

# Week 3 Progress Report

Tim Russell

*Department of Electrical and Computer Engineering  
University of Maine  
Orono, USA  
tgrussell@eiu.edu*

## I. INTRODUCTION

This week began with an introduction to dynamic programming [1], a possible avenue of approach to a solution to our problem of energy-efficient cache management. I started with a reading of chapter 10 of the textbook, *Computer Algorithms* [2]. The chapter was fully devoted to the topic of dynamic programming, and investigated a few applications to common problems in mathematics and computer science, including matrix multiplication and binary search tree optimizations. The gist of dynamic programming is take a problem and break it down into smaller subproblems. These subproblems are then tackled recursively until the researcher arrives at atomic problems that can be easily handled. Next, the array of problems is evaluated in a manner equivalent to a depth first search. Instead of naïvely recursing through the problems and quite possibly duplicating work, one uses *memoization* [3]; that is, one designs the algorithm so that when the solution to a subproblem is found, this solution is then recorded. The hope was that, following the dynamic programming approach, the problem of developing an energy-efficient page replacement algorithm could be broken down into subproblems.

Next, I analyzed a recently proposed energy-aware cache replacement algorithm [4] to determine its algorithmic complexity. The algorithm purports to use dynamic programming, so I thought that it might be a good example of that technique applied to our problem. Upon examination, I found numerous notational problems and irreconcilable difficulties in the system of equations presented. Additionally, the equations represented only the subcases for calculating energy expenditure of memory chips; no algorithm was actually given in the article. Finally, the use of the dynamic programming technique is notably absent from the approach. My analysis of Jianhui's work is as follows.

## II. ANALYSIS

### A. Notation

I add the following to the notation used in Jianhui's [4] paper:

- $m$  = the number of memory chips
  - $b$  = the number of distinct blocks on the disk
  - $j$  = the number of DMA "slots"
  - $n$  = the number of memory accesses
  - $d$  = the number of disk blocks a memory chip can hold
- \* Indexes start from 0

### B. Equation Cleanup

Next, I focused on reforming his equations. Jianhui's notation was difficult to understand, and many parts of the equations were very jumbled together. Additionally, equations (3) and (4) contained tests for (in)equality and assignments to values, making it difficult to ascertain if equality tests or assignments were desired. To the best of my ability, I differentiated between the two. I kept the tests in the equations (section II-E) and moved the assignments to the algorithm (section II-F).

### C. Algorithm Creation

There wasn't really an "algorithm" for the computation, so I created one that closely followed the algorithm given in *Power-Aware Storage Cache Management* (Zhu & Zhou).

### D. Analysis

It should be clear that for each iteration of the innermost `for`-loop, the first two cases are  $\mathcal{O}(1)$ , since they both simply involve some evaluations and an assignment. Similarly, the third and fourth cases involve constant-time evaluations and assignments. These two cases also involve minimizations, but both are on the order of the number of DMA "slots" ( $j$ ). This inner `for`-loop is evaluated for each possible combination of cache configurations for a single memory chip. Since there are  $b$  distinct disk blocks, and each memory chip can hold  $d$  blocks, this gives a worst-case total of  $b \times d \times j$  minimizations.

Of the next two cases, the second is more complex. To see this, note that both involve summations of  $f_k$  over all memory chips. The first of the two (from equation 5 above), though, involves a minimum which evaluates immediately. The minimum  $CT'_h$  is simply  $CT_h[CT_h.S_{\text{free}}]$ . As noted, the other case also has a summation, but its minimization is more complex. Specifically, the algorithm must look at all cache configurations and choices for  $r$  (which memory chip will be have a page replacement).  $r'$  can take on  $m$  values, i.e. the index of any memory chip. Each chip has  $b$  possible configurations after a replacement, so there are a total of  $m \times b$  different cache configurations. Additionally,  $CT'_k$  can take on any of  $j$  different values, depending on which DMA slot is used in the memory access. Hence,  $f_k(C'_k, CT'_k, r', i)$  can have a total of  $m \times b \times j$  values. This summation ranges from 1 to  $m$ , so this portion of the algorithm runs in  $\mathcal{O}(m^2 \times mbj) = \mathcal{O}(m^3bj)$ .

Finally, the outer for-loop runs from  $n$  times, so the overall complexity of this algorithm is  $\mathcal{O}(m^3bjn)$ .

### E. Equations

$$f_k(C_k, CT_k, r, i+1) = \begin{cases} f_k(C_k, CT_k, r, i) & \text{if } (i+1) - CT_k[S_{\text{busy}}] > \alpha \wedge a_{i+1} \notin C_k \wedge r \neq k & (1) \\ f_k(C_k, CT_k, r, i) + 1 & \text{if } (i+1) - CT_k[S_{\text{busy}}] \leq \alpha \wedge a_{i+1} \notin C_k \wedge r \neq k & (2) \\ f_k(C_k, CT'_k, r, i) + 1 & \text{if } a_{i+1} \in C_k \wedge r \neq k & (3) \\ f_k(C'_k, CT'_k, r, i) + 1 & \text{if } a_{i+1} \in C_k \in \text{repl}(C'_k, a_{i+1}) \wedge r = k & (4) \end{cases}$$

$$F(i+1) = \begin{cases} \sum_{k=1}^m f_k(C_k, CT_k, 0, i) + \min_{CT'_h} f_h(C_h, CT'_h, 0, i) & \text{if } a_{i+1} \in C_h \Leftrightarrow a_{i+1} \in C & (5) \\ \min_{C'_k, CT'_k, r' | r' \neq 0} \left( \sum_{k=1}^m f_k(C'_k, CT'_k, r', i) \right) & \text{if } a_{i+1} \notin C & (6) \end{cases}$$

### F. Algorithm

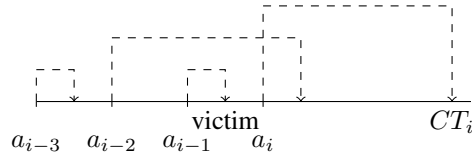
```

for  $i = 0$  to  $n - 1$  do
  for  $k = 0$  to  $m$  do
    for all  $C_k$  such that  $C_k$  is a set of  $d$  disk blocks do
      if  $(i+1) - CT_k[S_{\text{busy}}] > \alpha \wedge a_{i+1} \notin C_k \wedge r \neq k$  then
         $f_k(C_k, CT_k, r, i+1) \leftarrow f_k(C_k, CT_k, r, i)$ 
      else if  $(i+1) - CT_k[S_{\text{busy}}] \leq \alpha \wedge a_{i+1} \notin C_k \wedge r = k$  then
         $f_k(C_k, CT_k, r, i+1) \leftarrow f_k(C_k, CT_k, r, i) + 1$ 
      else if  $a_{i+1} \in C_k \wedge r \neq k$  then
         $f_k(C_k, CT_k, r, i+1) \leftarrow f_k(C_k, CT'_k, r, i) + 1$ 
         $CT'_k[S_{\text{free}}] \leftarrow (i+1) + \theta_m$ 
         $CT'_k[S_{\text{free}}] \leftarrow \{i | CT'_k[i] == \min(CT_k)\}$ 
      else if  $a_{i+1} \in C_k \in \text{repl}(C'_k, a_{i+1}) \wedge r = k$  then
         $f_k(C_k, CT_k, r, i+1) \leftarrow f_k(C'_k, CT'_k, r, i) + 1$ 
      end if
    end for
  end for
  if  $a_{i+1} \in C_h \Rightarrow a_{i+1} \in C$  then
     $F(i+1) \leftarrow \min_{CT'_h} f_h(C_h, CT'_h, 0, i) + \sum_{k=1|k \neq h}^m f_k(C_k, CT_k, 0, i)$ 
  else if  $a_{i+1} \notin C$  then
     $F(i+1) \leftarrow \min_{C'_k, CT'_k, r' | r' \neq 0} \left( \sum_{k=1}^m f_k(C'_k, CT'_k, r', i) \right)$ 
  end if
end for

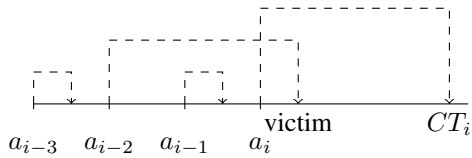
```

## III. GREEDY ALGORITHM

After my work analyzing Jianhui's algorithm, we became doubtful that dynamic programming would be a fruitful approach to our problem. We decided that a more viable alternative would be to develop a greedy algorithm to attack the problem. Baase and Van Gelder [2] provided a short background to the topic of greedy algorithms. With that small base, I made my first jab at a working algorithm. The MRU algorithm chooses as a victim the memory block which was mostly recently used. The terms *most recently used* refer to the time of memory request arrivals and their relation to current time. Hence, in the event of a cache miss, the victim chip is chosen as shown in the first figure below. Suppose the memory chip  $j$  serviced request  $a_{i-1}$ . Then MRU would choose the memory chip  $j$  as the victim chip and the most recently used block as the victim block.



I propose an alternative algorithm which uses the following approach. When request  $a_i$  (which is a cache miss), arrives (at time  $t_i$ ) we calculate  $t_c = t_i + \theta_{dm}$ , where  $\theta_{dm}$  is the time required to service a disk and memory access (i.e. to service a cache miss). The victim is then chosen to be the chip for which the completion time of its last memory request is closest to  $t_c$ . Hence, if chip  $k$  serviced memory request  $a_{i-2}$  in the below figure, that chip would be chosen as the victim chip.



The victim block in the chosen chip could then be chosen by an appropriate secondary algorithm, such as 2Q, LRU/k, etc. The objective is to generate energy savings by maximizing concurrent DMA operations. By choosing a chip whose latest DMA operation will complete as closely to the  $i^{\text{th}}$  request, we hope to "piggy-back" onto the already active chip's DMA operations. Clarifications of these ideas and choice of block replacement algorithm will follow in next week's report.

#### REFERENCES

- [1] (1999, Nov.) Algorithms and theory of computation handbook. Found at Dictionary of Algorithms and Data Structures. [Online]. Available: <http://www.nist.gov/dads/HTML/dynamicprog.html>
- [2] S. Baase and A. V. Gelder, *Computer Algorithms: Introduction to Design and Analysis*, 3rd ed. Boston, MA: Addison-Wesley Longman, 2000.
- [3] (2007, Oct.) Dictionary of algorithms and data structures. [Online]. Available: <http://www.nist.gov/dads/HTML/memoize.html>
- [4] J. Yue, Y. Zhu, and Z. Cai, "Energy- optimal buffer cache replacement," 2008, jianhui's dynamic programming equations.