

Week 4 Progress Report

Tim Russell

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

tgrussell@eiu.edu

I. INTRODUCTION

This week found my efforts focused on more fully developing the approach to my algorithm which was started last week. After reviewing some background on greedy algorithms [1], [2] and noting the greedy approach to similar problems in Zhu and Zhou [3] and Yue [4], I've determined that a greedy algorithm approach to the problem of energy-efficient cache replacement algorithms may generate good results. It has been noted [1], [3] that greedy algorithms do not always yield optimal strategies. However, several well-known problems, such as the *minimum spanning tree* and *short-path* [1], as well as *Huffman encodings* [2], have been solved optimally by greedy algorithms.

Additionally, greedy algorithms can yield initial approximations to optimal solutions, which may be the best result for which we can hope reasonably hope. To wit, Brehob, et. al [5] have proved that many classes of caching problems are NP-hard. It remains to be seen whether this result extends directly to our problem. If so, however, it means that, "no polynomial-time optimal replacement strategy exists for most nonstandard caches unless $P = NP$." [5]. In this light, suboptimal strategies for cache replacement will be the highest achievable mark of success. Albers, Arora, and Khanna [6] also evaluated some general caching problems to determine their hardness, showing that some specific models, such as the **Bit Model** and the **Cost Model** may be optimally solved in polynomial time.

The algorithm in section III seeks to improve energy efficiency by capitalizing on concurrent DMA operations. Modern computers utilize DMA (Direct Memory Access) to offload memory I/O operations from the CPU. Besides freeing the CPU to perform other actions during memory I/O, the DMA paradigm allows multiple concurrent access to memory to take place. In an abstracted model, a memory chip can be thought to have some number of DMA "slots" [4]. A memory chip with four DMA "slots", then, would be able to support up to four concurrent reads and/or writes, all of which can arrive at different times. Beyond the obvious performance benefit that such concurrency allows, energy efficiency of memory can improve under this model, because the same amount of energy is expended no matter how many DMA operations are currently underway on a given memory chip. In short, for a given block request a_i (see section II) which is a *cache miss*, the algorithm in section III looks at the

current DMA operations underway on each memory chip and, using knowledge of the length of time required for a disk and memory access, finds the chip which will allow the greatest concurrency (or *overlap*) when servicing both the operation(s) underway and a_i .

II. NOTATION

The following notation will be used in section III.

- Memory chip (or chip): Total system memory and cache is actually composed of multiple chips which act in concert. This algorithm supposes the rambus memory model, in which the power states of each chip can be handled independently of the others.
- a : A memory request.
- n : The number of memory requests (i.e., requests appear in the sequence $a_0, a_1, a_2, \dots, a_{n-1}$).
- k : The number of memory chips.
- a_i : The i^{th} memory request (denoting the current request, which is a cache miss).
- $L_j(a)$: The *leader* miss. The closest (with respect to time) cache miss to a , but occurring *before* a , which was serviced by memory chip j . When it is clear from the context which chip is meant, the index j may not be used.
- $F_j(a)$: The *follower* miss. The closest (with respect to time) cache miss to a , but occurring *after* a , which was serviced by memory chip j . When it is clear from the context which chip is meant, the index j may not be used.
- τ : A predefined time threshold, past which memory chips transition from the *active* power state (at which a memory chip must be to service any request) to a lower power state to conserve energy.
- $E(t)$: The energy consumption of a memory chip which is performing some operation during the time period t . Note that this value will vary with the power management model used by the memory chip.
- EP_j : The energy penalty incurred by choosing chip j to be the victim chip (thereby breaking its idle interval into two subintervals).
- θ_m : The amount of time required for memory access (as in a cache hit).
- θ_{dm} : The amount of time required for disk and memory access (as in a cache miss).

- $AT_j(a)$: The arrival time of a memory request a on chip j . When it is clear from the context which chip is meant, the index j may not be used.
- $CT_j(a)$: The completion time of a memory request a on chip j ; that is, the time at which the DMA operation corresponding to a given memory access on chip j is completed. This notion will be used most often in the context of cache misses. When it is clear from the context which chip is meant, the index j may not be used.
- Δ_j : The “distance” (with respect to time) between the completion of L_j and a_i . Specifically, $\Delta_j = CT(a_i) - CT(L_j(a_i))$.
- Ω_j : The amount (with respect to time) by which the DMA operations on chip j servicing L_j and a_i will overlap if chip j services a_i (i.e. chip j is chosen to be the victim chip). In the online implementation, we see that $\Omega_j = \theta_{dm} - \Delta_j$.

III. The Magic Dragon CACHE REPLACEMENT ALGORITHM

A simple approach to the algorithm for choosing the victim chip i presented next, and is fairly straightforward (the author hopes).

A. Algorithm

```

for  $i = 0$  to  $n - 1$  do
  for  $j = 0$  to  $k - 1$  do
     $CT_j(a_i) \leftarrow AT(a_i) + \theta_{dm}$ 
     $\Delta_j \leftarrow CT(a_i) - CT(L_j(a_i))$ 
  end for
  Choose the victim chip to be chip  $v$  such that
   $\Delta_v = \min_j \Delta_j \Rightarrow \Omega_v = \max_j \Omega_j$ .
end for

```

B. General Cases

We consider two general situations involving a cache miss a_i which help clarify our approach.

1) *Non-bursty Access*: In this case, the completion time of the leading misses for all k memory chips is prior or equal to the arrival time of a_i . Symbolically, it can be seen that the following holds for all $j \ni j \in \{0, 1, 2, \dots, k - 1\}$:

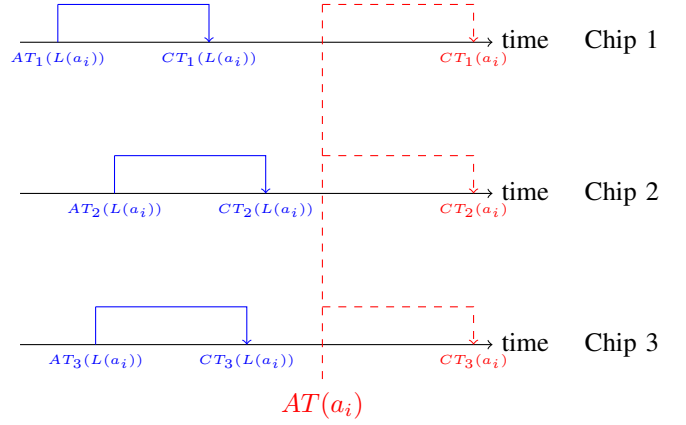
$$\begin{aligned}
 CT(L_j(a_i)) &\leq AT(a_i) \\
 &\leq CT_j(a_i) - \theta_{dm}
 \end{aligned}$$

so that

$$\begin{aligned}
 \Delta_j &= CT(a_i) - CT(L_j(a_i)) \\
 &\geq CT(a_i) - (CT_j(a_i) - \theta_{dm}) \\
 &\geq \theta_{dm}
 \end{aligned}$$

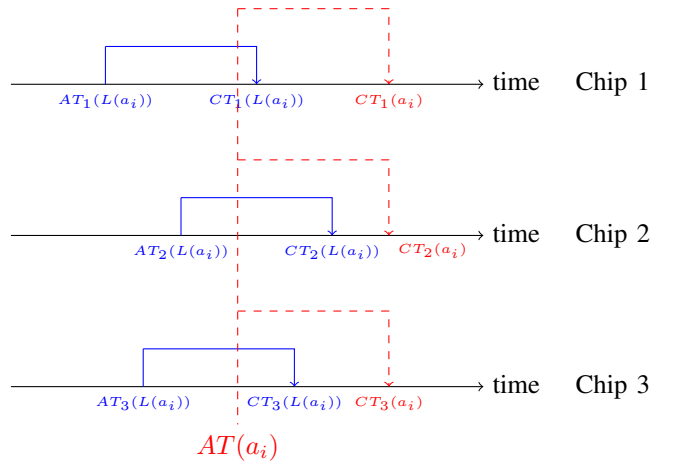
In this case, no energy savings can be achieved in our scheme, because no DMA overlapping is possible. All k chips have completed servicing their last cache miss ($L(a_i)$) before the arrival of a_i , so the DMA operation corresponding to a_i cannot

“piggy-back” on any other concurrent operation. This case is shown graphically below (suppose for simplicity that $k = 3$).



In this case, we can amend the algorithm so that it makes a better choice for victim. Since we can glean no energy savings from DMA concurrency, we will instead choose to maximize the energy savings offered by memory chips’ idle times. Under modern power management schemes, memory chips are allowed to transition to lower power state(s) after a certain amount of time being idle, τ . Transitioning to these power states conserves energy, but an additional energy and time cost is incurred when a powered-down memory chip is reaccessed for another read or write. Hence, we can achieve greater savings by choosing the chip for which the *energy penalty* is minimal. We define this energy penalty as Zhu and Zhou [3] do. With this amendment in hand, we examine the other common case.

2) *Bursty Access*: This case covers the entire range of burstiness, i.e. from a single overlapping DMA operation to tightly clustered memory accesses. In this case, for at least one of the k memory chips, the completion time of the leading miss will occur after the arrival time of a_i . Symbolically, $\exists j \ni \Delta_j < \theta_{dm}$. We graphically illustrate a simple example of this case graphically below.



In this case, it is clear that we can achieve the greatest energy savings via DMA concurrency by choosing chip 2 to be the victim chip.

C. Algorithm Revised

With our note in section III-B1, we can slightly adapt our algorithm to accommodate non-bursty access patterns, allowing greater flexibility for the algorithm.

```

for  $i = 0$  to  $n - 1$  do
  for  $j = 0$  to  $k - 1$  do
     $CT_j(a_i) \leftarrow AT(a_i) + \theta_{dm}$ 
     $\Delta_j \leftarrow CT(a_i) - CT(L_j(a_i))$ 
  end for
   $v \leftarrow$  index of the chip for which  $\Delta_v = \min_j \Delta_j$ 
  if  $\Delta_v < \theta_{dm}$  then
     $v$  is the index of the victim chip.
  else
    for  $j = 0$  to  $k - 1$  do
       $EP_j \leftarrow E(AT_j(a_i) - CT_j(L_j(a_i))) +$ 
       $E(AT_j(F_j(a_i)) - CT_j(a_i)) -$ 
       $E(AT_j(F_j(a_i)) - CT_j(L_j(a_i)))$ 
    end for
     $v \leftarrow$  index of the chip for which  $EP_v = \min_j EP_j$ .
  end if
end for

```

D. Optional Extension

An unspoken assumption in the above work is that we have no DMA resource conflicts. That is, we assume that, once a victim chip is chosen, it is sure to have a DMA “slot” that is available for use in servicing a_i . We don’t actually have that assurance in real-world situations, however, so we introduce another slight amendment to the algorithm in section III-C (pseudocode omitted for brevity). In the case that the first-choice victim has no available DMA slots, we can simply proceed to the next-best candidate memory chip, with this process continuing until either a suitable victim is found or we exhaust all choices. In the latter event, no chip can service the request anyway, so it must wait to be serviced.

IV. FUTURE WORK

The most immediate extension to the above work is that I must incorporate cache hits into my analysis. I had originally adopted that approach, but I was worried that making the determination for victim chips based on the overlapping access times of both hits and misses would adversely affect the cache hit ratio. For a more complete analysis of the efficacy of this algorithm, however, I will work the cache hits back into the memory access timelines. An expected result is that time complexity of the algorithm will slightly increase, but greater energy savings may develop/

Additionally, I will develop metrics to evaluate the efficiency—both in terms of energy savings and cache performance relative to common page replacement algorithms—of the *Magic Dragon*.

REFERENCES

- [1] S. Baase and A. V. Gelder, *Computer Algorithms: Introduction to Design and Analysis*, 3rd ed. Boston, MA: Addison-Wesley Longman, 2000.
- [2] J. Erickson, “Non-lecture a: Greedy algorithms,” Creative Commons Attribution-NonCommercial-ShareAlike 2.5, 2006.

- [3] Q. Zhu and Y. Zhou, “Power-aware storage cache management,” *IEEE Transactions on Computers*, vol. 54, no. 5, pp. 587–602, May 2005, doi: 10.1109/TC.2005.82.
- [4] J. Yue, Y. Zhu, and Z. Cai, “Energy- optimal buffer cache replacement,” 2008.
- [5] M. Brehob, S. Wagner, E. Torng, and R. Enbody, “Optimal replacement is np-hard for nonstandard caches,” *IEEE Transactions on Computers*, vol. 53, no. 1, pp. 73–76, Jan. 2004, iSSN:0018-9340.
- [6] S. Albers, S. Arora, and S. Khanna, “Page replacement for general caching problems,” in *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999, pp. 31–40.