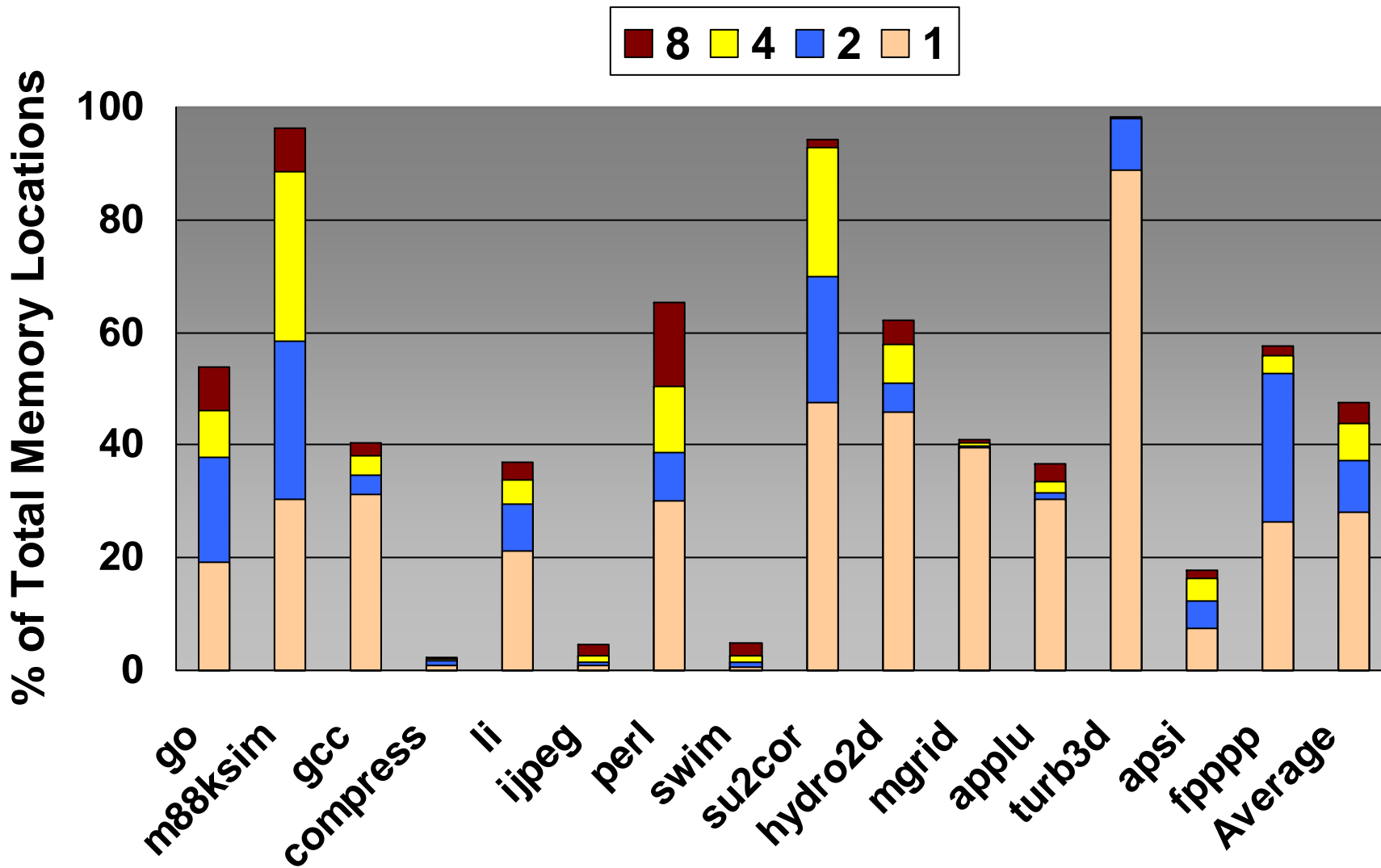


# Frequent Value Locality

- **Small number of values occupy a large fraction of memory locations in use**  
⇒ **frequent values.**
- Frequent values remain the same for long periods of time during execution  
⇒ they can be identified and exploited.
- Frequent values are distributed quite uniformly in memory  
⇒ they are encountered in all parts of the memory.

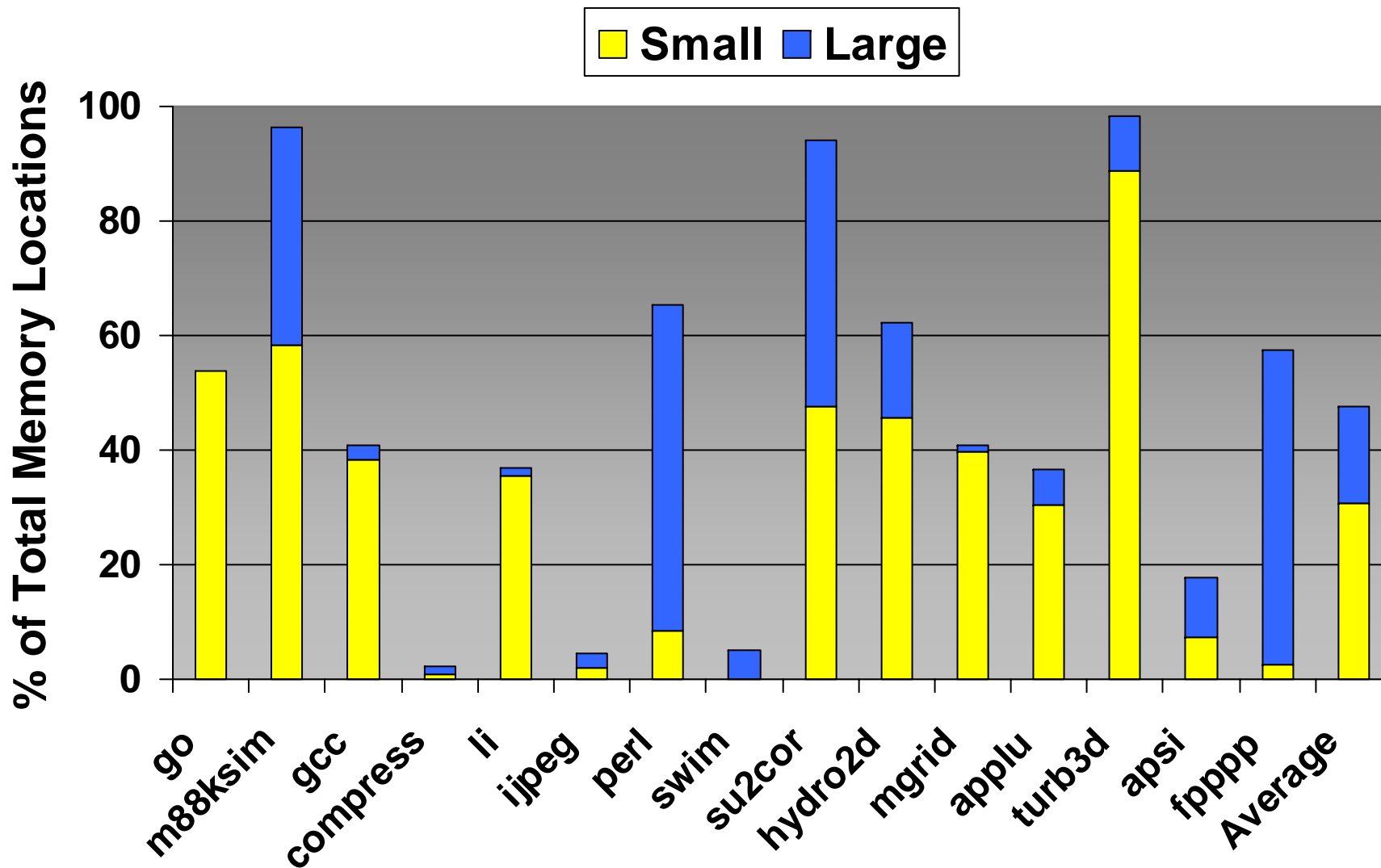


# Frequent Values in Spec95





# Frequent Values in Spec95



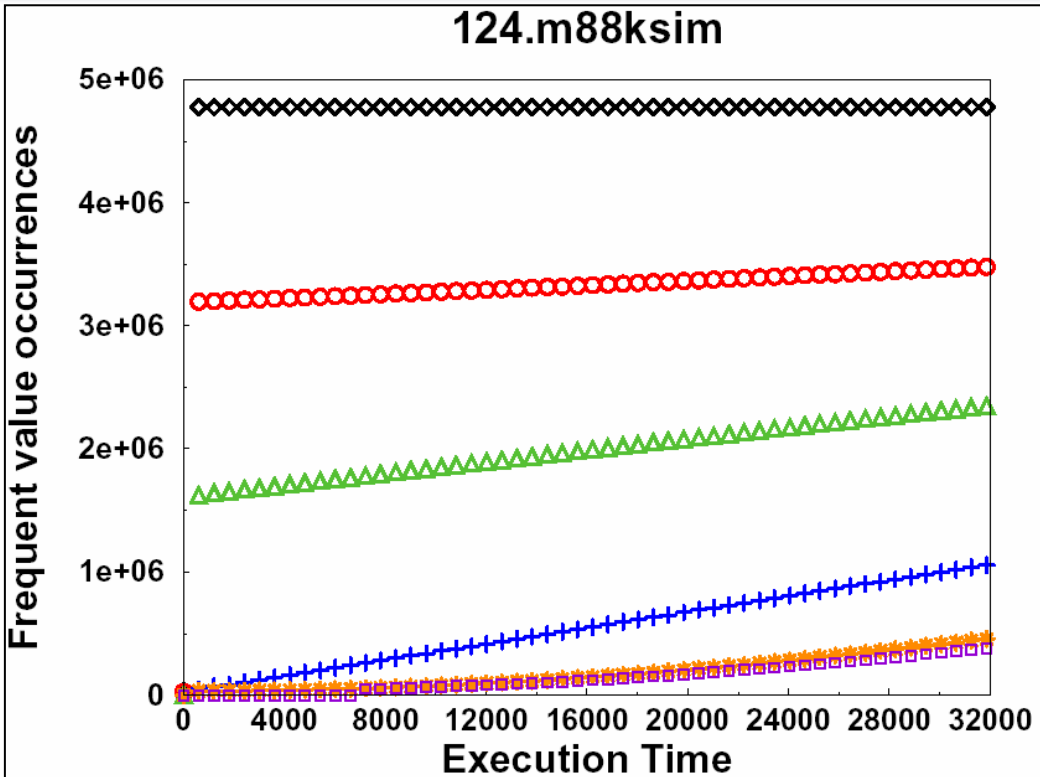


# Frequent Value Locality

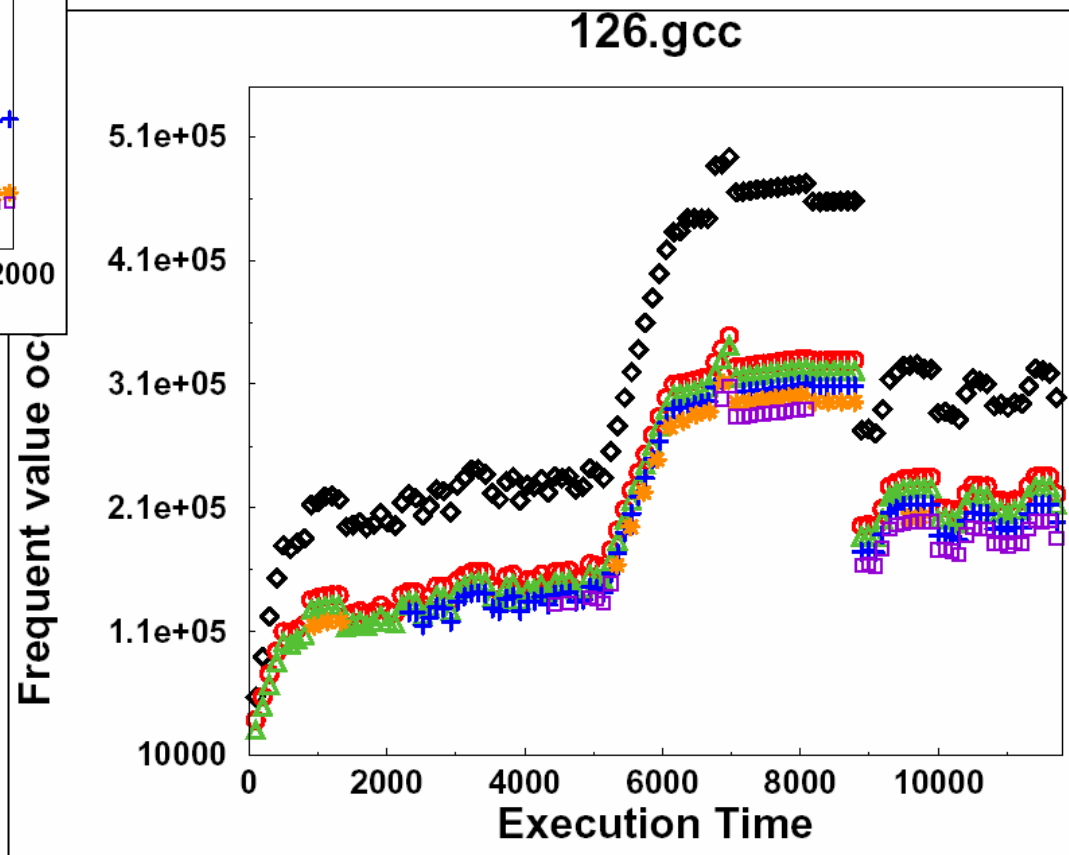
- Small number of values occupy a large fraction of memory locations in use  
⇒ frequent values.
- **Frequent values remain the same for long periods of time during execution**  
⇒ **they can be identified and exploited.**
- Frequent values are distributed quite uniformly in memory  
⇒ they are encountered in all parts of the memory.



# Frequent Values Over Program Execution

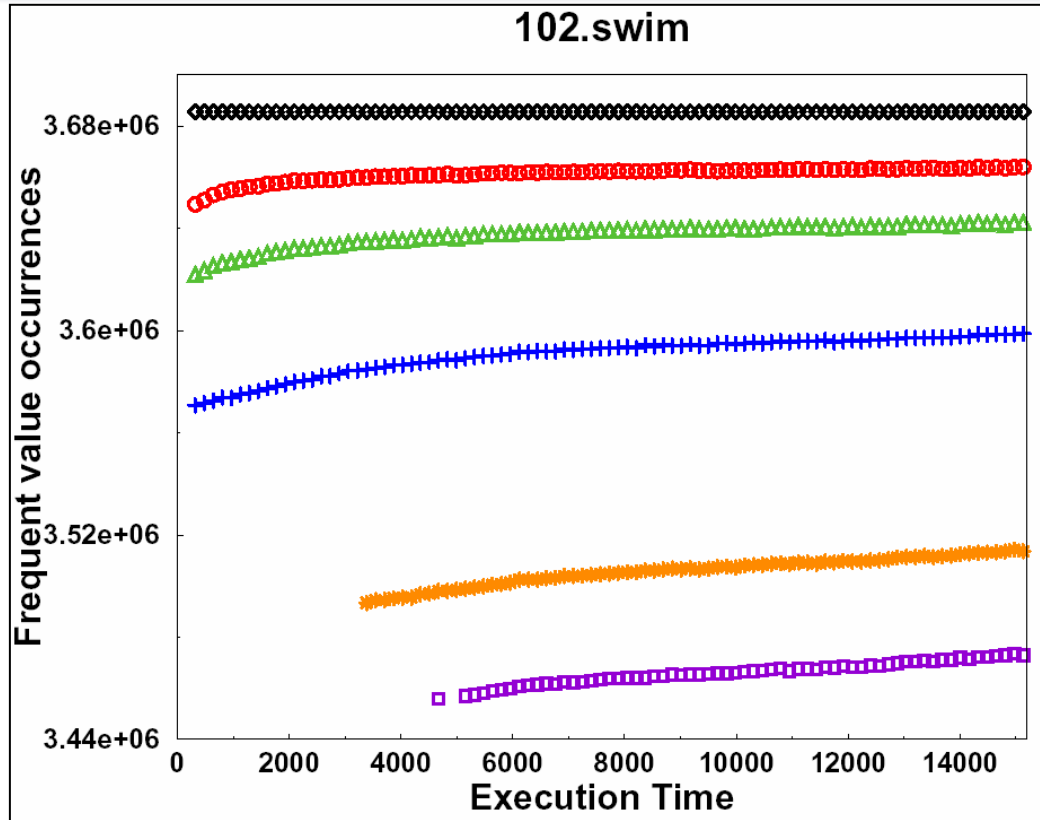


- ◇ Total
- Top 1 value
- △ Top 2 values
- + Top 4 values
- \* Top 8 values
- Top 10 values

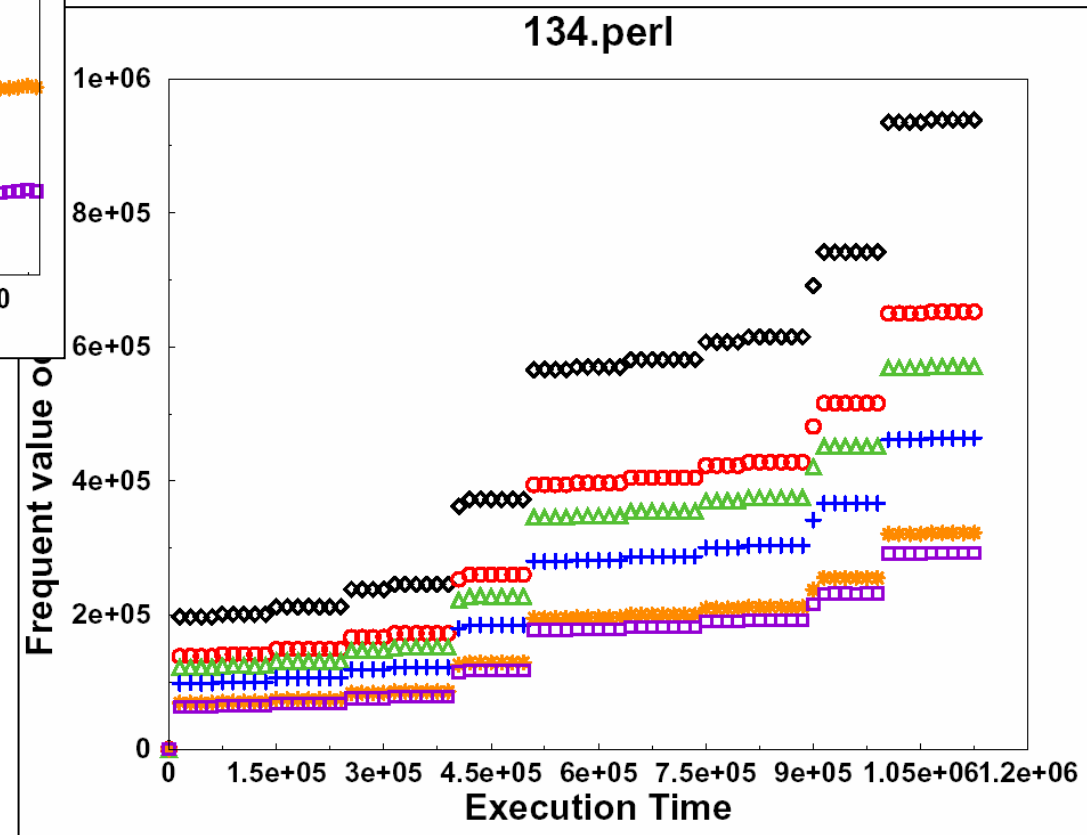




# Frequent Values Over Program Execution



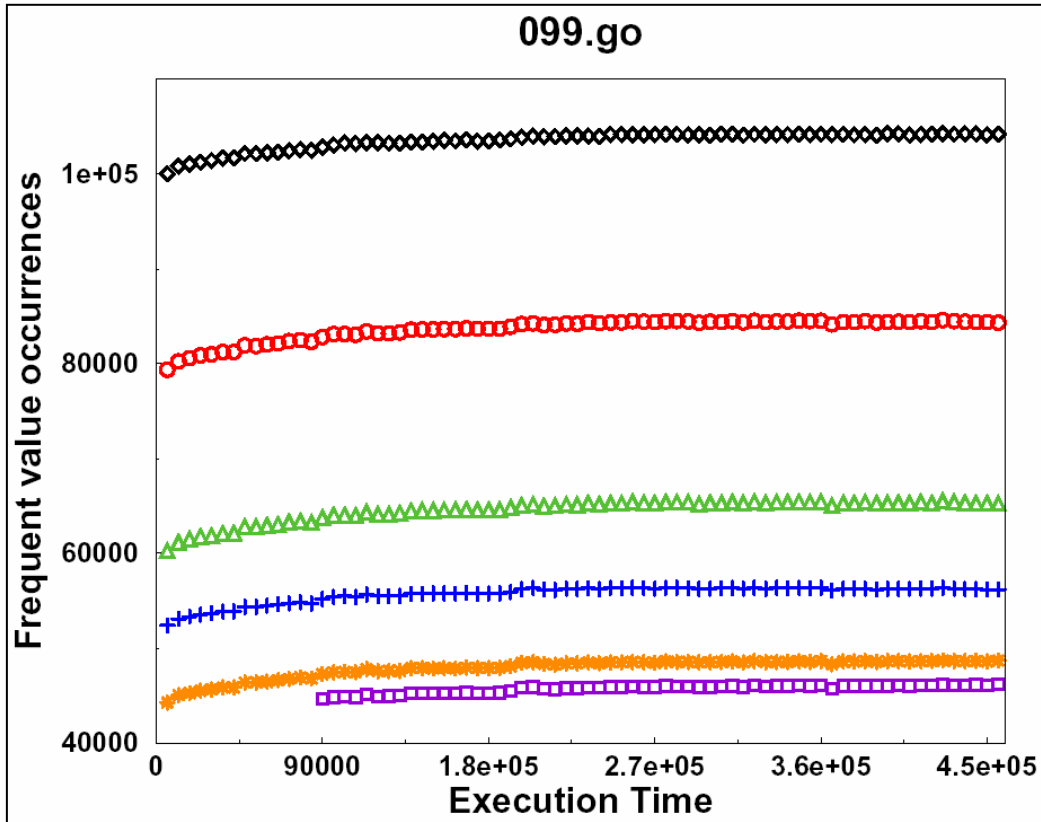
- ◇ Total
- Top 1 value
- △ Top 2 values
- + Top 4 values
- \* Top 8 values
- Top 10 values





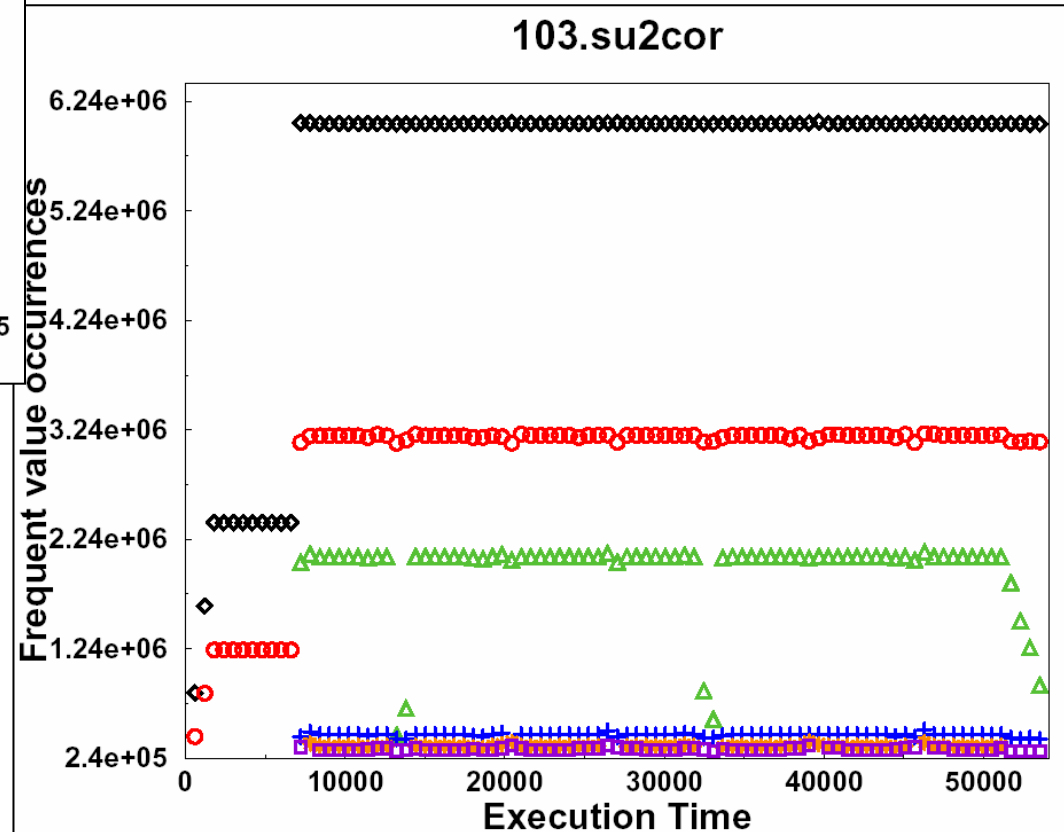
# Frequent Values Over Program Execution

099.go



- ◇ Total
- Top 1 value
- △ Top 2 values
- + Top 4 values
- \* Top 8 values
- Top 10 values

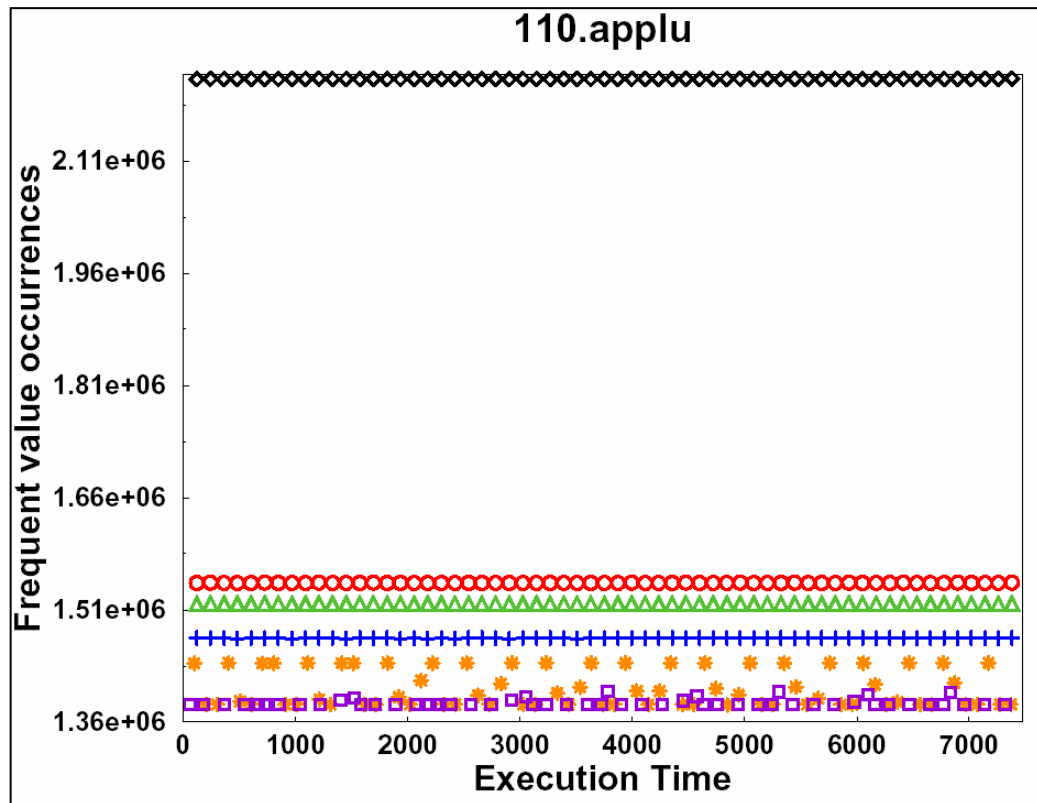
103.su2cor





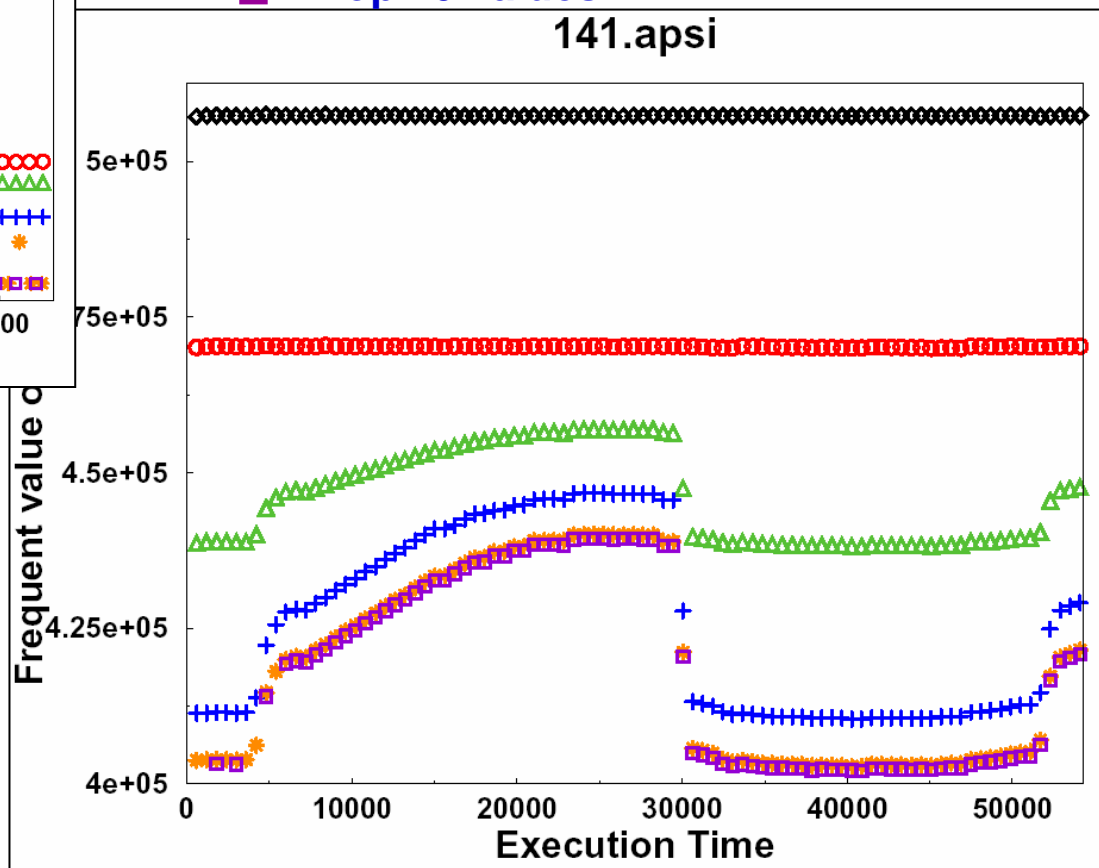
# Frequent Values Over Program Execution

110.aplu



- ◇ Total
- Top 1 value
- △ Top 2 values
- + Top 4 values
- \* Top 8 values
- Top 10 values

141.apsi



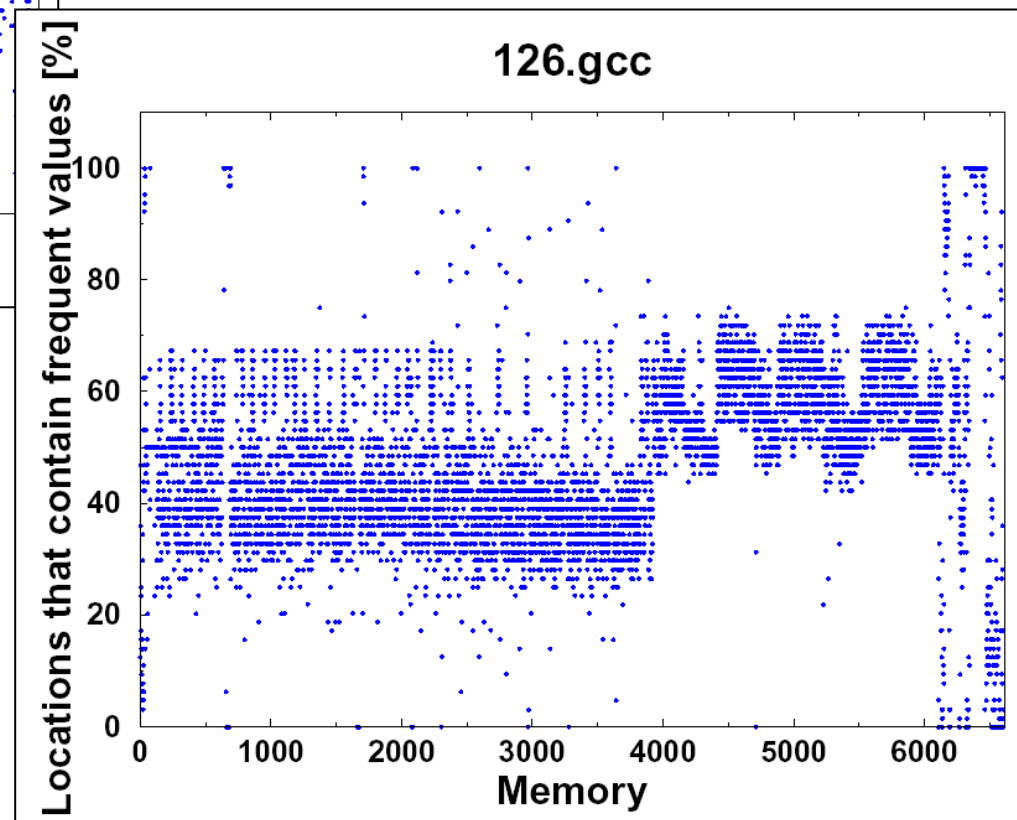
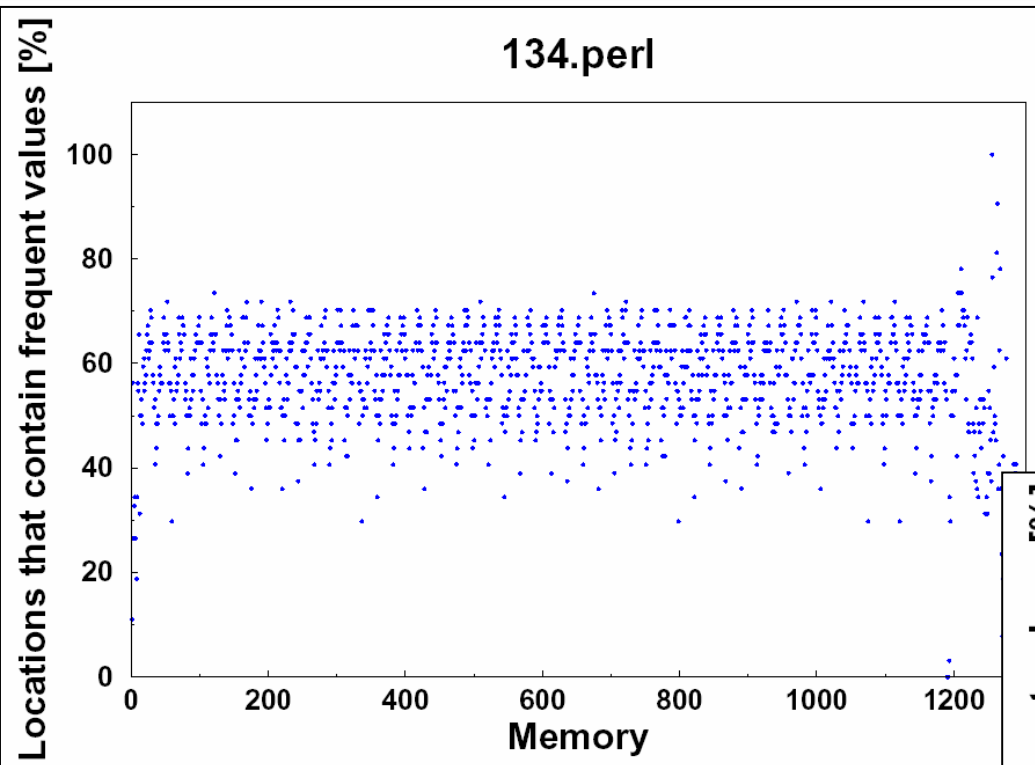


# Frequent Value Locality

- Small number of values occupy a large fraction of memory locations in use  
⇒ frequent values.
- Frequent values remain the same for long periods of time during execution  
⇒ they can be identified and exploited.
- **Frequent values are distributed quite uniformly in memory**  
⇒ **they are encountered in all parts of the memory.**

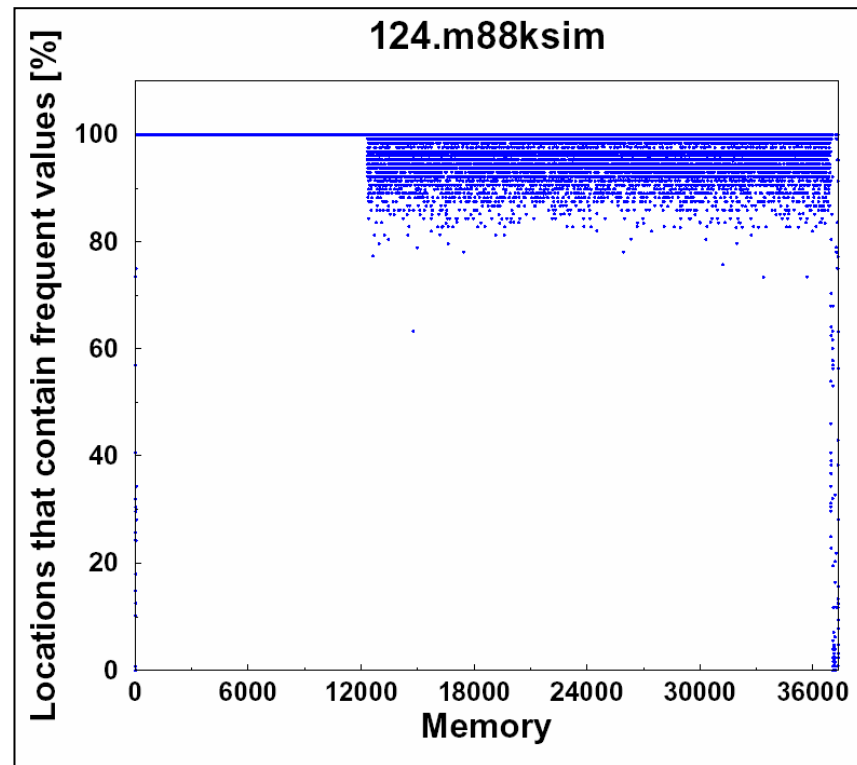
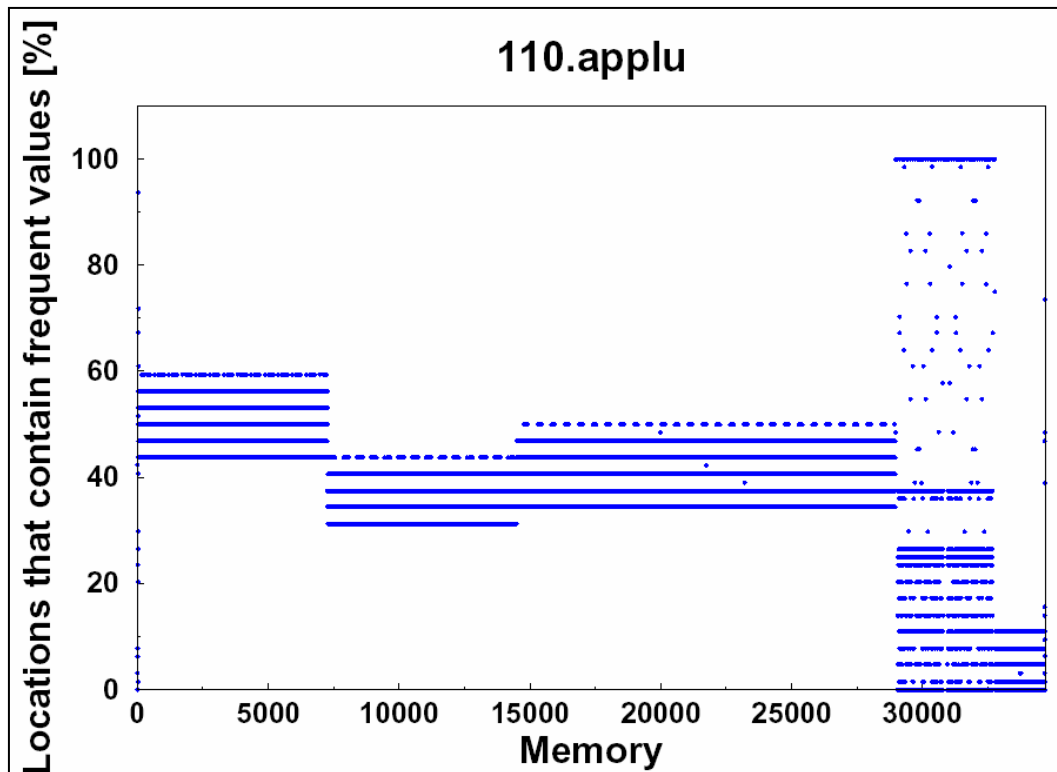


# Frequent Value Distribution in Memory



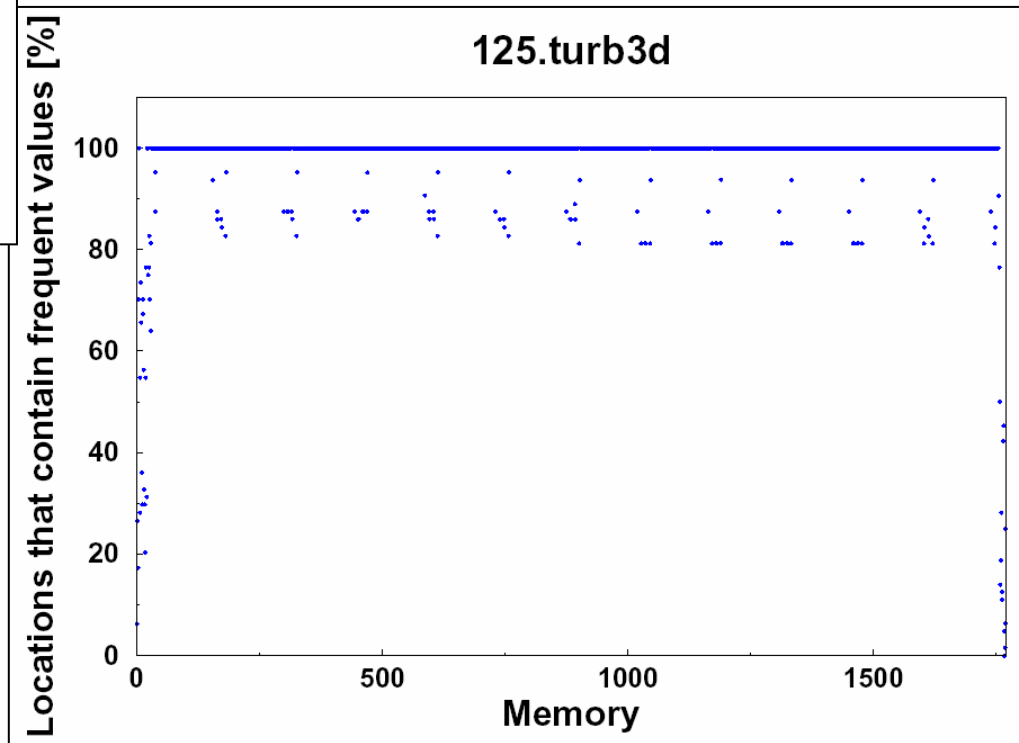
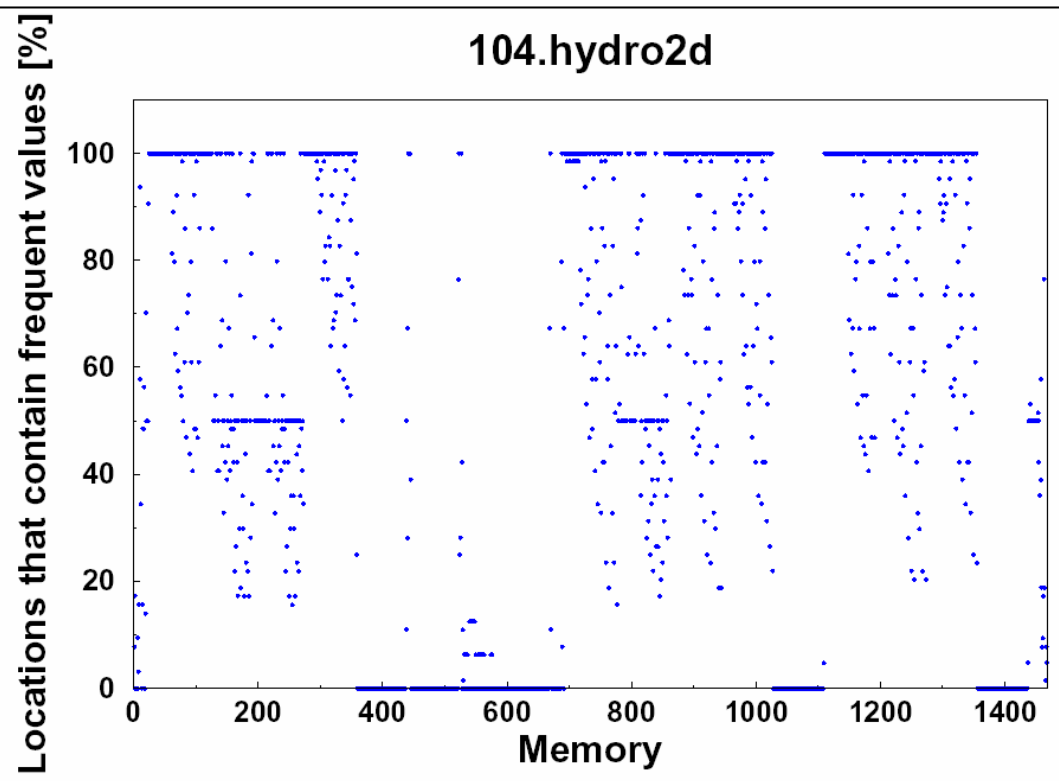


# Frequent Value Distribution in Memory



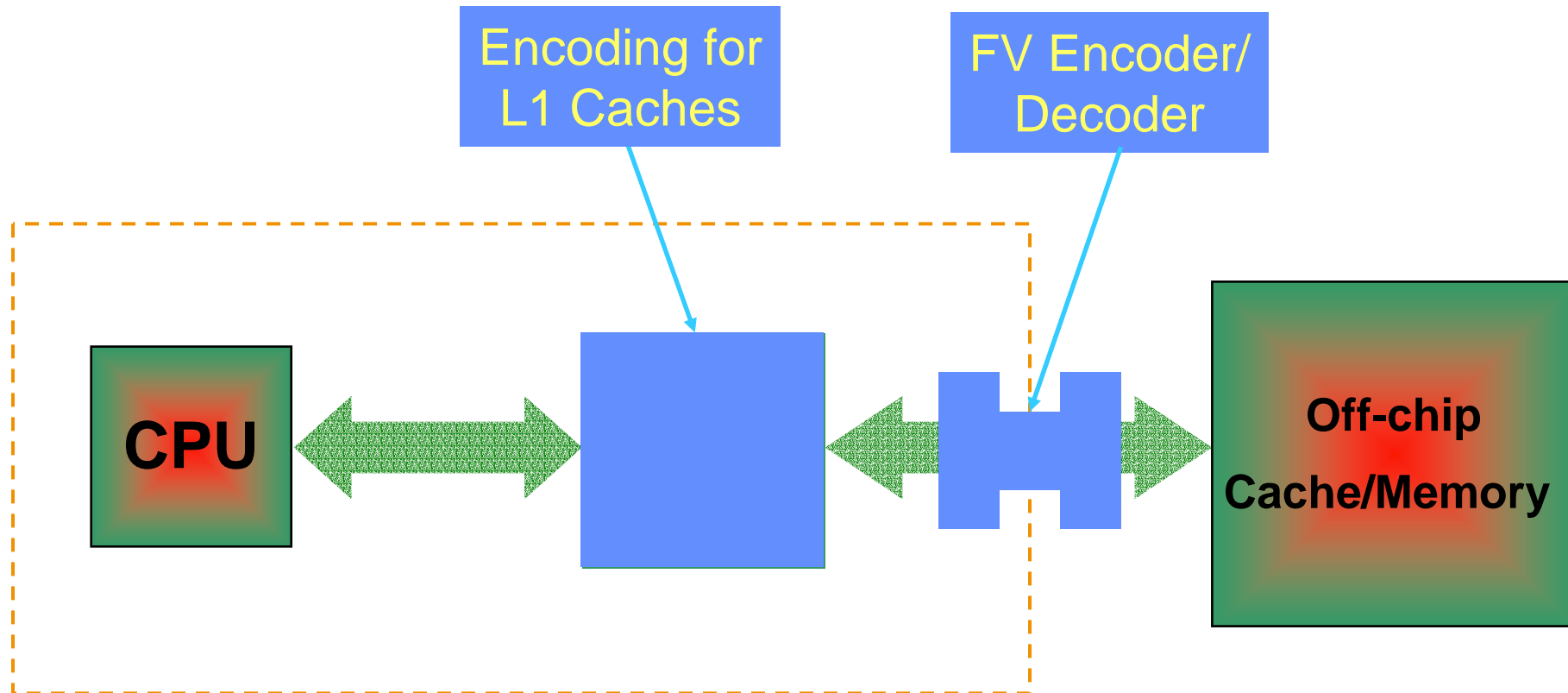


# Frequent Value Distribution in Memory





# Applications





# Applications

## □ Switching Activity on External Data Bus

- Frequently **transferred** values are **encoded** to **reduce switching**.
- **Dynamically varying** set of frequent values.

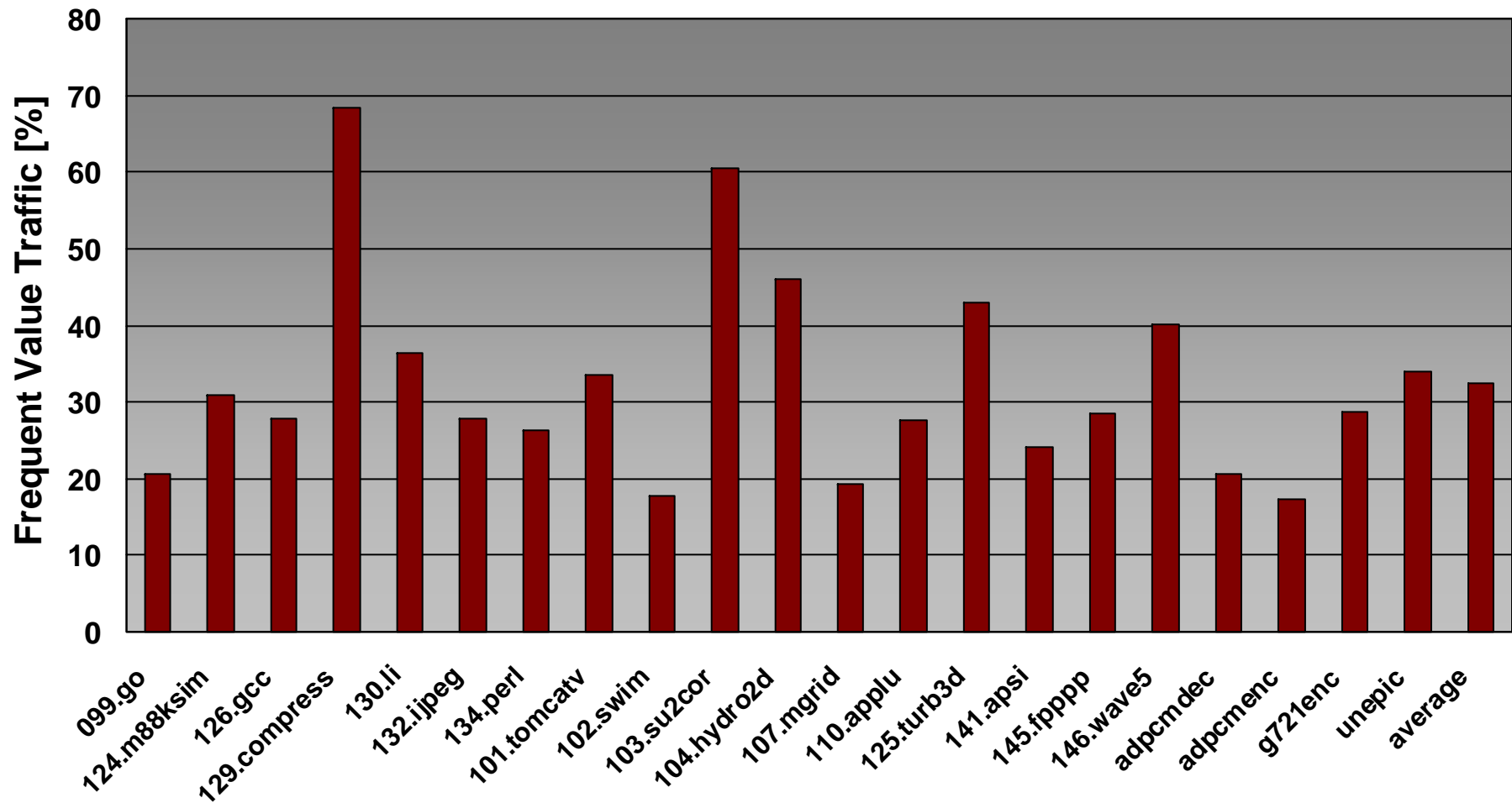
## □ Switching Activity in Data Caches

- Frequently **accessed** values are **encoded** or **compressed**.
- **Fixed** set of frequent values must be identified.



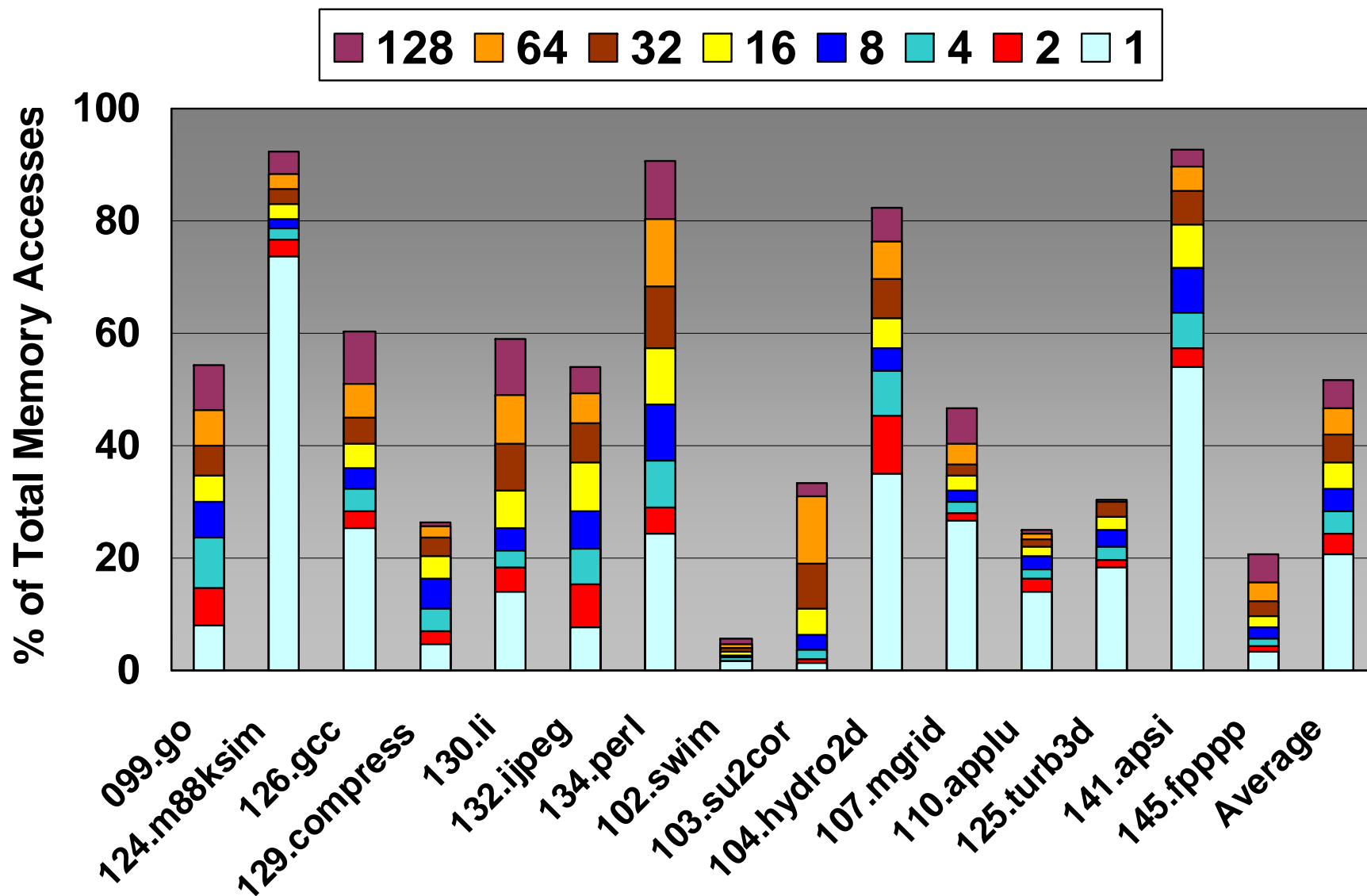
# Frequently Transferred Values

Changing set of 32 values





# Frequently Accessed Values





# Finding Frequent Values

## **Transferred Values - Data Bus Switching**

- ❑ **Update the list of values continuously**

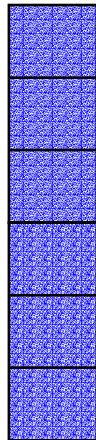
## **Accessed Values - Data Cache**

- ❑ **Profile early in execution and use later**
- ❑ **Profile one run and use in later runs**

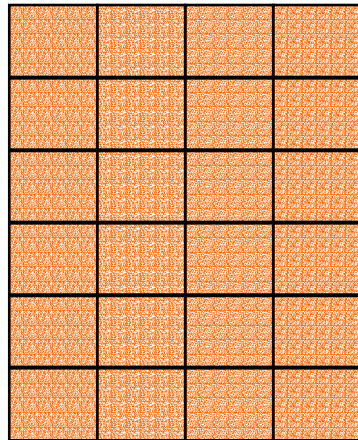


# Changing Set of Frequent Values

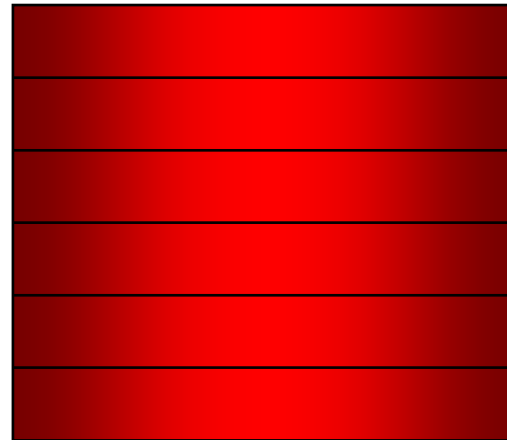
Ref. Bit



Timestamp



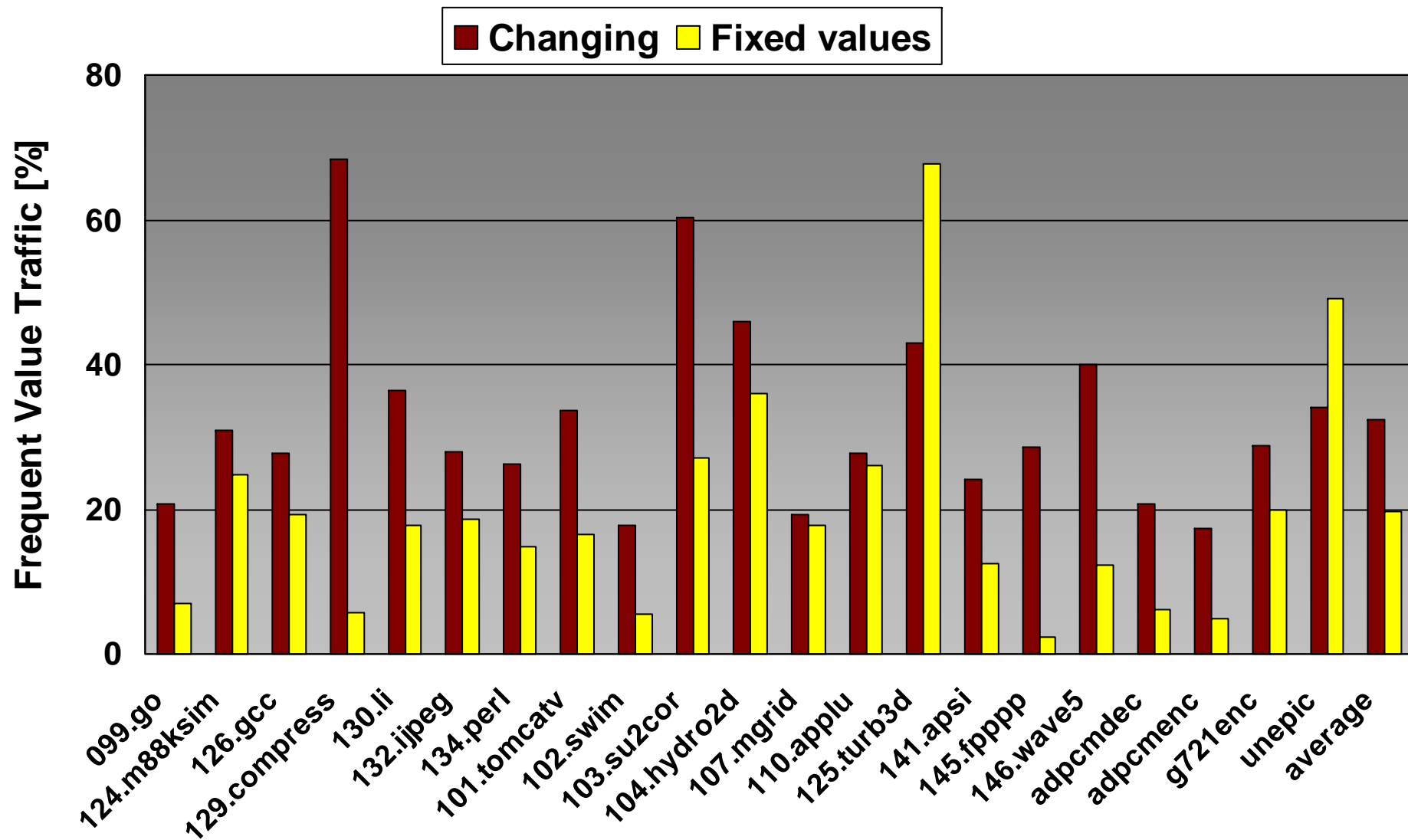
Frequent Values



- Reference bit is shifted right at regular intervals to form the timestamp.
- Timestamps (and reference bits) are used to implement LRU replacement policy.

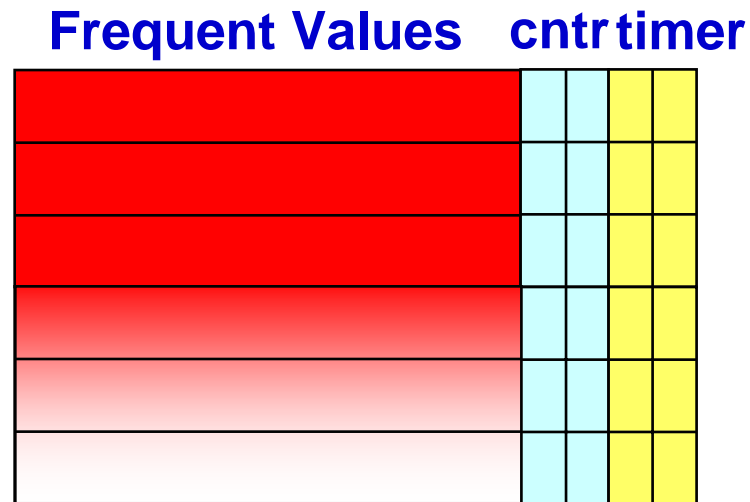


# Frequently Transferred Values





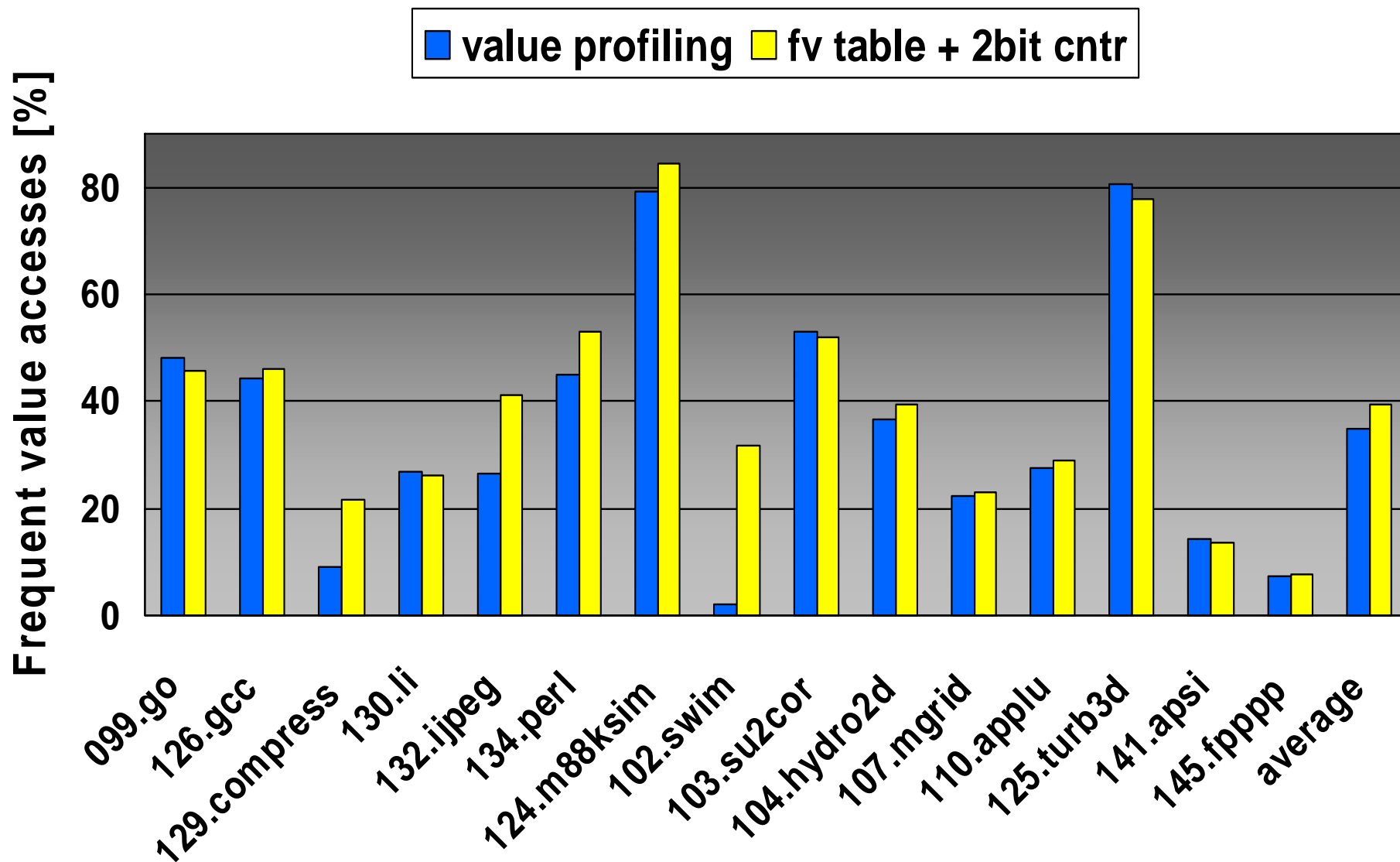
# Hardware Profiling



- To capture  $n$  values, use table of  $2n$  entries
- Increment **cntr** field every time value is seen
- Swap entry with the one above when **cntr** is saturated
- **timer** field is used for replacement
- Frequent values can be read out from the top  $n$  entries, after certain time

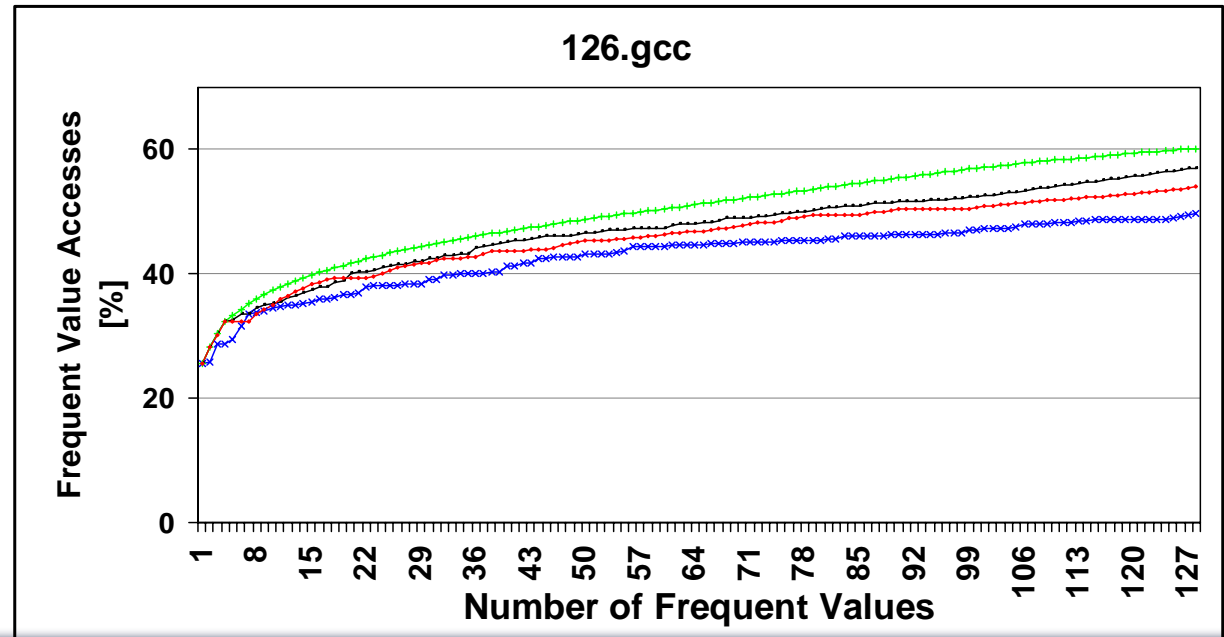
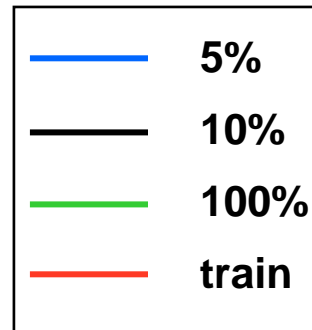
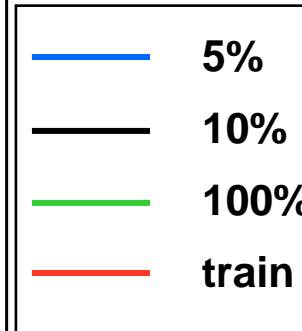
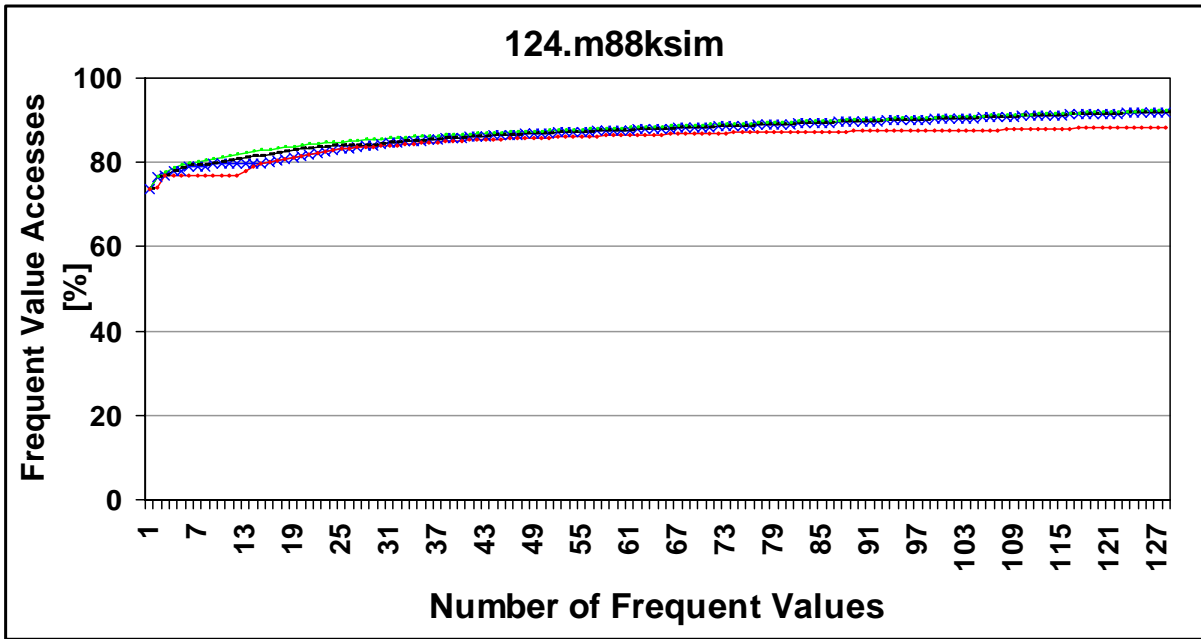


# Comparison with Value Profiling



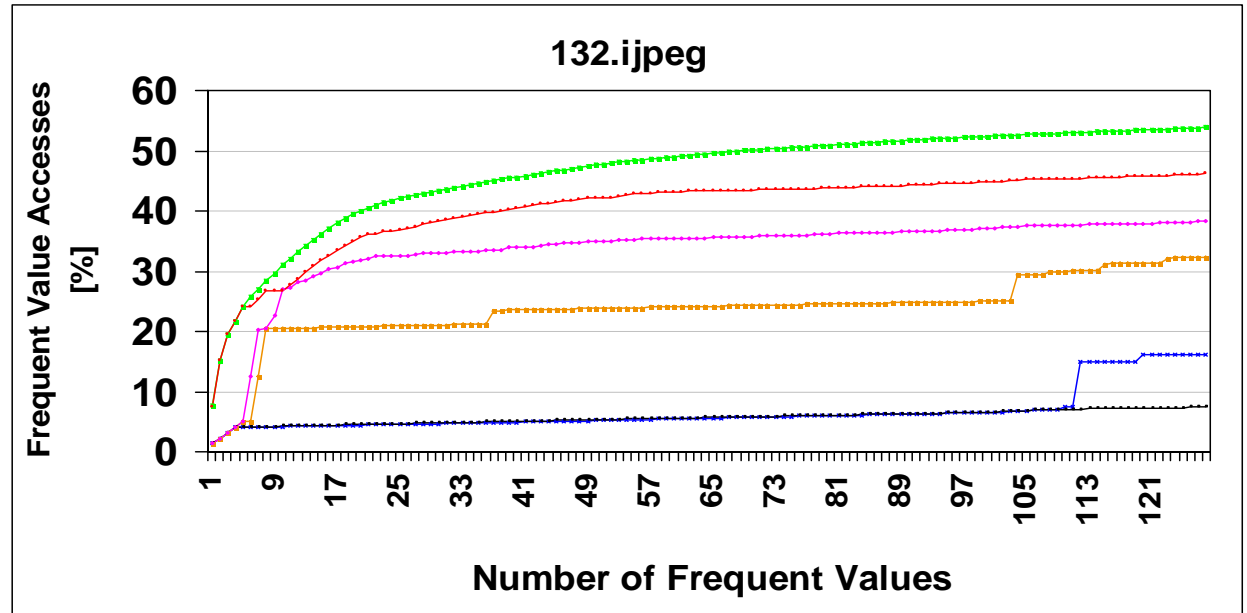
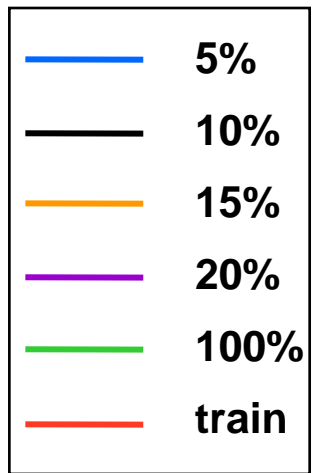
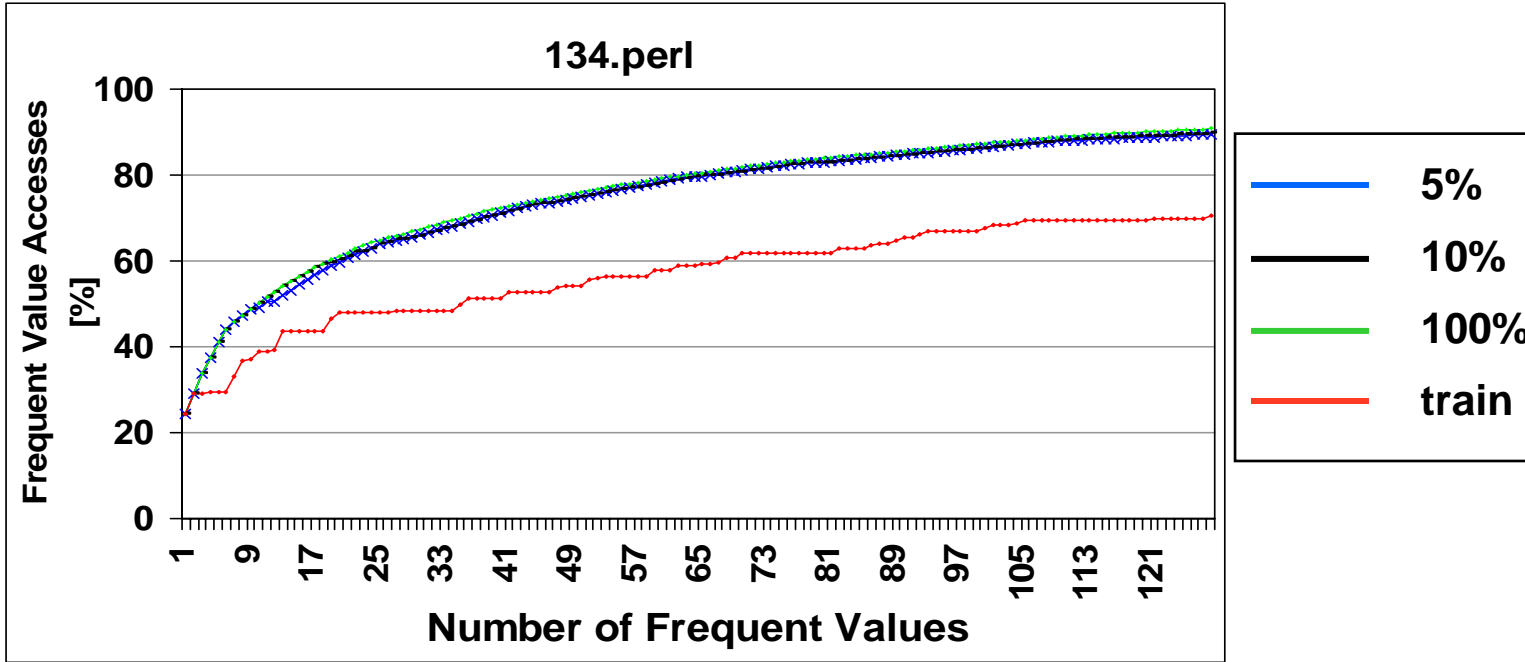


# Profile & Use: Same vs Different Run





# Profile & Use: Same vs Different Run





# Reducing Data Bus Switching

- ❑ **Capacitance** of an external bus is high.
- ❑ **Power consumption** can be reduced by minimizing **switching activity** of external buses.
- ❑ **Memory reference locality** can be exploited to substantially reduce **address bus** switching.
- ❑ Similar techniques for **data bus** do not exist.  
⇒ **Frequent value locality**



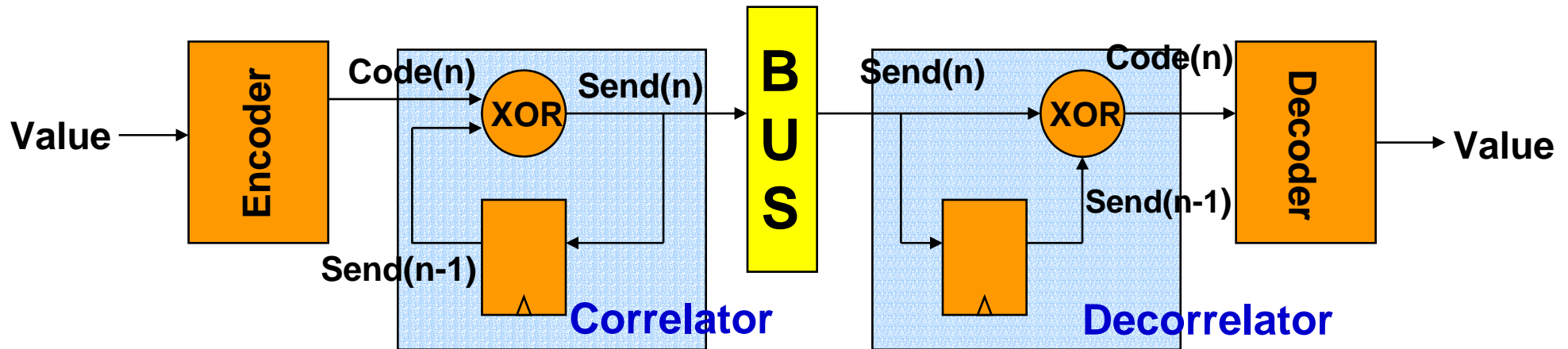
# Encoding Scheme

- ❑ **Create a table of frequent values** with no more than 32 entries.
- ❑ Transmit frequent values in **encoded form**.
- ❑ Use **one-hot encoding** - exactly one line is hot and the position of hot line identifies the frequent value.
  - ⇒ **If two successive values are frequent values, at most two lines will switch.**



# Encoding Scheme contd..

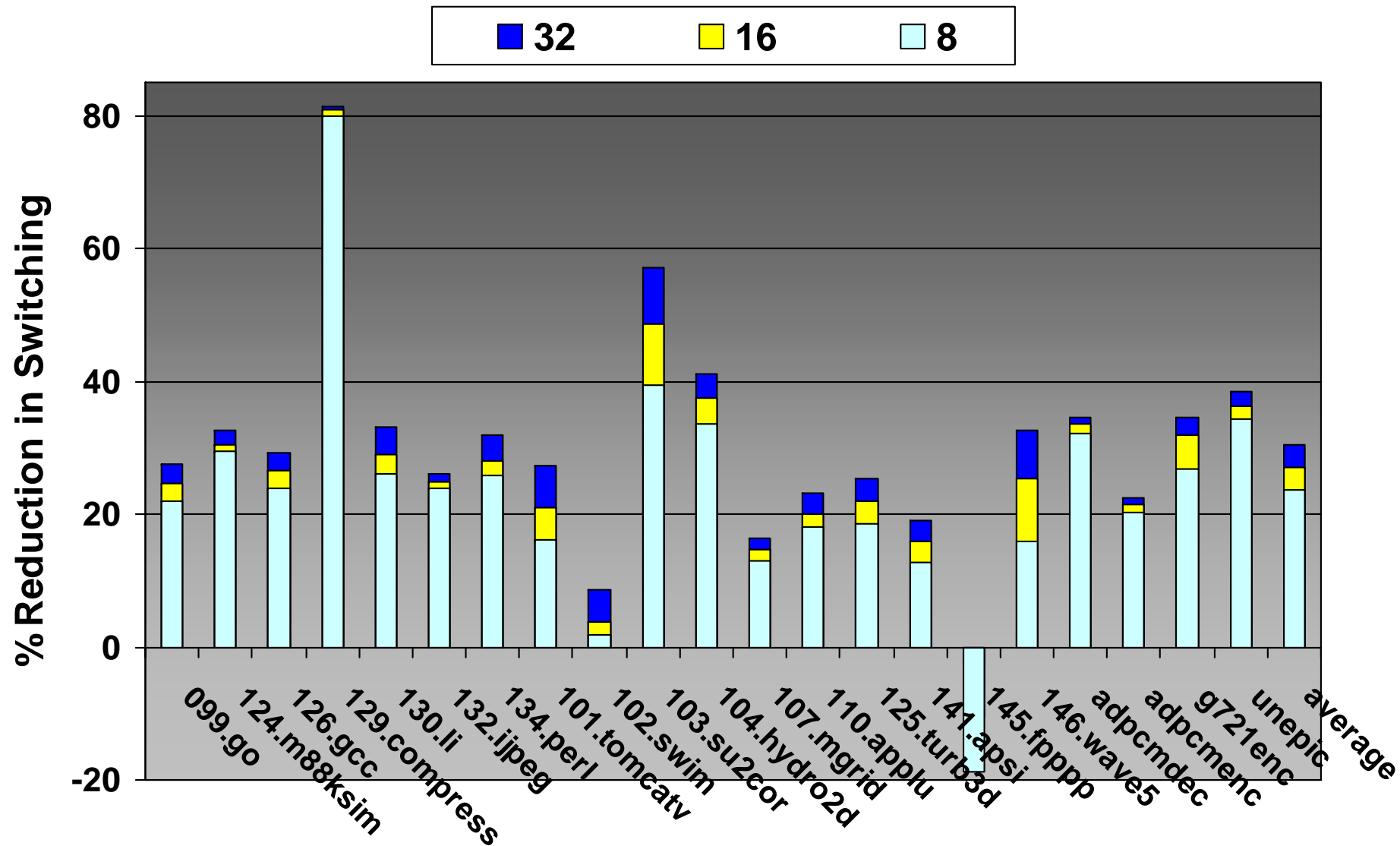
- By **XORing consecutive values** switching can be reduced for **(nonfrequent,frequent)** value pair.  
*Correlator switches only wires whose input is 1.*



$$\begin{aligned} \text{Send}(n) &= \text{Send}(n-1) \text{ XOR } \text{Code}(n) \\ \text{Code}(n) &= \text{Send}(n) \text{ XOR } \text{Send}(n-1) \end{aligned}$$

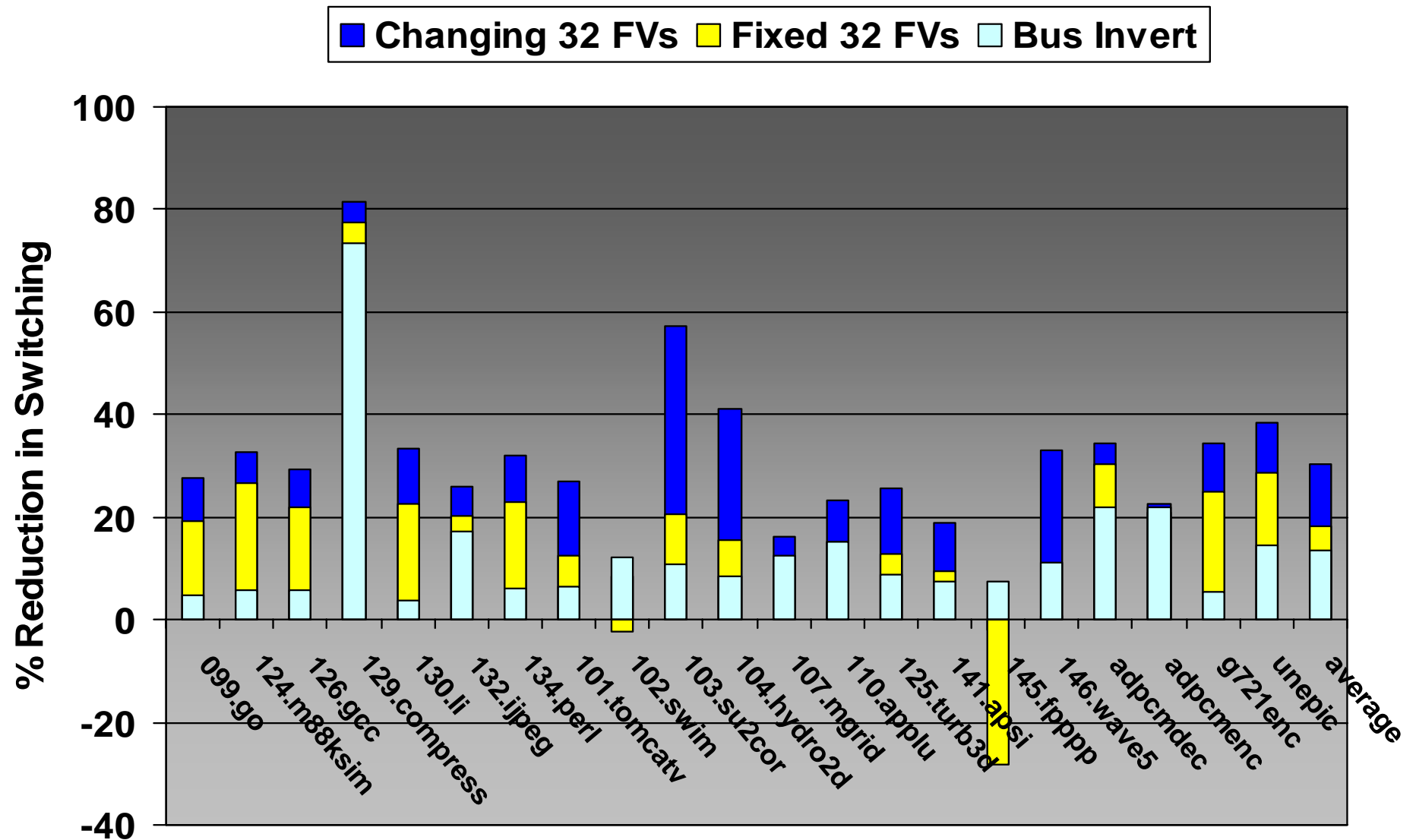


# Switching Reduction: Changing FVs



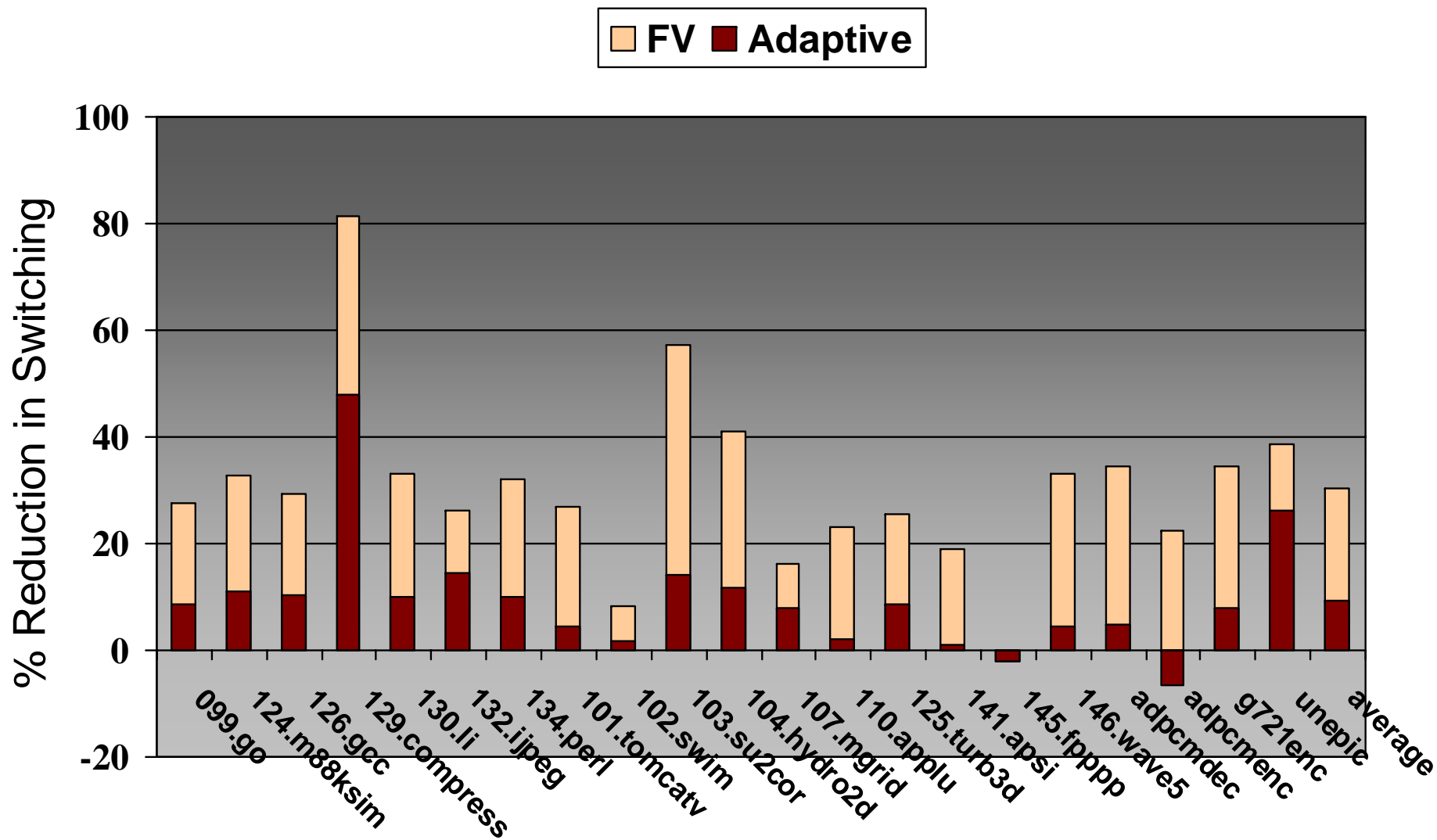


# Bus-invert Encoding vs FV Encoding



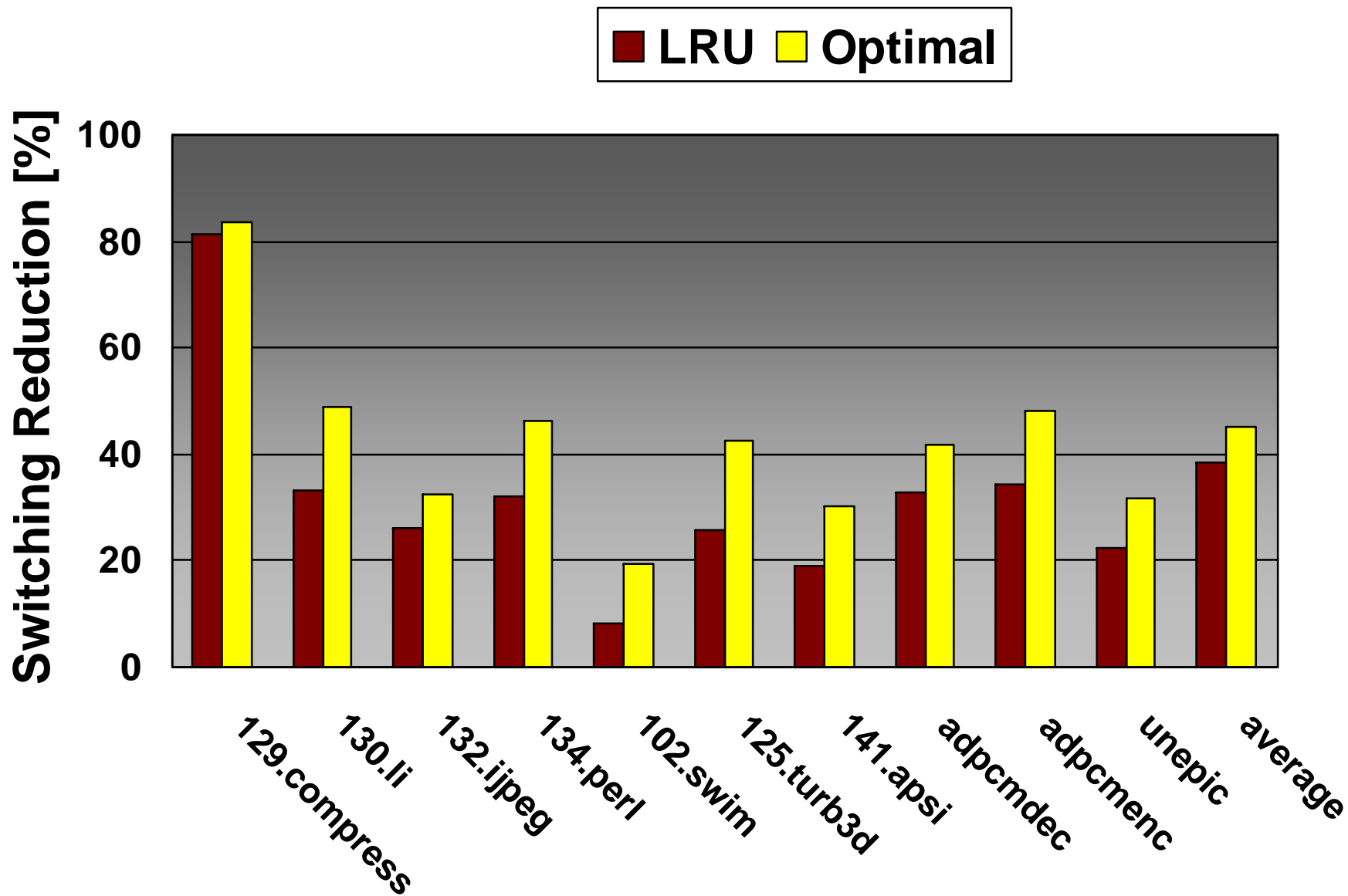


# FV Encoding vs Adaptive Encoding





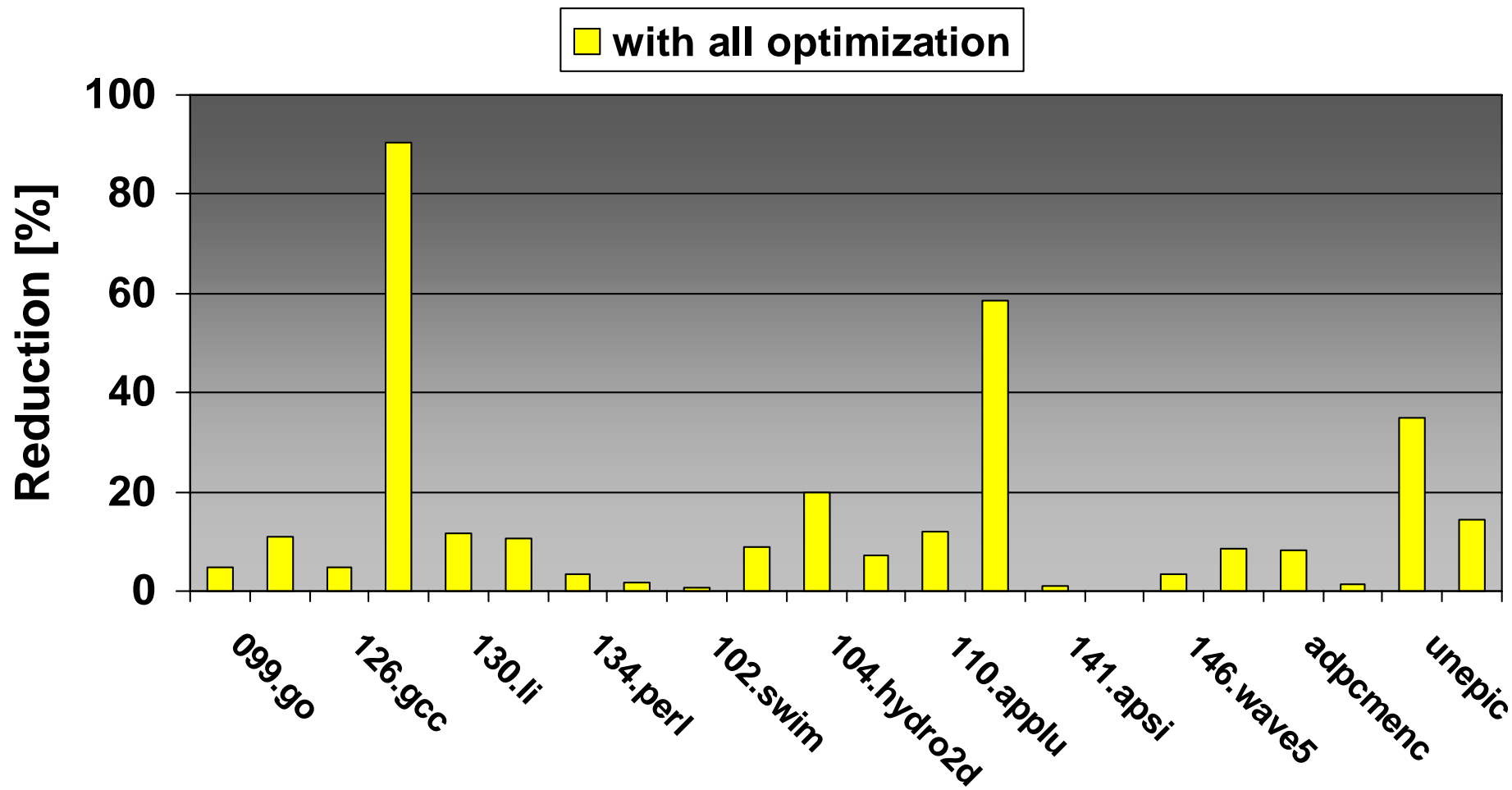
# Replacing Using LRU vs. Optimal





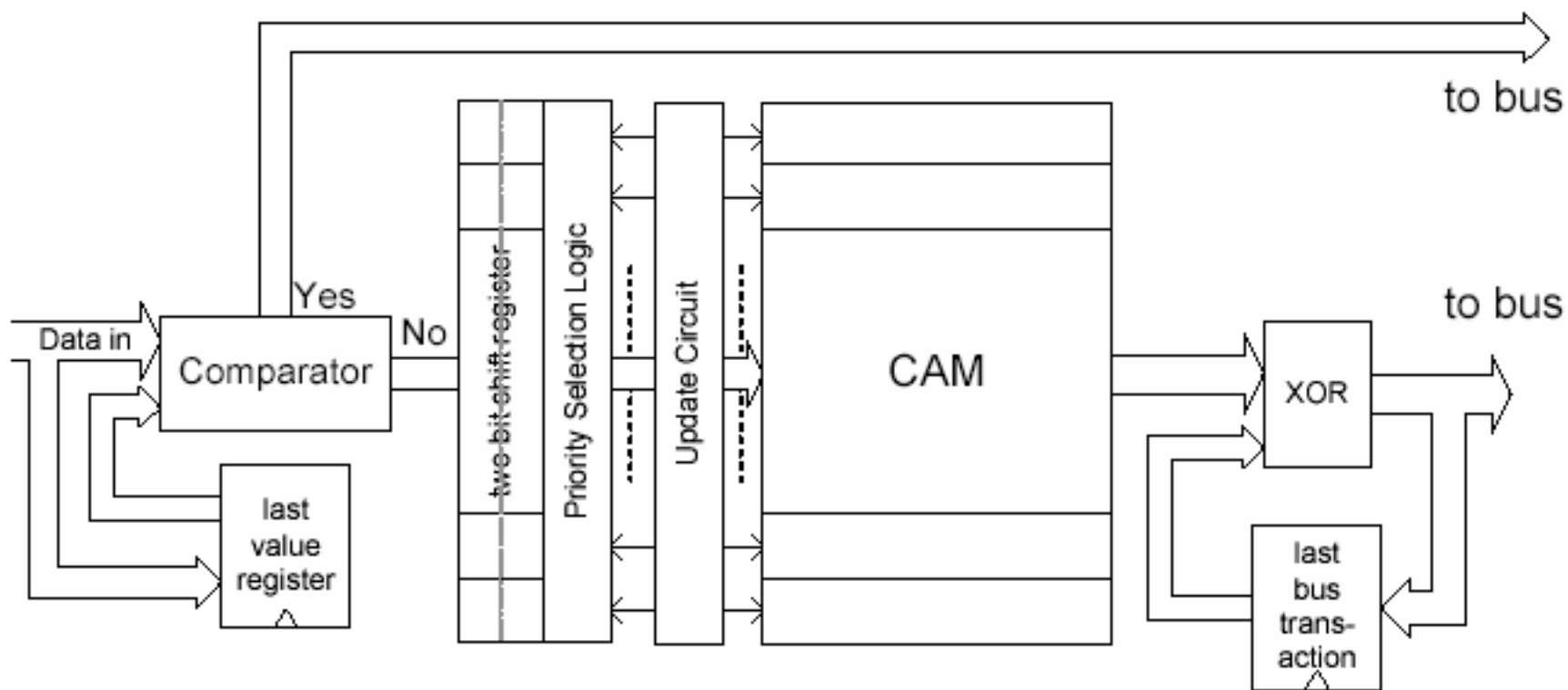
# Reducing Coding/Decoding Activity

- Exclude coding of values with few ones (e.g.,  $< 16$ ).





# Detailed Design





# Energy Savings

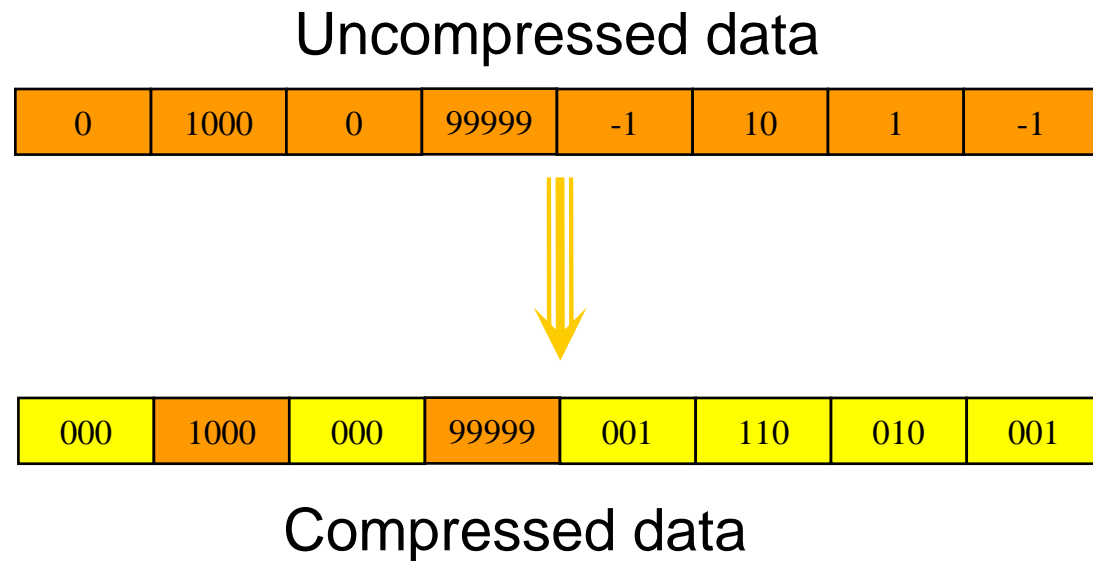
Benchmarks	En-/De-coder Energy (mJ)	Bus Energy (mJ)		Energy savings (%)
		Before	After	
099.go	0.72	32.27	23.32	25.51%
124.m88ksim	0.34	15.42	10.18	31.79%
126.gcc	0.02	0.70	0.49	26.89%
129.compress	0.02	1.26	0.24	79.69%
130.li	0.19	8.08	5.40	30.91%
132.jpeg	0.01	0.71	0.52	24.34%
134.perl	0.10	4.90	3.33	29.84%
101.tomcatv	0.62	28.32	20.61	25.02%
102.swim	0.07	3.81	3.48	6.81%
103.su2cor	0.45	22.50	9.60	55.32%
104.hydro2d	0.37	17.89	10.55	38.93%
107.mgrid	0.40	25.40	21.27	14.71%
110.applu	0.31	20.18	15.52	21.55%
125.turb3d	0.03	1.19	0.88	22.81%
141.apsi	0.11	5.49	4.45	16.91%
145.fpppp	0.66	30.93	31.58	-4.24%
146.wave5	0.06	3.07	2.06	30.98%
adpcmdec	0.00	0.06	0.04	32.91%
adpcmenc	0.00	0.04	0.03	20.63%
g721enc	0.56	25.61	16.76	32.37%
unepic	0.03	1.03	0.63	35.84%
Average				28.55%



# Frequent Value Cache

Frequent values are stored in encoded form.

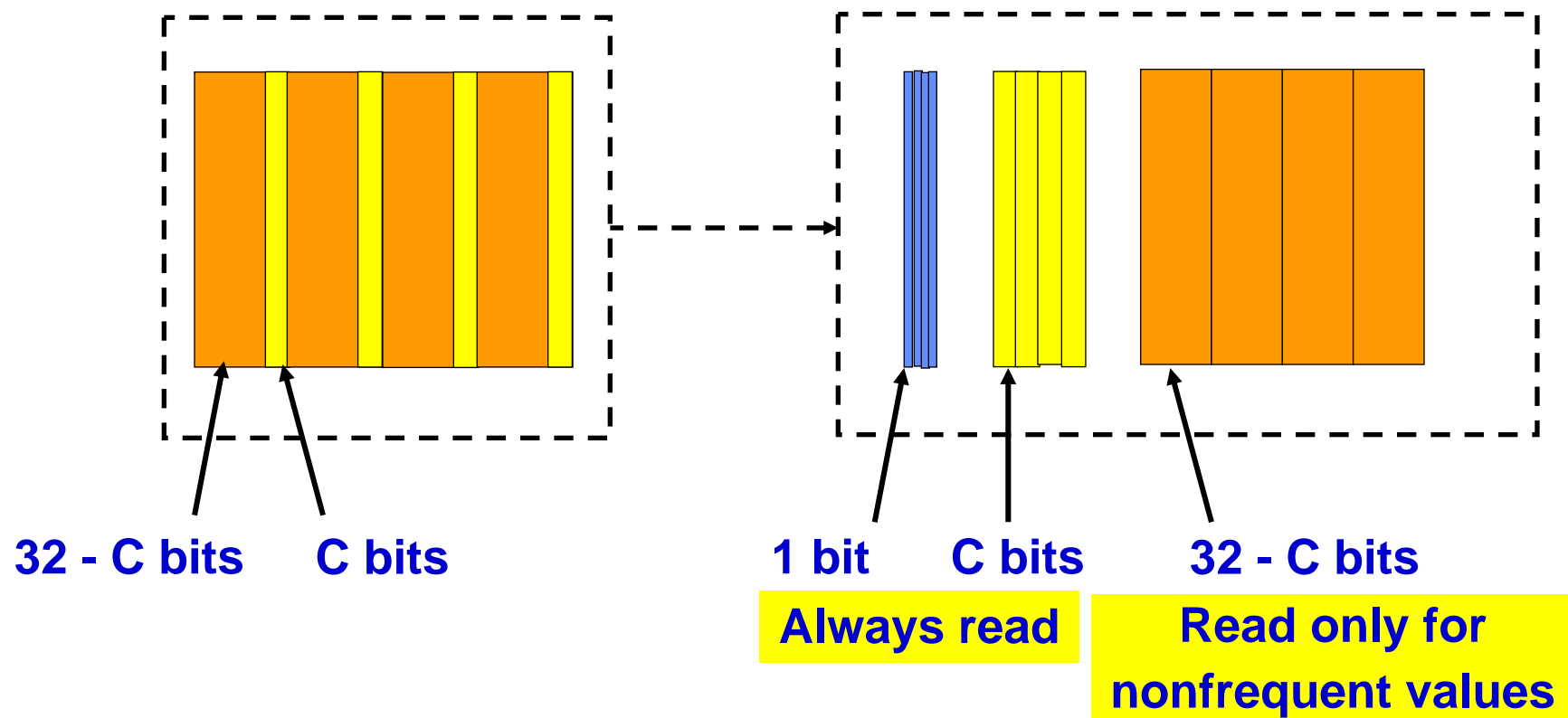
<u>32-bit FV</u>	<u>3-bit code</u>
0	000
-1	001
1	010
2	011
4	100
8	101
10	110
16	111





# Frequent Value Cache

## $2^C$ Frequent Values





# Access Timing

## Frequent Values – 1 Cycle Access

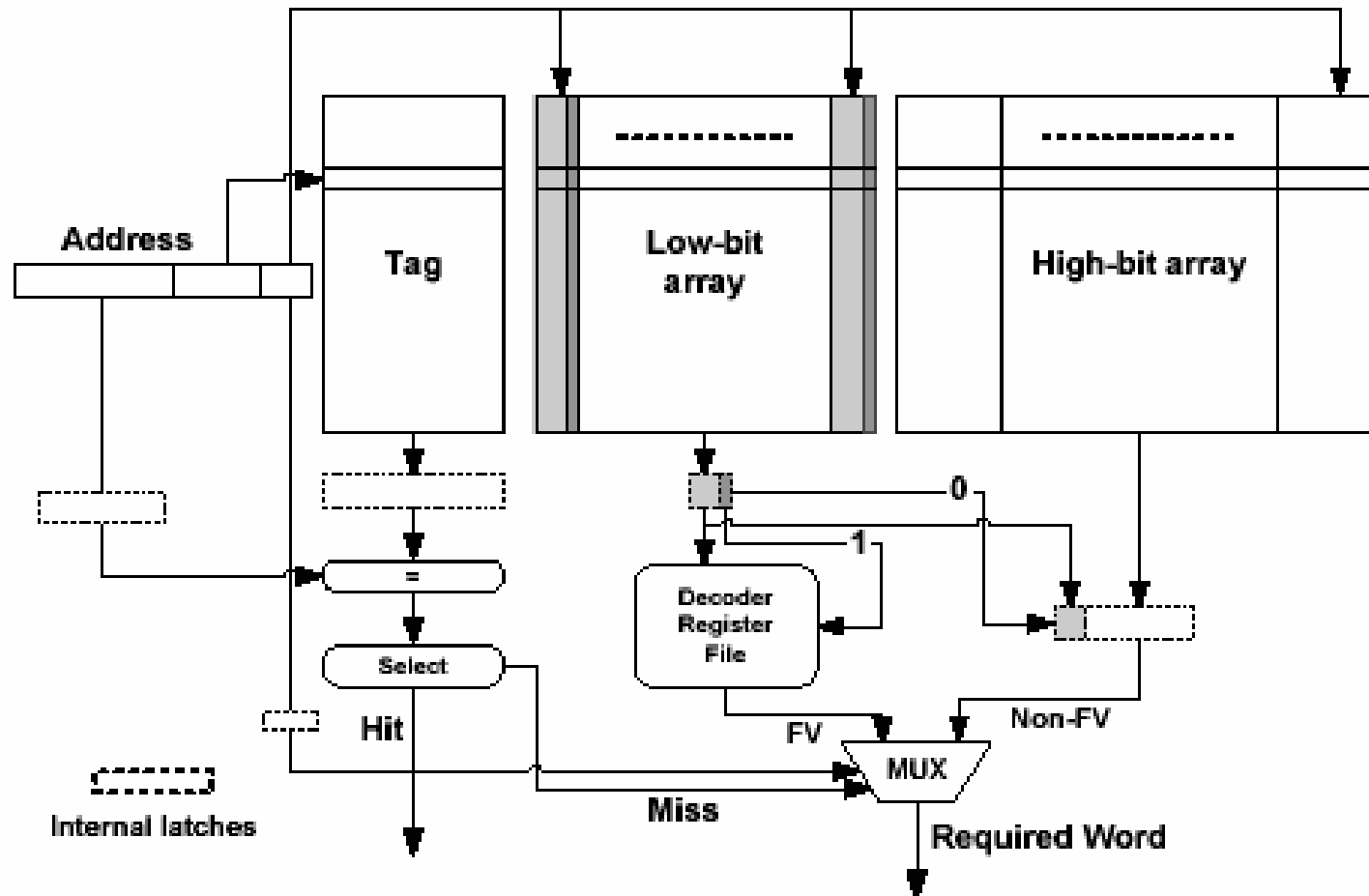
Tag path delay	
Read $C+1$ bits	Decode

## Nonfrequent Values – 2 Cycle Access

Tag path delay		Read $32 - c$ bits
Read $C+1$ bits	--	



# Detailed Design



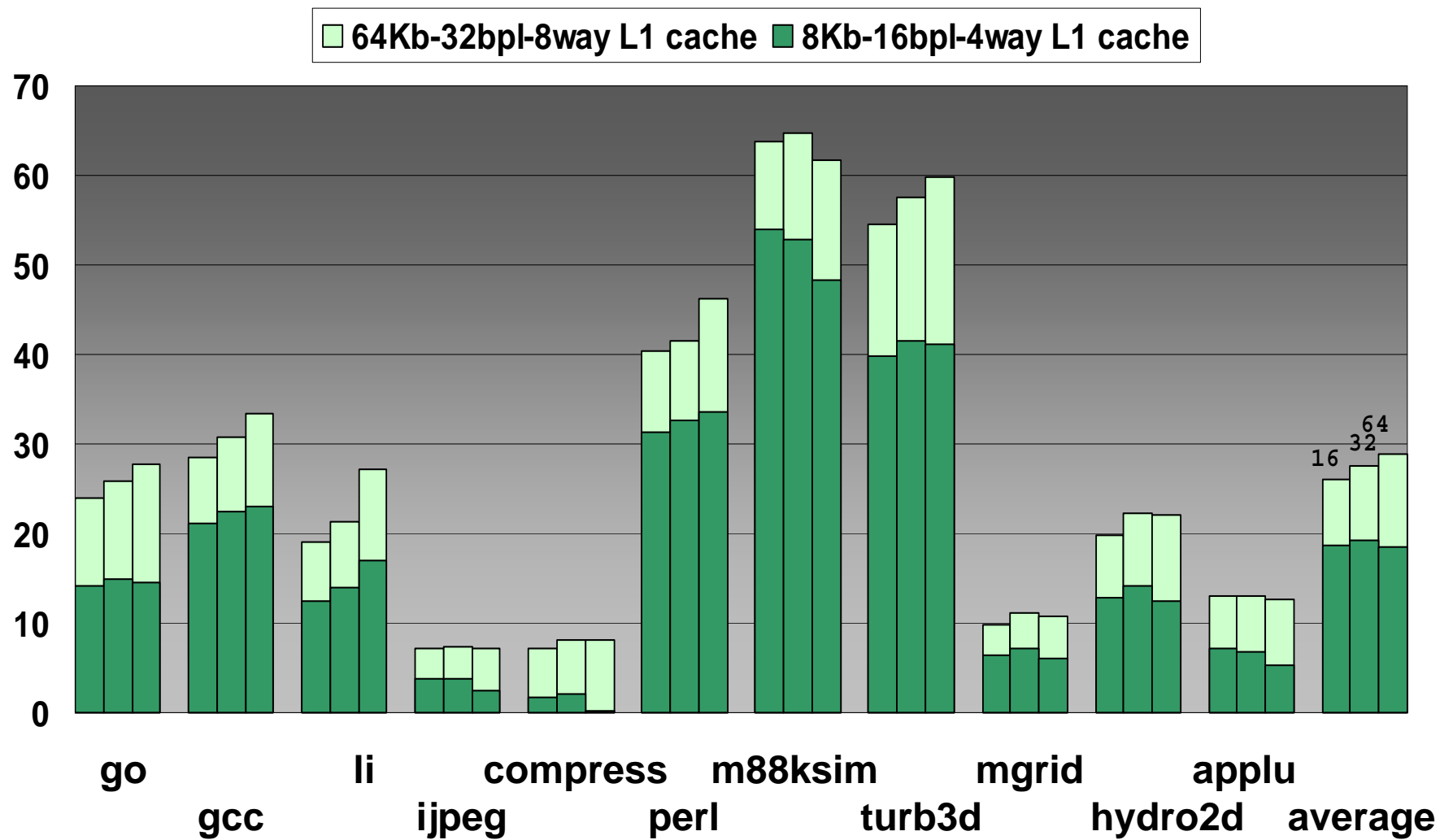


# FVC Critical Path Delay (ns)

	8KB 4 way		64KB 8 way	
	8bit per line	16bit per line	16bit per line	32bit per line
4 values	2.36+0	2.11+0	3.13+0	2.67+0
8 values	2.36+0	2.11+0	3.13+0	2.67+0
16 values	2.36+0	2.11+0	3.13+0	2.67+0
32 values	2.36+0	2.11+0	3.13+0	2.67+0
64 values	2.36+0	2.11+0	3.13+0	2.67+0
128 values	2.36+0.04	2.11+0.04	3.13+0.06	2.67+0

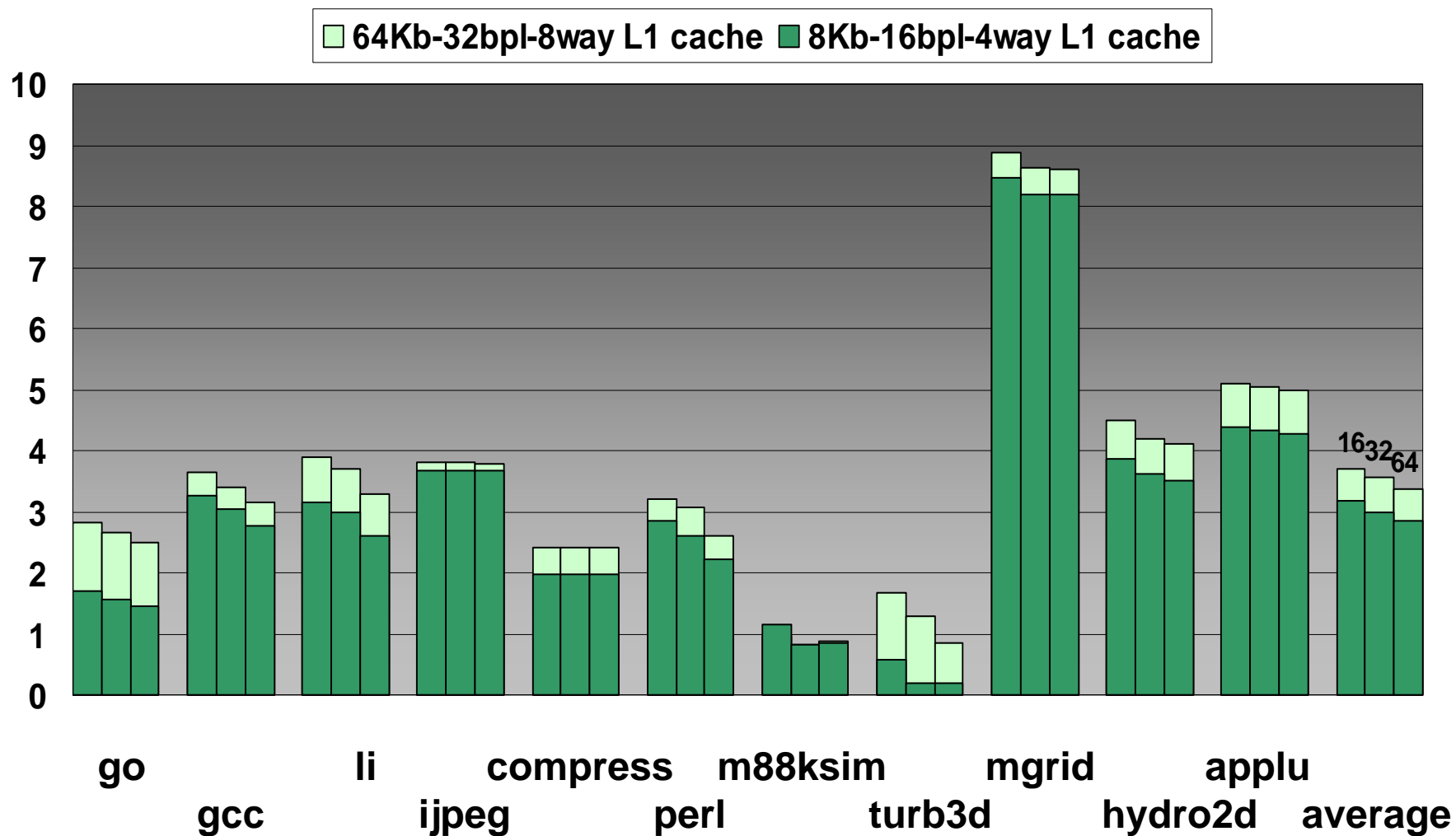


# Cache Energy Reduction [%]



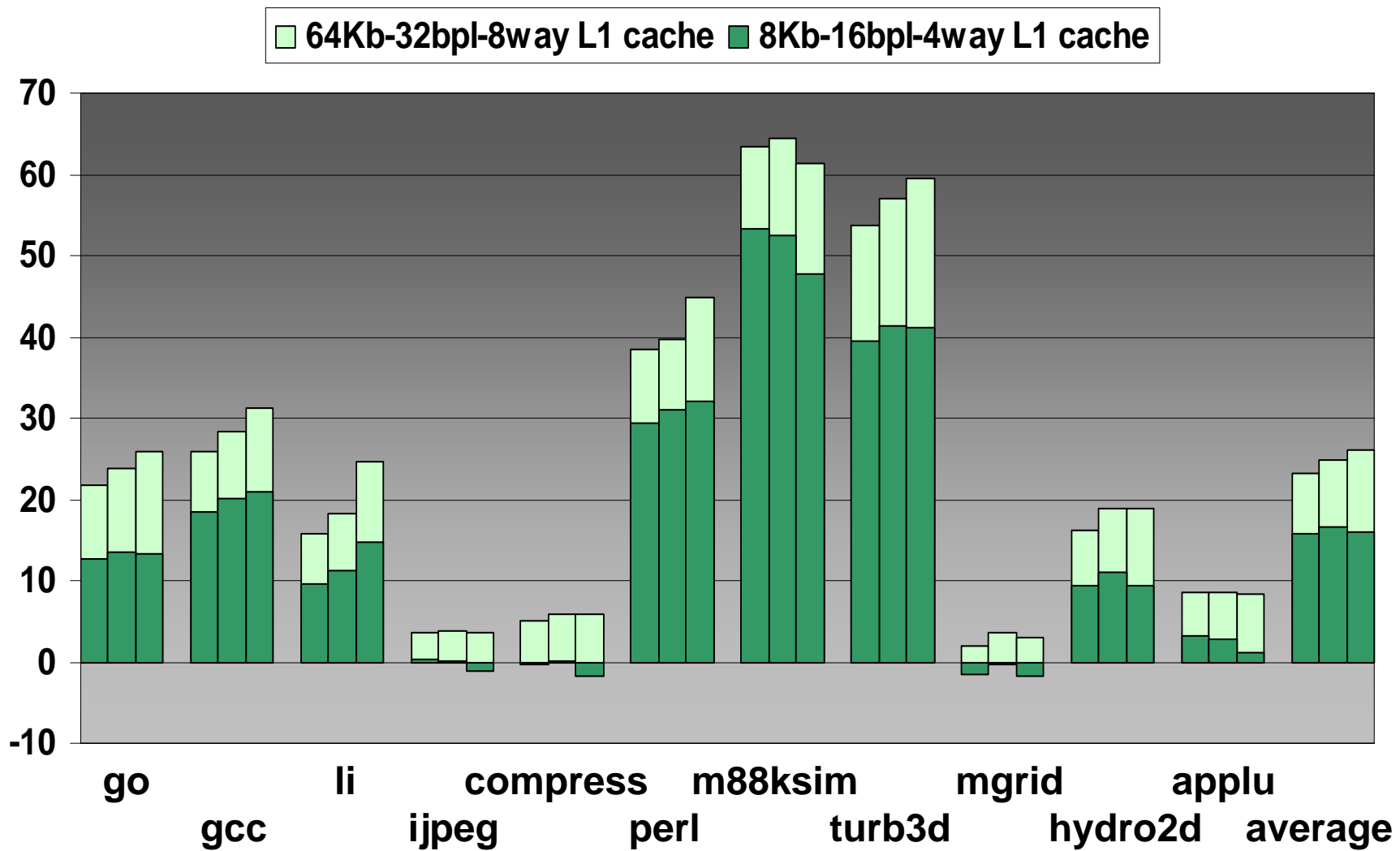


# Execution Time Increase [%]





# Energy x Delay Reduction [%]





# Summary

- ❑ **Frequent value phenomenon**
- ❑ **Frequent values can be found and exploited**
- ❑ **Applications**
  - ❑ **Switching in External Data Bus**
  - ❑ **Low Power Data Cache**



# Publications

**Jun Yang and Rajiv Gupta,**

**“Frequent Value Locality and its Applications,”**

*ACM Transactions on Embedded Computing Systems,*  
Vol. 1, No. 1, November 2002.

**Jun Yang, Rajiv Gupta, and Chuanjun Zhang**

**“Frequent Value Encoding for Low Power Data Buses,”**

*ACM Transactions on Design Automation of Electronic Systems,* to appear.

**Jun Yang and Rajiv Gupta**

**“Energy Efficient Frequent Value Data Cache Design,”**

*IEEE/ACM 35th International Symposium on Microarchitecture,*  
November 2002.



# Large and Small Values

