Computer Architecture- CEN 502

Abstract— Two caching algorithms viz. Least recently used and Adaptive Replacement policy have been implemented in this project.

I. Introduction

Cache is a component that stores data so future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation, or the duplicate of data stored elsewhere. A *cache hit* occurs when the requested data can be found in a cache, while a*c ache miss* occurs when it cannot. Cache hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests can be served from the cache, the faster the system performs.

To be cost-effective and to enable efficient use of data, caches are relatively small. Nevertheless, caches have proven themselves in many areas of computing because access patterns in typical computer application exhibit the locality of reference. Moreover, access patterns exhibit temporal locality if data is requested again that has been recently requested already, while spatial locality refers to requests for data physically stored close to data that has been already requested.

Cache Replacement Algorithm

In computing, cache algorithms (also frequently called cache replacement algorithms or cache replacement policies) are optimizing instructions—or algorithms—that a computer program or a hardware-maintained structure can follow in order to manage a cache of information stored on the computer. When the cache is full, the algorithm must choose which items to discard to make room for the new ones. This is done to increase the hits thereby reducing the overall acess time.

We have implemented two caching algorithms :

1. Least Recently used
2. Adaptive Replacement cache

II. Least Recently Used Algorithm (LRU)

The algorithm keeps track of the most recently used and least recently used data. This can be done using an data structure of the size of the cache size. A data structure needs to be maintained to keep track of least recently used and most recently used elements.

I Implemented this using C++ and used STL Vector data structure. If a element referenced is found in the cache it is inserted at the beginning of the cache else if it is not found the least recently element used i.e at end is removed and the referenced element is inserted at the starting of the cache.

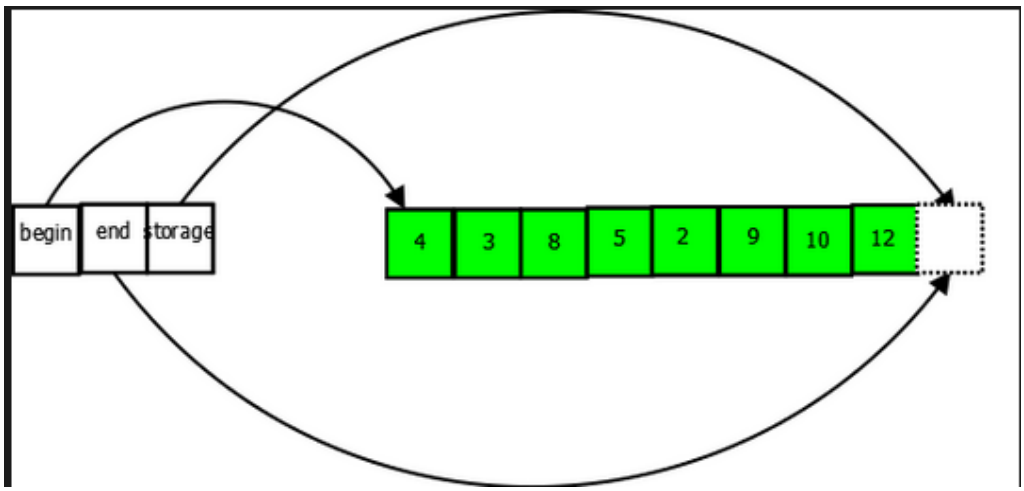

Fig 1:Internal Implementation of Vector

It is a dynamically allocated array of elements. Whenever we want an array in C++, a sequence of elements, vector is probably what we want to use. Vectors are good for random read access and insertion and deletion in the back (takes amortized constant time).her

Implementation:

A class Lru is created containing following data structures in the file lru.h:

- _Size , integer type to store size of cache given using command line argument
- _cache, vector type data structure to maintain the LRU cache
- _hit, integer type to keep track of the number of hits occurring for a particular trace file
- _miss, integer type to keep track of the number of hits occurring for a particular trace file
- hit ratio is printed using double data structure
- Truncating is used while printing hit ration upto two decimal places
- Hit ratio = hit/(hit+miss),this formula is used to compute hit ratio

Optimization:

Earlier it was implemented using an array but using vector greatly increases the speed .

.

III. Adaptive Replacement Cache

ARC improves the basic LRU strategy by splitting the cache directory into two lists, T1 and T2, for recently and frequently referenced entries. In turn, each of these is extended with a ghost list (B1 or B2), which is attached to the bottom of the two lists. These ghost lists act as scorecards by keeping track of the history of recently evicted cache entries, and the algorithm uses ghost hits to adapt to recent change in resource usage. Note that the ghost lists only contain metadata (keys for the entries) and not the resource data itself, i.e. as an entry is evicted into a ghost list its data is discarded. The combined cache directory is organised in four LRU lists:

1. T1, for recent cache entries.
2. T2, for frequent entries, referenced at least twice.
3. B1,ghost entries recently evicted from the T1 cache, but are still tracked.
4. B2, similar ghost entries, but evicted from T2.

T1 and B1 together are referred to as L1, a combined history of recent single references. Similarly, L2 is the combination of

Implementation:

Similar to LRU STL vector data structure is used in c++ for implementation of T1,T2,B1,B2 list
A class named Arc is created in arc.h contains the following variables:

- _Size , integer type to store size of cache given using command line argument
- _p, integer type for dynamic scaling of the cache
- _hit, integer type to keep track of the number of hits occurring for a particular trace file
- _miss, integer type to keep track of the number of hits occurring for a particular trace file
- _T1, STL vector to implement T1
- _T2, STL vector to implement T2
- _B1, STL vector to implement B1
- _B2, STL vector to implement B2
- hit ratio is printed using double data structure
- Truncating is used for printing hit ratio upto two decimal places
- Hit ratio = hit/(hit+miss),this formula is used to compute hit ratio

.

Optimization:

The number of searches,insertion and deletion had to be reduced to minimum to reduce the runtime of the program. Also the size of individual list needs to be controlled as it may go beyond the given size if not handled properly.