www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 3 Issue 6 June, 2014 Page No. 6260-6266

Novel Cache Replacement Algorithm

Namrata Dafre1, Deepak Kapgate2

1Student of M.Tech. (CSE), GHRAET, Nagpur,

Nagpur University(MS), India.

2Department of C.S.E., GHRAET, Nagpur,

Nagpur University (MS), India,

Abstract: In a computer architecture Cache memory have been introduced to balance performance and cost of the system. To improve the performance of a cache memory in terms of hit ratio and good response time system needs to employ efficient cache replacement policy. In this paper we proposed and implemented a novel cache replacement policy which predicts future request of a block depending upon its access pattern. Algorithm uses the concept of inter-reference recency of a block to detect access pattern and uses history of response time for efficient replacement. Result shows novel cache replacement policy has low response time and high hit ratio comparatively to other policy. Proposed policy concludes that it reduces response time and improves hit ratio.

Keywords: buffer cache, access patterns, cache replacement policies, buffer cache management techniques.

Introduction

To achieve better performance Cache memory uses concept of locality of reference. Use of cache memory is necessary because, it helps to reduce the time to move information to and from the processor. The concept is that at any given time the processor will be accessing memory in a small or localized region of memory, cache memory loads this region allowing the processor to access the memory region faster. When a new block is brought into the cache, one of the existing blocks must be replaced. For this purpose we need replacement policies. To achieve high speed, cache implements an efficient replacement policy. A good caching algorithm must have characteristics such as Low memory overhead, faster access to data, low response time to achieve a good hit rate.

In computer architecture Buffer cache is introduced to reduce the frequency of access made to the secondary storage devices and enhance the system throughput [2]. cache reduces read latency, while the buffer cache is to reduce writing operations. cache works between CPU and RAM while cache buffer works between RAM and external storage [1]. Different workloads and programs may have different accessing patterns like Sequential, looping-like, temporally-clustered, probabilistic references. To make cache more and more efficient replacement algorithms are used. They are classified into three types, replacement algorithms that incorporate longer reference histories than LRU[5] such as LRU-K[6], 2Q[13], LRFU[14], EELRU[12], MQ, LIRS[16], and ARC[15]. ACFS [28] and TIP [29] are replacement algorithms that rely on

application hints. Replacement algorithms that actively detects the I/O access patterns various levels such as block level, application level, file level and program context level.

SEQ detects long sequence of page cache misses at block-level and applies MRU policy to avoid scan pollution. DEAR observes the I/O pattern of each application at application-level and applies MRU, LRU and LFU according to the pattern detected. UBM at file-level detects access pattern of each individual file and uses MRU and LRU. AMP [21] and PCC [22] at program-context level, separates I/O stream into sub-stream and detects the pattern of each sub-stream.

Cache memory management has two aspects; they are use of a buffer cache management techniques and selection of an efficient replacement policy. The paper mainly focuses on implementation of Novel Cache Replacement algorithm which predicts future request of a block by knowing its access pattern. This research shows that how the proposed Novel Cache Replacement algorithm leads to minimization of reduction in response time.

The rest of the paper is organized as follows: Section II discusses about the existing cache replacement algorithms and cache management techniques. Section III, explains proposed algorithm and flow chart of proposed algorithm. Section IV shows the implementation details of algorithm and says about working environment. Section V shows the results calculated. Finally Section VI concludes the paper with future scope.

2. Literature survey

Least Recently Used algorithm is used widely because of its simplicity. It keeps track of the cache lines according to time they have been used and replaces page which have not been used for longer time.
But it has some limitations such as inability to cope with access patterns with weak locality and scan pollution. It causes thrashing for the workloads larger than cache size. To overcome limitations of LRU various policies have been introduced such as LRU-K[6], EELRU[12], FIFO[10], SC, Optimal replacement[9], LFU[5], 2Q[13], MRU[7], LRFU[14], Dueling CLOCK[19], LIRS [16], ARC [15].

At the same time to manage buffer cache various techniques such as block pre-fetching, prediction based on reuse distance and by detecting block access pattern has been introduced. In Block pre-fetching mechanism, data blocks are read prior and kept into main memory, to deal with the delay associated with the access made to the disk. User or compiler inserted hints are used in informed pre-fetching. I/O request are traced to obtain the information about the system call made by the applications and used in predictive caching. Automatic Pre-fetching And Caching System (APACS) is the example of block pre-fetching technique. In Distance based prediction mechanism, reuse distance of a block is used. Reuse distance of a block is the time difference between two consecutive references to a block. The reuse distance of a block can be obtained by use of a program counter. Re-Reference Interval Prediction (RRIP) technique has suggested Static RRIP (SRRIP) and Dynamic RRIP (DRRIP), Signature Based Hit Predictor comes under this technique. In block access pattern based detection mechanism, reference regularities are exploited to detect access pattern of a block. Unified Buffer Management (UBM)[23], Program-Counter based Classification (PCC)[22], DEtection based Adaptive Replacement (DEAR) [24] works by this technique.

Yifeng Zhu, et.al.[25] proposed a Robust Adaptive buffer Cache management scheme (RACE). In this scheme cache is partitioned by using marginal gain function to allocate blocks according to its access pattern. To know block access pattern it keeps track of references to a block by using file hash table and PC hash table. File hash table uses attributes such as inode, start and end block number, the last access made to the first block, looping period, last access to the referenced block, last accessed block and a PC hash table uses attributes such as fresh counter, reuse counter. After pattern detection block is

allocated into cache if free space is available or replacement is done with the existing replacement policy which is best suitable for the detected access pattern.

Reetu Gupta et.al. [26], Proposed Block Pattern Based Buffer Cache Management is a methodology which works above both program context level and file level. This scheme analyzes past access behavior and program context from I/O request and helps to predict block access patterns. It uses data structure which has values such as File hash table and PC hash table of a block. It has improved hit ratio by 10% to 15% over LRU.

In this paper the author is proposing a Novel Cache Replacement Algorithm which predicts future request of a block by accessing history information of a block.

3. Proposed Novel Cache Replacement Algorithm

The proposed predictive novel cache replacement algorithm uses information recency as well as inter-reference recency (IRR) of a block.

Algorithm first detects access pattern of a block and allocate it to the related cache partition. It keeps track of block parameters such as current accessed time, last accessed time, old IRR, new IRR, 'hitbit' has value zero if a block accessed first time or one if it is accessed repeatedly, 'numref' counts total number of references to a block till current time. From the IRR value algorithm comes to know access pattern of a block. Again a block maintains a variable 'p' which has value zero, one or two and it shows pattern of a block. If 0 for sequential, 1 for looping and 2 for other pattern.

According to block's 'p' value it is allocated to related cache partition. At the time of allocation three cases may occur, first is block gets allocated easily as a sequential type, second is block is already present in cache and need to move it from one partition to other as its pattern changes and third is need of replacement to allocate a block. Replacement can be done with the existing LRU policy or with proposed novel cache replacement policy. Proposed algorithm uses a parameter 'numref' which maintains total no. of references of a block till current time. The block having less no of accesses is a better candidate for replacement.

3.1 Flowchart :

The flowchart for Proposed Prediction Algorithm:

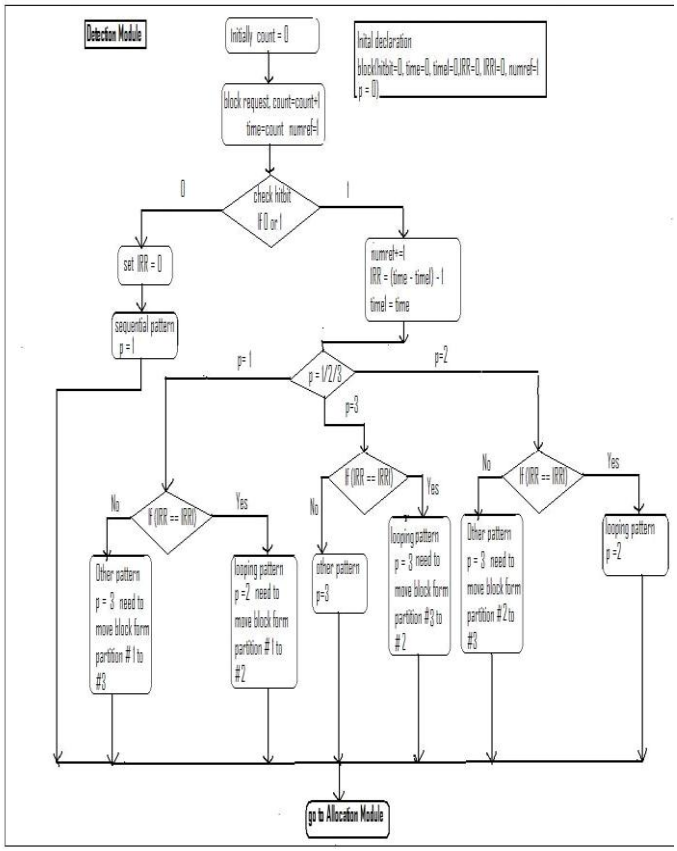

Figure 1 – Flowchart of Pattern detection module

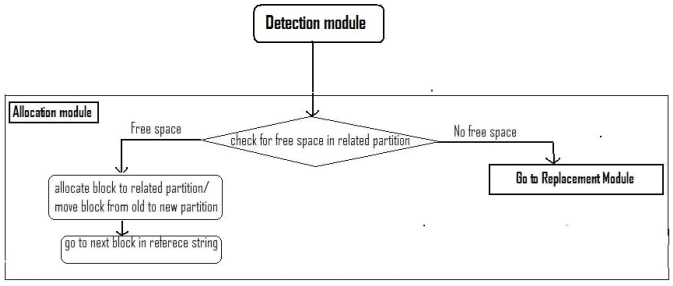

Figure 2 - Flowchart of allocation module

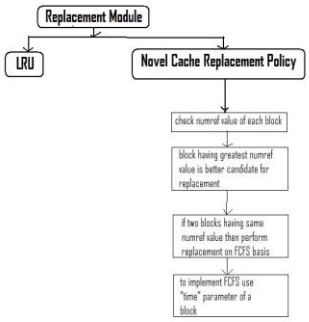

Figure 3 – Flowchart of Replacement module

3.2 Proposed Algorithm

Proposed algorithm works in three steps,

3.2.1 Detection of a block access pattern.

Cache is partitioned into three parts i.e. fixed no of blocks are dedicated to each partitions.

If a block is accessed first time (i.e. hitbit=0), then pattern is sequential and p=0. If a bloc is accessed repeatedly (i.e. hitbit=1), then pattern is looping (p=1) or other (p=2).

To detect whether it is looping or other, block access period is detected. If block's period is fixed then it is looping and if not then it is other.

3.2.2 Allocate block to its related cache partition

According to value of variable 'p' then cache partition is allocated. At the time of allocation three cases may occur.

First, block is detected as sequential, and allocated if space is free and replacement is done if cache is full.

Second, block is already present in cache then its looping or other pattern. Then block is moved to related cache partition according to the new access pattern. If space is not free then replacement is done.

3.2.3 Replacement

Replacement is done with the existing LRU policy and novel cache replacement policy.

4. Implementation Details

The proposed algorithm is implemented using Java Processor And Cache simulator.

The GUI has a menu bar having a options such as configuration, simulation. In configuration option we can set memory size, cache blocks, no of slots per cache block, sequential or parallel access, write back or write through option and the replacement policy selection. In simulation option we can perform it step-by-step or instantaneous and can save report too. Simulator's GUI shows four windows memory, cache memory, registers and simulation report. Memory window shows memory contents, cache memory window shows changes in cache memory, processor window shows contents of registers and flags and program counter, simulation report window shows step-by-step simulations showing no. of hits and miss. After complete simulation it shows result in terms of hit ratio, miss ratio, access time and performance gain.

5. Results Calculated

The Proposed algorithm is implemented using JPACS simulator. Some microprocessor instructions are compiled and then simulations are performed. Final results are shown below in a screen shot.

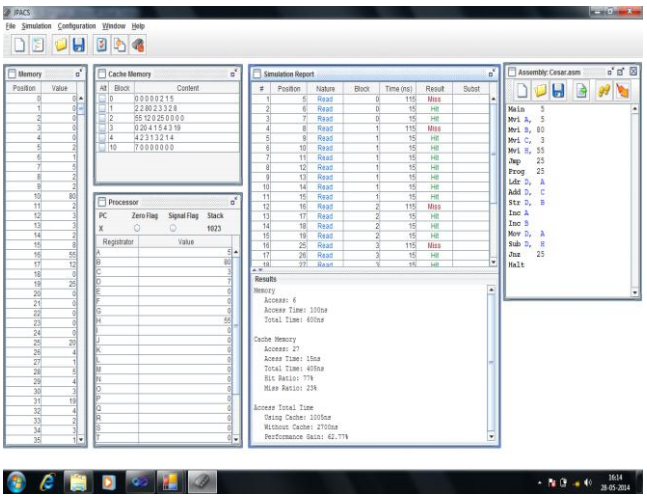

Figure 4 – Screenshot of JPACS simulator showing results

In above screen fig. 4 the all the four windows shows that instructions are compiled and simulation is performed and it shows the step-by-step changes occurred into memory, cache memory, and processor and simulation report. The result shows in terms of no of memory accesses, total access time, no. of cache memory accesses, total access time, hit ratio, miss ratio and performance gain.

Results are shown in the above stated terms are shown as below:

Results by using existing LRU replacement policy are shown in below fig. 5,

```
Memory
    Access: 6
    Access Time: 100ns
    Total Time: 600ns

Cache Memory
    Access: 27
    Acess Time: 15ns
    Total Time: 405ns
    Hit Ratio: 77%
    Miss Ratio: 23%

Access Total Time
    Using Cache: 1005ns
    Without Cache: 2700ns
    Performance Gain: 62.77%
```

Figure 5 – Result with LRU replacement results by using novel cache replacement policy are shown in below fig.6,

```
Memory
    Access: 57
    Access Time: 100ns
    Total Time: 5700ns

Cache Memory
    Access: 1216
    Acess Time: 15ns
    Total Time: 18240ns
    Hit Ratio: 98%
    Miss Ratio: 2%

Access Total Time
    Using Cache: 23940ns
    Without Cache: 121600ns
    Performance Gain: 80.31%
```

Figure 6 – Result with NCRP policy

6. Conclusion

By observing the results, novel cache replacement policy minimizes the response time and improves cache memory hit ratio which leads to improve system performance. The proposed policy is made by combining inter-reference recency and frequency based history information of a block. Result shows that proposed algorithm performs better than existing least recently used policy in some cases in which LRU fails.

References

[1]Hou Fang, zhao Yue-long, Hou fang, "A Cache Management Algorithm Based on Page Miss Cost", in proc of International conference on Information Engineering and computer science, ICIECS, ISBN: 978-1-4244-4994-1 pp. 1-4, 2009.

[2] Prof.P.K. Biswas, "Lecture Series on Digital Computer Organization," Internet: http://nptel.iitm.ac.in, Sep 2009 [Aug 12, 2012].

[3] M. J. Bach, "Operating System The design of the UNIX", Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[4] A. S. Tanenbaum, A. S.Woodhull, "Operating Systems Design and Implementation", Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987.

[5] Donghee Lee, et.al.,."On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies",SIGMETRICS'99 Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages 134-143, NY, USA, 1999.

[6] E. J. O'Neil, P. E. O'Neil, and G. Weikum., "The LRU-K Page Replacement Algorithm for Database Disk Buffering" , ACM Conference on SIGMOD, pg. no. 297–306, 1993.

[7] K. So and R. N. Rechtschaffen, "Cache operations by MRU change." IEEE Trans. Computers, vol. 37, no. 6, pp. 700–709,1988.

[8] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," IBM Syst. J., vol. 5, no. 2, pp. 78-101, 1966.

[9] Alfred V. Aho, Jeffrey D. Ullman, et.al., "Principles of Optimal Page Replacement", Journal of ACM, Vol 18, Issue 1, Pages 80-93,Jan 1971.

[10] R. Turner and H. Levy, "Segmented FIFO Page Replacement", In Proceedings of SIGMETRICS ,1981.

[11] A. Dan and D. Towsley, "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes", in Proceedings of ACM SIGMETRICS, Boulder, Colorado, United States, pp. 143—152,1990.

[12] Y. Smaragdakis, S. Kaplan, and P. Wilson, "EELRU: simple and effective adaptive page replacement," in Proceedings of ACM SIGMETRICS international conference on Measurement and modeling of computer systems, New York, USA, pg. no. 122–133, 1999.

[13] T. Johnson and D. Shasha, " 2Q : A Low Overhead High Performance Buffer Management Replacement Algorithm" , In Proceedings of the 20th International Conference on VLDB, pg. no. 439–450, 1994.

[14] D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, and C. Kim, "LRFU: A Spectrum of Policies that Subsumes the LRU and LFU Policies", IEEE Transactions on Computers, vol. 50, Issue no. 12, pp.1352-1361, Dec. 2001

[15] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST), pg no. 115–130, Mar 2003.

[16] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," in Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pg. no. 31–42, June 2002.

[17] Seon-yeong Park, et.al., "CFLRU: A Replacement Algorithm for Flash Memory", CASES'06, October 23–25, Seoul, Korea ,2006.

[18] LI Zhan-sheng, et.al.,"CRFP: A Novel Adaptive Replacement Policy Combined the LRU and LFU", IEEE 8th International Conference on Computer and Information Technology Workshops, 2008.

[19] Andhi Janapsatya, Aleksandar Ignjatovi´c, et.al., "Dueling CLOCK: Adaptive Cache Replacement Policy Based on The CLOCK Algorithm", 2010.

[20] T. Puzak, et.al, "Analysis of cache replacement algorithms," Ph.D. dissertation, Dep. Elec. Comput. Eng., Univ. Massachusetts, Feb.1985.

[21] F. Zhou, R. von Behren, and E. Brewer, "AMP: Program context specific buffer caching," in Proceedings of the USENIX Technical Conference, Apr. 2005

[22] C. Gniady, A. R. Butt, and Y. C. Hu, "Program-counter based pattern classification in buffer caching", in Proceedings of 6th Symposium on Operating System Design and Implementation ,pg. no. 395–408, Dec. 2004.

[23] J. M. Kim, J. Choi, J. Kim, S. H. Noh, S. L.Min, Y. Cho, and C. S. Kim,"A low-overhead, high-performance unified buffer management scheme that exploits sequential and looping references," in 4th Symposium on Operating System Design and Implementation ,pg. no. 119–134, Oct. 2000

[24] Jongmoo Choiy, Sam H. Nohz, Sang LyulMiny, Yookun Cho, "An Adaptive Block Management Scheme Using On-Line Detection of Block Reference Patterns", International Workshop on , pg no. 172 – 179, Dayton, Aug 1998.

[25] Yifeng Zhu, Hong Jiang ,"RACE: A Robust Adaptive Caching Strategy for Buffer Cache", IEEE Transaction on computers, 2007.

[26]Urmila Shrawankar, Reetu Gupta, "Block Pattern Based Buffer Cache Management", The 8th International Conference on Computer Science & Education, April 26-28, Colombo, 2013.

[27]Reetu Gupta, Urmila Shrawankar, "Managing Buffer Cache by Block Access Pattern", IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 6, No 2, November 2012.

[28] P. Cao, E. W. Felten, et.al,"Implementation and performance of integrated application-controlled file caching, prefetching, and disk scheduling," ACM Transactions on Computer Systems, vol. 14, Issue 4, pp. 311–343, 1996.

[29] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed prefetching and caching," in Proceedings of the fifteenth ACM symposium on Operating systems principles (SOSP), New York, USA: ACM Press, 1995.

[30]A. Jaleel, C. Jean, S. C. Steely, "ShiP: Signature Based Hit Predictor for High Performance Caching", ACM International symposium on Computer Architectur, pg 430-431,2011.

[31] Zhiyang Ding,et.al., "An Automatic Prefetching and Caching System", IEEE, 2010.

[32] Heung Seok Jeon, "Practical Buffer Cache Management Scheme based on Simple Prefetching", IEEE Transactions on Consumer Electronics, Vol. on - 52, Issue No. 3, August 2006.

[33] G. Keramidas, P. Petoumenou, S. kaxiras, "Cache Replacement Based on Reuse Distance Prediction", Computer Design IEEE International Conference, Pg no. 245-250, Oct 2007.

[34] G. Keramidas, P. Petoumenou, S. kaxiras, " Instruction Based Reuse Distance Prediction for Efficient Cache Management", Pet International Symposium on System, Architecture, Modeling and Simulations, pg no. 48-49, July 2009.