Coding for memory systems

July 10, 2013

1 Problem Statement

The objective of this project is to come up with coding theoretic solutions to avoid bank conflicts in a multi-port memory. In a multi-port memory data is stored on memory banks which is access by multiple ports. Simultaneous data access by two or more ports to the same bank results in a bank conflict as only one I/O request to a bank can be served at a time. Presently bank conflict problem is avoided by performing repetition where multiple copies of the data is stored in different banks. For example, Fig 1 shows a repetition solution for storing $\mathbf{a}$ on a 2-read and 1-write memory [1]. However, the repetition solution entails large storage requirement. In the example shown in Fig. 1, data $\mathbf{a} = (a_1, \ldots, a_M)$ is stored on 4 memory banks of size M each. This scheme has rate $\frac{1}{4}$.

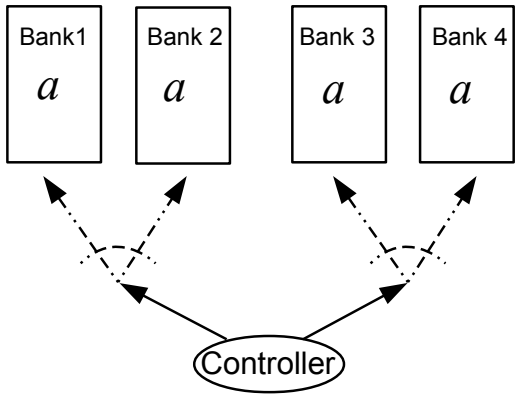

Figure 1: Repetition based solution for a 2-read and 1-write memory.

In a coding theoretic solution to bank conflict problem, an erasure code is used as opposed to simple repetition. An erasure code introduces redundancy in the memory systems so that when a memory bank is serving to a processor, another read request for the data stored in this memory bank (by another processor) can be served by treating the busy bank in erasure. In this situation the second request is served by reconstructing the requested data from the codeword symbols available from free banks.
Recently, the erasure coding based solutions for multi-port memories have been proposed by Memoir Systems. Algorithmic Memory (by Memoir Systems) is using erasure codes (like Reed-Solomon codes). However, different codes exist allowing efficiency for different trade-offs. For example, regenerating codes [2] from distributed storage systems (DSS) literature allow for trading-off: 1) the complexity of the circuitry (or bandwidth) used for the recovery of busy bank (when a bank conflict occurs) and 2) amount of storage space in each memory bank.
There is another class of codes in DSS literature, namely locally repairable codes [3], which allow bank recovery (in the event of bank conflicts) by contacting a small number of other banks. In particular, let

multi-port memory implement an r-locally repairable code and assume that there are 2 simultaneous access to the same memory bank. One of these 2 simultaneous requests can be served from the bank itself. Now, we can treat this bank in erasure and contact a set of r other banks to serve the second request.
Noting the possibility of using various classes of codes for better read access in multi-port memories, we explain the advantage of coding theoretic solutions with the help of an example. We consider a scenario with the following set of parameters:

- Let the size of each memory bank be M symbols.
- We have to store 2-block long data ($2M$ symbols) on the memory system.
- Let $(\mathbf{a}, \mathbf{b})$ denote the data to be stored with $\mathbf{a} = (a(1), \ldots, a(M))$ and $\mathbf{b} = (b(1), \ldots, b(M))$.
- We want to design a multi-port memory that supports any 2 reads $(a(i), a(j))$, $(b(i), b(j))$, or $(a(i), b(j))$; and 1 write $a(k)$ or $b(k)$.

Now, using a 4-repetition based scheme [1], we obtain the memory design described in Fig. **??**. As mentioned before, this scheme has rate $\frac{1}{4}$. (Here, we use 8 memory banks to store 2 data blocks.) This scheme consumes takes 8 area unit for 8 memory banks (one area unit for one memory bank) and additional space for controller circuitry.

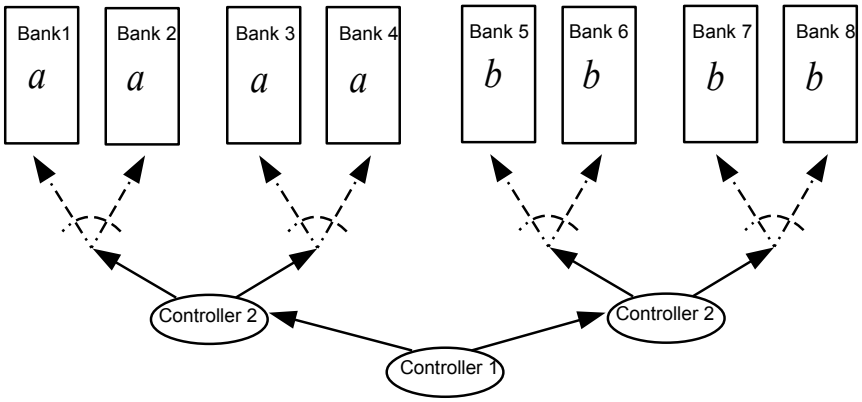

Figure 2: Storing $(\mathbf{a}, \mathbf{b})$ using a 4-repetition based solution.

Now, instead of performing a 4- repetition, we use a XOR code to facilitate 2 simultaneous reads as described in Fig. 3. Note that we also have to satisfy 1 write with 2 simultaneous reads. For this, we simply perform 2-repetition of XOR based scheme as shown in Fig. 4.

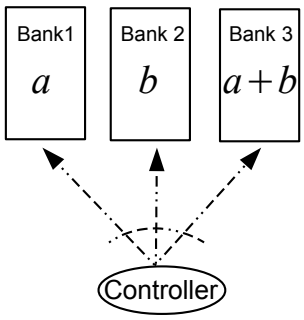

Figure 3: A XOR based solution for 2 simultaneous reads.

For the memory designed in Fig. 4, 2 simultaneous data read requests are forwarded to one group and 1 write request is forwarded to another group. The selection of a group for read requests is performed by the controller based on the status of memory banks. (Since write requests goes to only 1 group, the controller sends read requests to that group which contains the most updated version of requested data.)

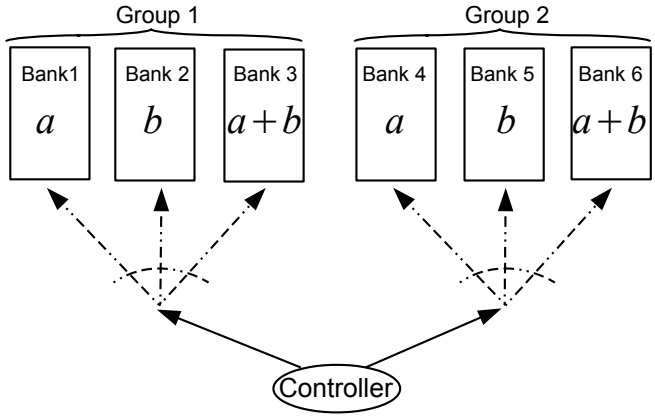

Figure 4: A XOR based design for 2-read and 1-write memory.

This scheme uses 6 memory banks to store data worth of 2 memory banks. This corresponds to rate $\frac{1}{3}$. The area required by this design is 6 area units for 6 memory banks and additional space of controller circuitry. Therefore, this simple XOR based scheme saves space of 2 memory banks as compared to 4-repetition scheme.

As described above, using special classes of codes, e.g., locally repairable codes, regenerating codes may give more attractive solutions when we are coding across multiple data blocks (not just 2 as assumed in the schemes described above).

Moreover, in the XOR based design in Fig. 4, we have used 2-repetition to accommodate additional write request. This is done keeping the worst case I/O request pattern in mind. We can consider the models, where we allow for a small delay during reads in some cases. After a write requests, the processors do not always request the data updated in the previous cycle. In this case, we can buffer some write requests and clear them when the opportunity arise, i.e., when the memory banks corresponding to a write request are not being accessed for reads, we can perform this write on memory banks. Moreover, a read request can potentially be served from the buffer itself when a write request for the data to be read from previous cycles is present in the write buffer. In this model, a read delay may occur only when the write buffer gets full and incoming read requests can not be served directly from the buffer. In this case, the write buffer needs to be emptied to accommodate an incoming write request and the current read requests may have to wait if they cause bank conflict the ongoing writing process. Similar ideas in RAID (with non-volatile cache) literature [4, 5].

In addition, designing reading friendly codes that also have small writing overhead (or update efficiency) is an interesting problem in the context of multi-port memories.

References

[1]] Jonathan Y. Zhang, "SRAM Operation and Multi-Port SRAM Realization," *Wireless Baseband, Futurewei U.S. R&D.*

[2] A. G. Dimakis, P. Godfrey, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage system", in *Proc. of IEEE INFOCOM*, May 2007.

[3] A. S. Rawat, O. O. Koyluoglu, N. Silberstein, and S. Vishwanath, "Optimal locally repairable and secure codes for distributed storage systems," *CoRR, abs/1210.6954*, 2012.

[4] A. Thomasian and C. Liu, "Performance comparison of mirrored disk scheduling methods with a shared non-volatile cache," *Distributed and Parallel Databases*, vol. 18, pp. 253 281, 2005.

[5] B. S. Gill and D. S. Modha, "WOW: wise ordering for writes-combining spatial and temporal locality in non-volatile caches," in *Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies (FAST'05)*, pp. 129 142, 2005.