

# Week 6 Progress Report

Tim Russell

*Department of Electrical and Computer Engineering  
University of Maine  
Orono, USA*

tgrussell@eiu.edu

## I. INTRODUCTION

### A. *Magic Dragon Background*

Trudging ever-onward with the work of developing a viable *offline* algorithm, I spent the bulk of this week trying a radically different approach to solving the problem. As noted before in earlier weeks, one way to achieve energy savings in memory to engineer concurrent DMA operations on the same memory chip. Since the same amount of energy is expended whether one or multiple concurrent operations are underway on a given memory chip, we maximize the benefit of energy usage by piggy-backing multiple memory requests on a chip that is already active. In past weeks, my focus, stemming from this observation, was to have *Magic Dragon* choose the chip which was going to be active the greatest amount of time during the completion of the current cache miss as its victim. In that way, the chip, which was already engaged in a DMA operation due to a recent, previous cache hit or miss could also support the new cache miss.

### B. *MD Problems*

An unaddressed situation with my algorithm is that I made only energy-based cache replacement decisions. That is, all of the replacement decisions were made with the sole aim of conserving energy. I made no provision for monitoring performance or for making trade-offs between performance and energy conservation. I made some attempts to develop metrics to address these issues, but with little success. This, along with some advice from Jianhui, led me to try a new approach.

## II. *Slipper Lizard* ALGORITHM

### A. *Introduction: Blocks Need Friends*

Running with the idea of overlapping DMA operations, Jianhui proposed a new strategy. If we viewed our cache *replacement* algorithm as a *placement* algorithm, we might try to intelligently place blocks together. The idea of forming bursty groups of block has been studied before [1] but here we take a different approach. Specifically, we might try to determine if groups of two or more blocks had a strong temporal correlation, meaning that if block  $A$  is accessed, then very often block  $B$  is accessed very soon after,  $A$  and  $B$  would have this strong temporal correlation. After identifying such groups of blocks, we could then place them in the following manner. Suppose that  $A$  and  $B$  are strongly correlated disk

blocks, and suppose that  $A$  is currently in the cache (in chip  $j$ ) but that  $B$  is not. When  $B$  is next accessed (a miss), we should place  $B$  in chip  $j$ , so that will be the victim chip. In doing so, we suppose that whenever  $A$  is accessed,  $B$  is also likely to be accessed soon. Hence, by placing both on chip  $j$ , we can take advantage of the possible DMA concurrency of two consecutive cache hits, or at least of keeping chip  $j$  active over a tighter time period, allowing other memory chips to stay idle longer.

### B. *Forming Block Groups*

One clear way to form these block groups is the following. Suppose that we know the entire request sequence (as in the *offline* case). Then we can use a *sliding time window* to evaluate temporal correlation. By *sliding*, we mean that move the window along so that it encompasses the current request under consideration ( $a_i$ ) and extends some time distance  $\theta_m$  (i.e., the *time window*). Suppose that  $a_i$  is a request for block  $A$ . For each block requested within the time window, we suppose that there is some temporal correlation between the two. From this analysis, we can construct an undirected, weighted graph. Each node in the graph is a block number, and there is an edge between any two nodes if there is a temporal correlation between the two. Further, the edges are of uniform length 1, and the weight of an edge  $\overline{AB}$  is the number times that  $A$  and  $B$  are found to have a temporal correlation.

Now suppose that the number of blocks that can be held in any memory chip is  $b$ . After building the above graph over the entire request sequence, we can then form block groups by a greedy algorithm. First, find the maximum-weight path of length  $b-1$ . Form a group from the nodes along that path, and then remove those nodes and their associated edges. Again, find the maximum-weight path of length  $b-1$ , form the group of nodes, and remove the nodes from the graph. Continue the processes until the number of groups formed is equal to the number of memory chips.

### C. *Implementation*

We borrow the overall structure of the 2Q algorithm [2] in addressing the formation of block groups. That is, we utilize two LRU queues and a ghost buffer, though the mechanics of our approach differs widely from that of the actual 2Q algorithm.

## REFERENCES

- [1] F. Chen and X. Zhang, "Caching for bursts (c-burst): Let hard disks sleep well and work energetically," in *International Symposium on Low Power Electronics and Design*. Bangalore, India: Indian Institute of Science, Aug. 2008.
- [2] T. Johnson and D. Shasha, "2q: A low overhead high performance buffer management replacement algorithm," in *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 439–450.