ALTERNATIVE FORMULATIONS OF THE PAGING PROBLEM FOR CACHE WITH READ THROUGH

Cathy Jo Linn

Department of Computer Science
Vanderbilt University
Nashville, Tennessee 37235

Cache memory is now widely accepted as a cost effective way of improving system performance. Significant reductions in the average data access time have been achieved using very simplistic paging algorithms, such as LRU, implemented in hardware. In this paper we wish to investigate more sophisticated algorithms for the management of intelligent cache systems.

Keywords: cache management; read through; optimal algorithms; Belady's optimal paging algorithm; graph shortest distance problem.

1. CACHE WITH READ THROUGH

The proposed model of a cache includes the option of read through (Figure 1). If desired, the processor can fetch directly from primary memory instead of first paging into cache. We observe that transfering a page, P1, into a full cache requires the writing out of a page, P2. If P2 is then immediately referenced, it must be reloaded into cache. In a situation in which P1 is never referenced again, performance could be improved by using read through to access P1 and leaving P2 in the cache. Current page replacement algorithms are not immediately applicable to this new hardware. When a cache page fault occurs, we now must not only decide whether to bring in a page or use the read through option, but also which page to replace.

2. ALGORITHMS

We will now present and analyze two approaches to determining an optimal paging scheme for cache with read through. In these algorithms we will assume full knowledge of the future reference string. Since we are dealing with transfers between cache and primary memory, we will also assume no rotational delays. This will mean that if a 1-page transfer takes time t, a n-page transfer will take time nt.

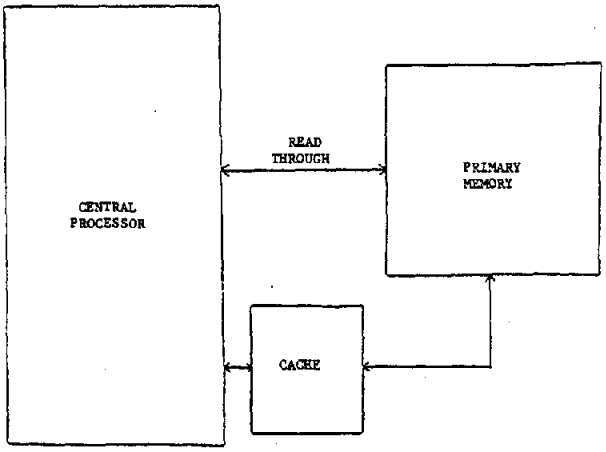

Fig. 1. Cache with Read Through.

Let k be the number of cache page frames available and let M be the set of pages available, {1,2,3,...m}. Then the reference string will be :

$$\omega = x_1, x_2, \ldots, x_n$$

where $x_i \in M$ and represents the page being referenced at time t_i. Let $S_i \subseteq M$ be the set of pages in cache at time t_i.

2.1 AN IPP FORMULATION

Given the above definitions we can now formalize the problem as an Integer Programming Problem. The task is to determine S_i for i=1 to n such that $\sum_{i=1}^{n} Z_i$ is minimized. Z_i is the cost incurred in moving from t_{i-1} to t_i and

$$Z_i = \begin{cases} P+C & \text{if } x_i \notin S_{i-1} \text{ and } x_i \in S_i \\ R & \text{if } x_i \notin S_{i-1} \text{ and } x_i \notin S_i \\ C & \text{if } x_i \in S_{i-1} \text{ and } x_i \in S_i \\ P+R & \text{if } x_i \in S_{i-1} \text{ and } x_i \notin S_i. \end{cases}$$

The S_i's are subject to the constraints:

$|S_i| \le k, \quad 0 \le i \le n$

$S_0 = \phi$, and

$S_i \in S_{i-1} + x_i$.

We include the fourth cost for completeness only since it is clear that it will never occur in an optimal solution. The last constraint is included so that demand paging only is allowed. We also note that if P+C≤R we would never choose to use the read through option.

Our solution to the IPP is based on the following observation. Once we decide which references are to be accessed via the read through option, we have simplified the situation to that of a standard cache where all remaining references must be made from cache. Also, for optimality we need only page on demand, since it has been shown that prepaging will offer no benefit [COFF73].

Belady [BELA66] has presented an optimal algorithm for this simplified case. We depict this algorithm in Figure 2. The execution time of the initialization For-loops is O(|M'|)=O(n). The rest of the algorithm is contained in the next For-loop and executes n times. In this loop we execute either the `then` or the `else` portion of the 'If' statement. The `then` portion takes execution time of O(|S|)=O(k) due to the set assignment operation. The `else` section itself contains a loop that executes O(|S|)=O(k) times. We will separate out the while-loop contained in that section. It will execute at most a total of n times since there are only n elements in all of the Appear lists. We can now see that the entire algorithm is O(2n+nk)=O(nk).

An algorithm to determine the paging scheme resulting in the minimum cost for cache with read through would proceed as follows:

(1) Pick a subset of references from the reference string. These are the references that will be referenced via the read through option.

(2) Apply Belady's algorithm to the remaining references accumulating the cost. Add to this cost R*(number of read throughs). We note that in some cases this may result in using read through to access data that is already in cache. Clearly, these cases will not be among the optimal cases. They will not, however, prevent the algorithm from locating an optimal solution.

(3) Repeat step 1 and 2 until all 2^n subsets have been processed, keeping track of the subset resulting in the minimum cost.

A program implementing this algorithm is given in Figure 3. The execution time of this program is $O(nk2^n)$.

2.2 A GRAPH FORMULATION

The cache management problem can also be formulated as a graph shortest-distance problem. Such a formulation of the problem results in an entirely different type of algorithm.

The graph representing one instance of the problem will consist of n+2 stages corresponding to time t_0 through time t_{n+1}. The vertices that make up $stage_i$ for $1 \le i \le n$ represent all the possible subsets of pages in cache. Let $v_{i,j}$ be the vertex in stage i representing that the set of pages, S_j, is contained in cache at time t_i. (There are $\binom{m}{k}+\binom{m}{k-1}+\ldots+\binom{m}{0}=O(km^k)$ of these subsets.) At t_0 there is one state with zero pages in cache. At t_{n+1} there is also one state.

The arcs of this directed graph are constructed in the following manner. From each vertex $v_{i,j}$, (for $0 \le i \le n$), create an arc leaving $v_{i,j}$ and entering the following vertices at stage i+1:

(1) $v_{i+1,j}$, the vertex representing no change in the set of pages in cache, and

(2) if $|S_j| < k$ then the vertex representing the cache contents of $S_j + \{x_i\}$ at time t_i

(3) if $|S_j| = k$ then the vertices representing the cache contents of $S_j + \{x_i\} - \{y\}$ for all $y \in S_j$.

The distances on these arcs are the same as the costs stated earlier: P+C if x_i is paged in, C if $x_i \in S_{i-1}$ and $x_i \in S_i$, or R if $x_i \notin S_{i-1}$ and x_i is not paged in. There will be no arcs with a distance of P+R since we do not include these in the construction.

```
* Form a set M'⊆M of the memory pages that appear in ω

     M'={i|∃x_j=i for 0≤j≤n};

* Search the reference string x_1-x_n forming a list
* of the times each  member of M' is referenced.

     For i=1 to n do
          Add i to the end of Appear_x_i list;

     For i∈M' do
          Add ∞ to end of Appear_i list;

     Cost=0;
     S_0=empty;

* Proceed down the reference list, changing the
* set of pages in cache, S_i, and accumulating the cost.

     For i=1 to n do
          If x_i∈S_i-1 then

* If page is in cache the just access it from cache

               begin
               Cost=Cost+C;
               S_i=S_i-1;
               end

          else

* else need to page it in, so find the member of S_i-1 that
* is referenced farthest in the future and replace it

               begin
               Cost=Cost+P+C;
               Max=0;
               For j∈S_i-1 do

                    begin
*                   Our lists will tell us when the next
*                   occurence is but first we must delete any
*                   occurences we have already past
                    While First-element(Appear_j)<i do
                         Appear_j=Appear_j-First-element(Appear_j);

                    If First-element(Appear_j)>Max then
                         begin
                         Savemax=j;
                         Max=First-element(Appear_j);
                         end
                    end
               S_i=S_i-1-{Savemax}+{x_i};
               end
```

Fig. 2. Belady's Optimal Demand Paging Algorithm

```
Mincost=∞
Save=empty

For S∈2^{i|1≤i≤n} (the power set of reference indices) do:
     begin
     Form ω'=x'1,x'2,...x'n-|S|

          by deleting the references whose indices are member of S.
     Call Belady's Algorithm for ω', returning  Cost
     Cost=Cost+|S|*R
     If Mincost>Cost then
          begin
          Save=S
          Mincost=Cost
          end
     end
```

Fig. 3. Optimal Paging Algorithm for Cache with Read Through

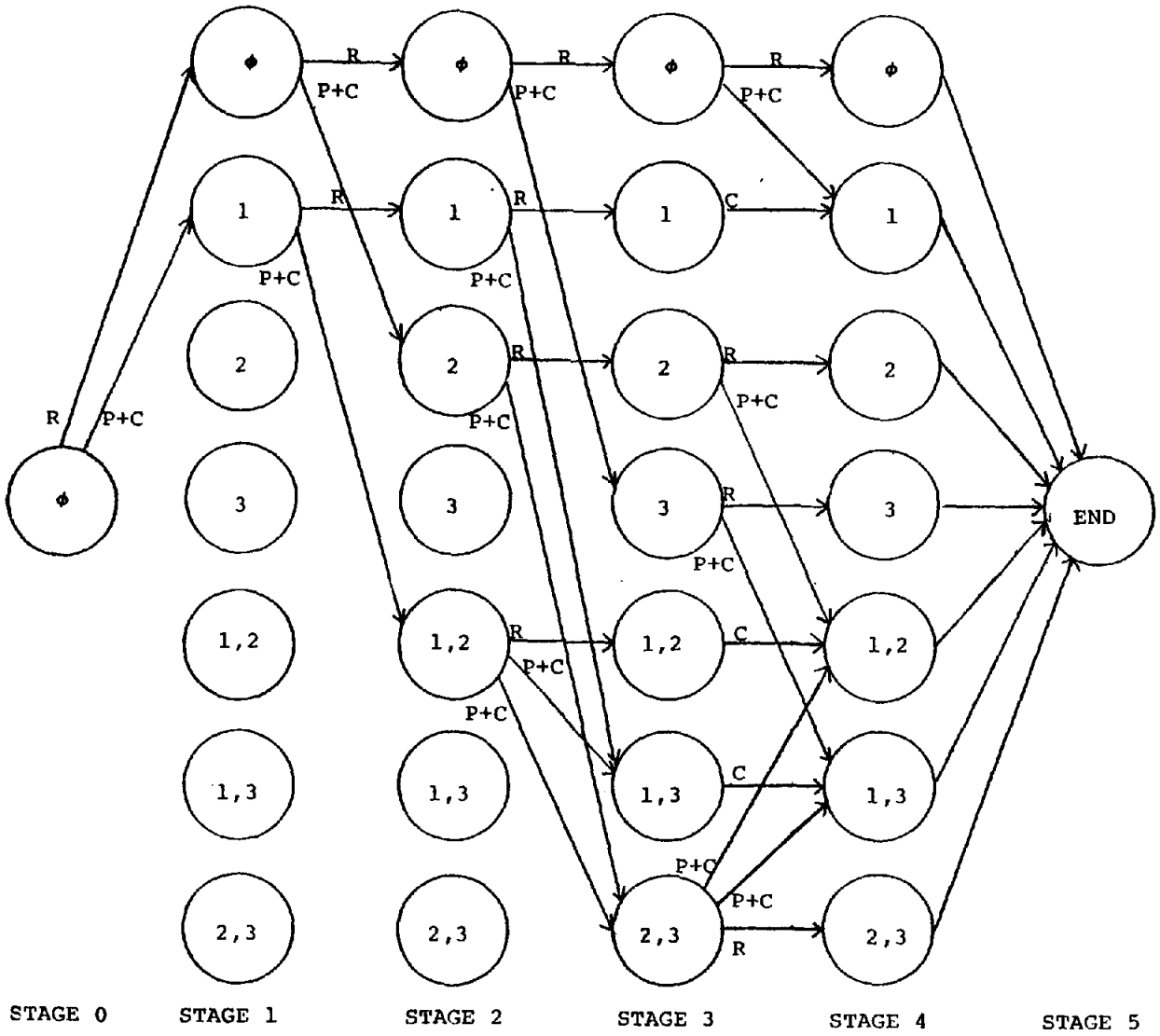

Fig. 4. Graph for ω=1231, M={1,2,3}, and k=2.

The vertices in stage n each have one arc leaving and entering the vertex at stage n+1. The distances on these arcs are 0.

The arcs in the above construction include all arcs corresponding to demand paging operations. We can create only these arcs because we know that a demand paging algorithm will give an optimal answer. Figure 4 shows the graph constructed for the reference string ω=1231, M={1,2,3}, and k=2.

A graph obtained from the above construction has $n*(\binom{m}{k}+\binom{m}{k-1}+\ldots+\binom{m}{0})$ +2 vertices or $O(nkm^k)$. It has no more than $(n-1)(k+1)(\binom{m}{k}+\binom{m}{k-1}+\ldots+\binom{m}{0})$ arcs or $O(nk^2m^k)$.

The problem now is to find the shortest distance from the vertex at stage 0 to the vertex at stage n+1. Dijkstra [DIJK59] provides a shortest distance algorithm whose execution time is $O(v^2)$, where v is the number of vertices. For our graph this would be $O(n^2k^2m^{2k})$. Since our weights can be integerized, we could alternatively apply Wagner's shortest distance algorithm for edge-sparse graphs [WAGN76]. The execution time of this algorithm is O(max(v,e,d)) where v is the number of vertices, e is the number of edges, and d is the maximum distance of any edge. For our graph this would be $O(e)=O(nk^2m^k)$.

3. CONCLUSIONS

This paper has investigated alternative formulations of the paging problem for cache with read through. The IPP formulation results in an algorithm that is linear with respect to the cache size but exponential with respect to the length of the reference string. The algorithms obtained from the graph formulation of the paging problem are linear with respect to the length of the reference string but exponential with respect to the cache size. We find this apparent tradeoff in execution time of the optimal paging algorithm quite interesting. We note that other similiar attempts to formulate this problem have resulted in algorithms with execution times that have this same characteristic.

REFERENCES

[BELA66] Belady, L. A. "A Study of Replacement Algorithms for a Virtual Storage Computer," IBM Systems Journal, 5, 1 (1966), 78-101.

[COFF73] Coffman, E. G. and Denning, Peter J. Operating Systems Theory, Englewood Cliffs: Prentice-Hall, Inc., 1973.

[DIJK59] Dijkstra, E. W. "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, 1, (1959), 269-271.

[WAGN76] Wagner, Robert A. "A Shortest Path Algorithm for Edge-Sparse Graphs," JACM, 23 1 (1976), 50-57.