Comparison of Cache Page Replacement Techniques to Enhance Cache Memory Performance

Pancham
M.Tech, Student
IET Alwar, Rajsthan, India

Deepak Chaudhary
Assistant Professor
IET Alwar, Rajsthan, India

Ruchin Gupta
Assistant Professor
AKGEC, Ghaziabad, India

ABSTRACT
Caching is a very important technique for improving the computer system performance, it employed to hide the latency gap between memory and the CPU by exploiting locality in memory accesses. In modern computer architectures a page miss cause the several hundred CPU cycles [1, 15]. In a memory hierarchy, a cache performs faster than auxiliary storage, but is more expensive. Some important page replacement policies such as FIFO, LRU, RANDOM and SECOND CHANCE are used in cache memory to replace the page whenever it is needed. So page replacement policy severely affects the performance of cache memory. So here our purpose is to simulate FIFO, LRU, RANDOM and SECOND CHANCE policies and to compare the results for various applications such as bzip, swim and gcc traces (taken from SPEC2000 benchmark for simulation) etc. using MS-Excel [3, 10, 12] .

General terms
Computer Architecture, Operating system, cache memory, main memory and Performance evaluation of system

Keywords
Cache performance, page replacement policies, page faults, MS-Excel, C-language

1. INTRODUCTION
In modern computer Architecture Page replacement is an important part of an operating system. When we needed a page, it is searched in translation look aside buffers (TLB) or page tables and if found missing from main memory a page fault will be occur. The size of the cache memory smaller than auxiliary memory [4, 5]. The role of the page replacement is to identify the page to evict from main memory when page fault occur and replace it by new page from disk than contains requested datum or instruction. The problem is very common to the block replacement in cache memories but the page replacement is more critical as page transfers from disk to memory with respect magnitudes slower than block transfers from main memory to cache memory. Many page replacement policies are derived and tested. Some of them include Fist-in-First-out (FIFO), least recently used (LRU), RANDOM and SECOND CHANCE. Good page placement policies reduced the page faults cost resulting in higher performance. Since the more page faults the operating system encounters the more resources are wasted in paging in/out instead of doing useful work. And resulting it causes the serious thrashing problems. In this paper, we present the relative competitive analysis of the following page replacement policies including FIFO, LRU, RANDOM and SECOND CHANCE. The relative competitive performances of the page replacement policy are relative to the performance of another policy [14, 19].

2. REPLACEMENT ALGORITHMS
Let us briefly explain the basic definition of the replacement policies for the preparation of this paper [12, 19]:

2.1 First in, First out (FIFO)
In first-in-first out page replacement policy, when a page is needed,
the page that has been in memory for the longest period of time is chosen to replace. The rationale is that a page that has only recently been swapped in will have a higher probability of being used again soon. However a frequently used page still gets swapped out when it gets to be old enough, even though it will have to be brought in again immediately [8, 19].

2.2 Least Recently Used (LRU)
Least recently used page replacement policy is based on past aspect is a mirror of the pattern in the near future. Pages that have been accessed recently are likely to continue to be accessed and ought to be kept in physical memory. An allocated memory page of a program will become a replacement candidate if the page has not been accessed for a certain period of time under two conditions: (1) the program does not need to access the page; and (2) the program is conducting page faults (a sleeping process) so that it is not able to access the page although it might have done so without the page faults. However, LRU page re- placement implementations do not discriminate between two types of LRU pages and treat them equally. So it means that LRU can be made closer to optimal policy by making improvement into that [9, 19].

2.3 Random Replacement (RR)
It randomly selects the particular page and discards it to make space when necessary. This algorithm does not require any information to access history. For its simplicity it is used for ARM processors. Random replacement policy randomly replaces the page in memory when it needed. This eliminates the overhead cost to tracking the page references. Usually it is better than FIFO policy and for looking memory reference it is better than LRU but generally LRU perform better result in practice. Generally OS/390 uses LRU replacement but when LRU performance degenerate it fall back to Random replacement. Intel i860 processor used a random replacement policy [11, 19].

2.4 Second Chance Replacement (SR)
In Second Chance page replacement policy, the pages for removal are consider in a round robin manner, and a page that has been accessed between consecutive considerations will not

be replaced. The page replaced is the one that considered in a round robin matter has not been accessed since its last consideration[16, 17].

Second Chance Implementation algorithm:

1. Add a "second chance" bit to each memory frame.
2. Each time a memory frame is referenced, set the "second chance" bit to ONE (1) - this will give the frame a second chance.
3. A new page read into a memory frame has the second chance bit set to ZERO (0)
4. When you need to find a page for removal, look in a round robin manner in the memory frames:
   - If the second chance bit is ONE, reset its second chance bit (to ZERO) and continue.
   - If the second chance bit is ZERO, replace the page in that memory frame.

3. SIMULATOR

The proper choice of a page replacement algorithm is actually quite a complex matter. To make the proper choice, we must know something about real applications. How do they really access memory? Do they generate many page accesses in order? To answer these questions, we must see what real applications do. In this, paper evaluates how real applications respond to a variety of page replacement algorithms. Modifying a real operating system to use different page replacement algorithms is quite a technical mess, so it will make this by simulation. We write a program that simulates the behaviour of a memory system using a variety of page replacement algorithms. We obtain memory traces from real applications so that we can evaluate algorithm properly. Here the purpose is to build a simulator that reads a memory trace and simulates the action of a cache memory with a single level page table in single programming model. The simulator keeps track of what pages are loaded into memory [6]. As it processes each memory event from the trace, it should check to see if the corresponding page is loaded. If not, it should choose a page to remove from memory. Assume that all pages and page frames are 4 KB etc. It implements different page replacement algorithms such as FIFO, LRU RANDOM and SECOND CHANCE. The simulator is written in plain C in MS-DOS environment It assumes that reference string is containing six thousands references stored in an array. Numbers of frames are varied and no of page faults are calculated. Reference strings of different applications are taken as input and numbers of page faults are calculated and graphs are plotted in MS-EXCEL between no. of frames vs. no. of page faults [2, 3, 18].

3.1 Memory Traces

Each trace obtained from Internet is a real recording of a running program, taken from the SPEC2000 benchmarks. Real traces are enormously big having billions and billions of memory accesses. However, a relatively small trace will be more than enough. Each trace only consists of one million memory accesses taken from the beginning of each program. Traces are gcc.trace.gz, swim.trace.gz , bzip.trace.gz Each trace is a series of lines, each listing hexadecimal memory addresses followed by R or W to indicate a read or a write.
For example, gcc.trace trace starts like this:
0041f7a0R 13f5e2c0R 05e78900R 004758a0R

4. SIMULATION RESULTS

We executed several programs for 9 different no. of frame size and compared it with DIRECT, FIFO, LRU, RANDOM and SECOND CHANCE page replacement policies. As we compared the page faults for above policies in table 10, table 11, and table12 for standard traces bzip, swim and gcc the performance of the policies are varied. We assume that all pages, frames size are 3KB and 1KB etc. and reference string of size 3 thousand [7,13].

Table 1 (for Direct mapping)

No. of frames	Page faults for bzip.txt	Page faults for swim.txt	Page faults for gcc.txt
3	811	1404	1931
7	696	1113	1519
15	580	849	1302
31	465	480	1141
63	393	367	993
127	331	340	878
255	300	219	787
511	282	218	716
1023	265	179	672

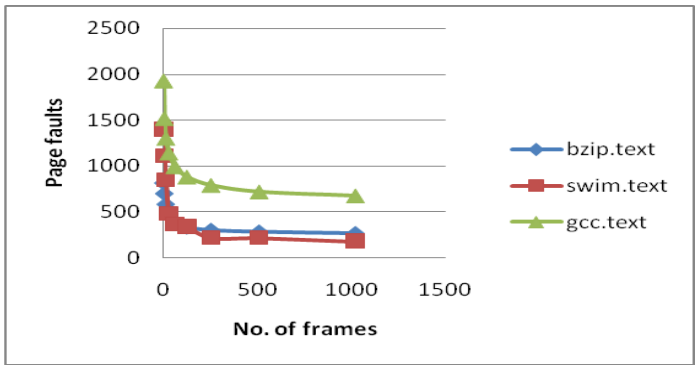


Figure 1 (for Direct mapping)

Table 2 (for FIFO associative mapping)

No. of frames	Page faults for bzip.txt	Page faults for swim.txt	Page faults for gcc.txt
3	719	1397	1826
7	579	1131	1415
15	472	832	1121
31	378	319	928
63	329	125	858
127	278	82	828
255	244	82	803
511	244	82	575
1023	244	82	571

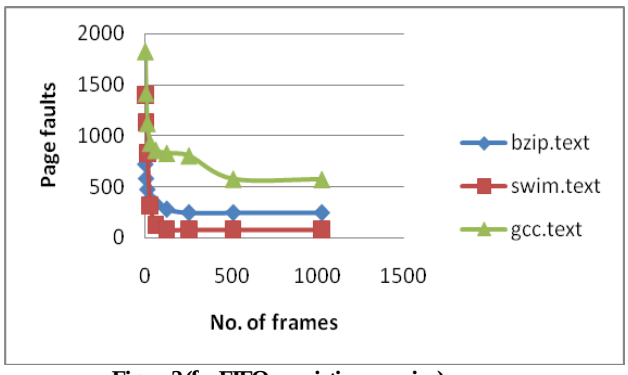


Figure 2 (for FIFO associative mapping)

Table 3 (for FIFO Set-associative mapping)

No. of frames	Page faults for bzip.txt	Page faults for swim.txt	Page faults for gcc.txt
3	719	1397	1826
7	600	1142	1428
15	483	765	1145
31	402	355	977
63	352	221	905
127	292	132	838
255	259	98	756
511	248	86	636
1023	244	82	585

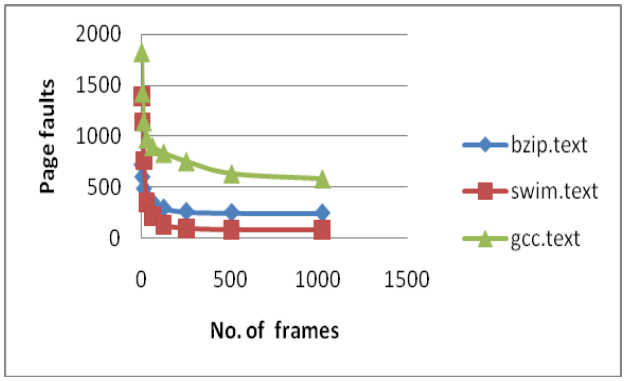


Figure 3 (for FIFO Set-associative mapping)

Table 4 (for LRU associative mapping)

No. of frames	Page faults for bzip.txt	Page faults for swim.txt	Page faults for gcc.txt
3	682	1143	1737
7	549	1009	1272
15	447	687	1007
31	368	173	886
63	313	95	849
127	275	82	819
255	244	82	796
511	244	82	573
1023	244	82	571

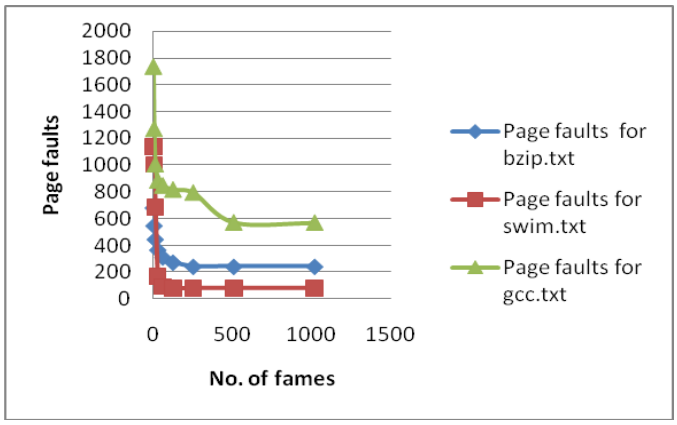


Figure 4 (for LRU associative mapping)

No. of frames	Page faults for bzip.txt	Page faults for swim.txt	Page faults for gcc.txt
3	682	1143	1737
7	571	995	1310
15	457	660	1054
31	378	263	938
63	335	179	888
127	287	117	830
255	257	91	752
511	247	84	630
1023	244	82	583

Table 5 (for LRU Set-associative mapping)

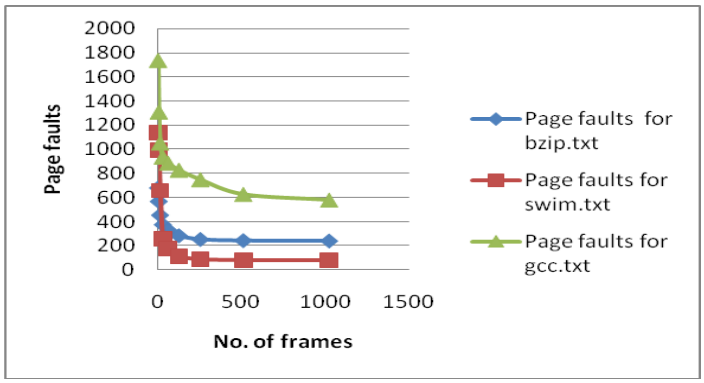


Figure 5 (for LRU Set-associative mapping)

Table 6 (for RANDOM associative mapping)

No. of frames	Page faults for bzip.txt	Page faults for swim.txt	Page faults for gcc.txt
3	760	1425	1846
7	620	1127	1475
15	545	713	1278
31	449	304	1140
63	347	118	961
127	274	82	836
255	244	82	731
511	244	82	574
1023	244	82	571

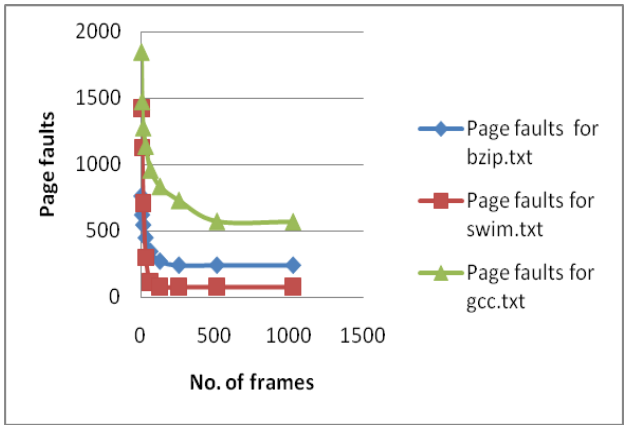


Figure 6 (for RANDOM associative mapping)

Table 7 (for RANDOM Set-associative mapping)

No. of frames	Page faults for bzip.txt	Page faults for swim.txt	Page faults for gcc.txt
3	760	1425	1846
7	645	1116	1478
15	497	703	1216
31	438	338	1041
63	353	189	977
127	302	153	870
255	257	93	747
511	246	86	631
1023	244	82	578

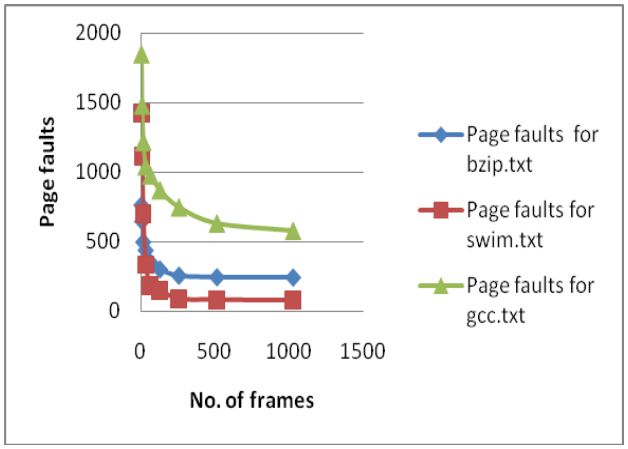


Figure 7 (for RANDOM Set-associative mapping)

Table 8 (for SECOND CHANCE associative mapping)

No. of frames	Page faults for bzip.txt	Page faults for swim.txt	Page faults for gcc.txt
3	688	1155	1699
7	558	979	1253
15	446	617	995
31	367	168	893
63	317	91	849
127	281	82	820
255	244	82	693
511	244	82	574
1023	244	82	571

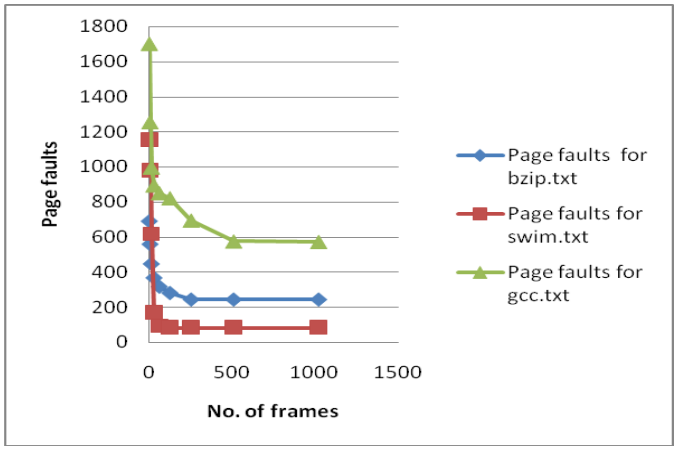


Figure 8 (for SECOND CHANCE associative mapping)

Table 9 (for SECOND CHANCE Set- associative mapping)

No. of frames	Page faults for bzip.txt	Page faults for swim.txt	Page faults for gcc.txt
3	688	1155	1699
7	568	979	1324
15	473	586	1095
31	399	277	968
63	357	206	903
127	312	121	852
255	278	101	761
511	259	100	686
1023	256	93	629

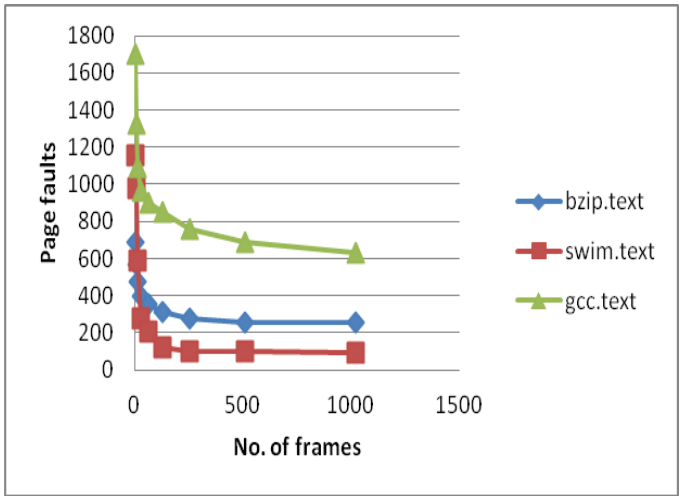


Figure 9 (for SECOND CHANCE Set-associative mapping)

Table10 (Using bzip application)

No. of frames	DIRECT	No. of Page faults using FIFO Associative	No. of Page faults using FIFO Set Associative	No. of Page faults using LRU Associative	No. of Page faults using LRU Set Associative	No. of Page faults using Random Associative	No. of Page faults using Random Set Associative	No. of Page faults using Second Chance Associative	No. of Page faults using Second Chance Set Associative
3	811	719	719	682	682	760	760	688	688
7	696	579	600	549	571	620	645	558	568
15	580	472	483	447	457	545	497	446	473
31	465	378	402	368	378	449	438	367	399
63	393	329	352	313	335	347	353	317	357
127	331	278	292	275	287	274	302	281	312
255	300	244	259	244	257	244	257	244	278
511	282	244	248	244	247	244	246	244	259
1023	265	244	244	244	244	244	244	244	256

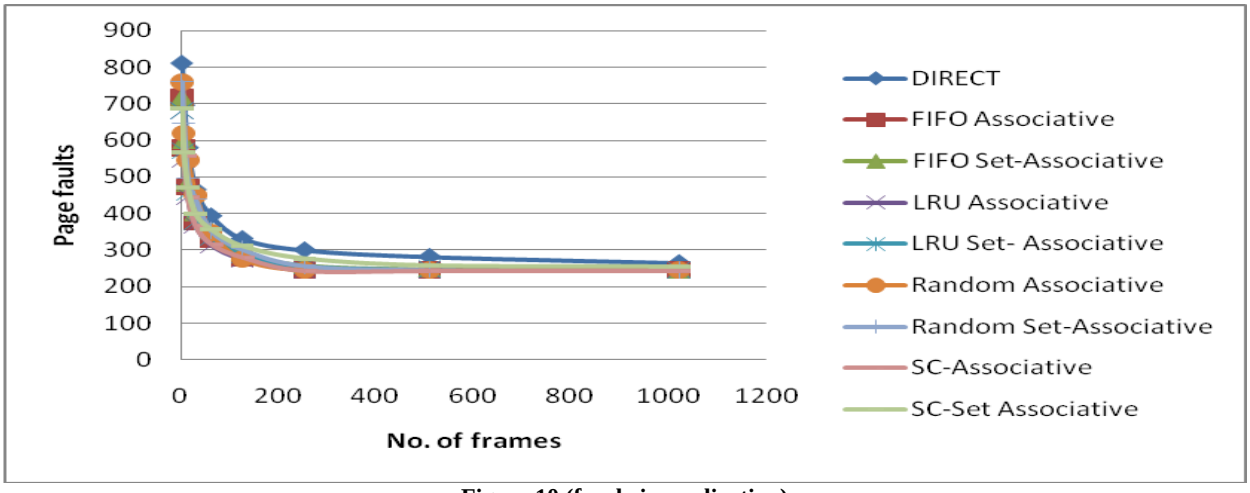


Figure 10 (for bzip application)

Table11 (Using swim application)

No. of frames	DIRECT	No. of Page faults using FIFO Associative	No. of Page faults using FIFO Set Associative	No. of Page faults using LRU Associative	No. of Page faults using LRU Set Associative	No. of Page faults using Random Associative	No. of Page faults using Random Set Associative	No. of Page faults using Second Chance Associative	No. of Page faults using Second Chance Set Associative
3	1404	1397	1397	1143	1143	1425	1425	1155	1155
7	1113	1131	1142	1009	995	1127	1116	979	979
15	849	832	765	687	660	713	703	617	586
31	480	319	355	173	263	304	338	168	277
63	367	125	221	95	179	118	189	91	206
127	340	82	132	82	117	82	153	82	121
255	219	82	98	82	91	82	93	82	101
511	218	82	86	82	84	82	86	82	100
1023	179	82	82	82	82	82	82	82	93

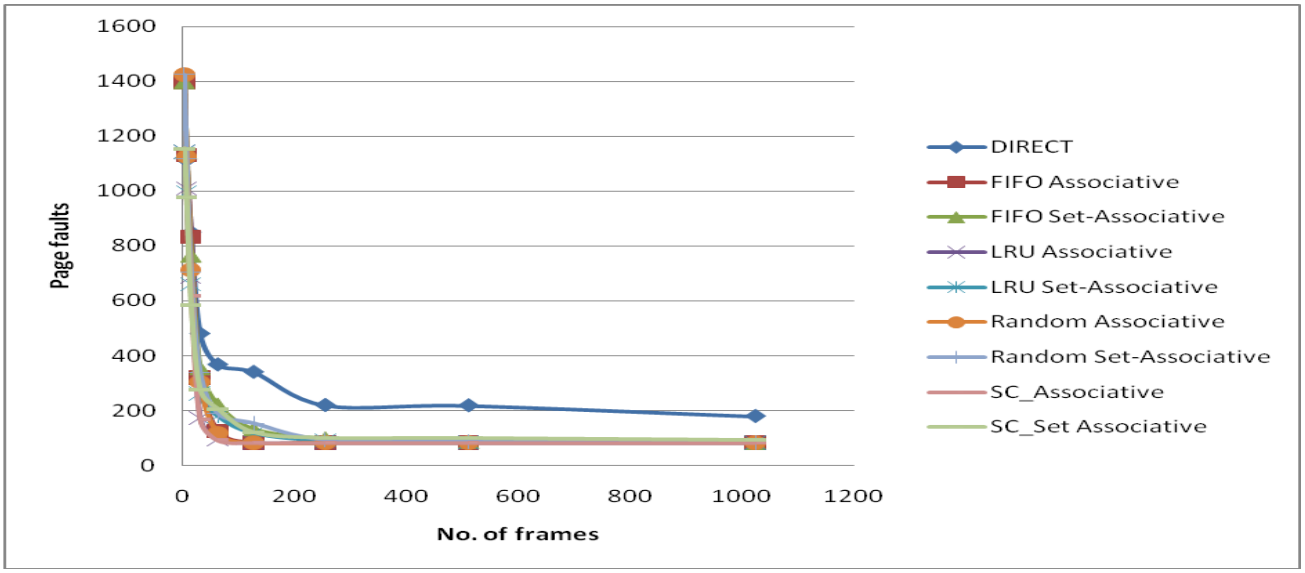
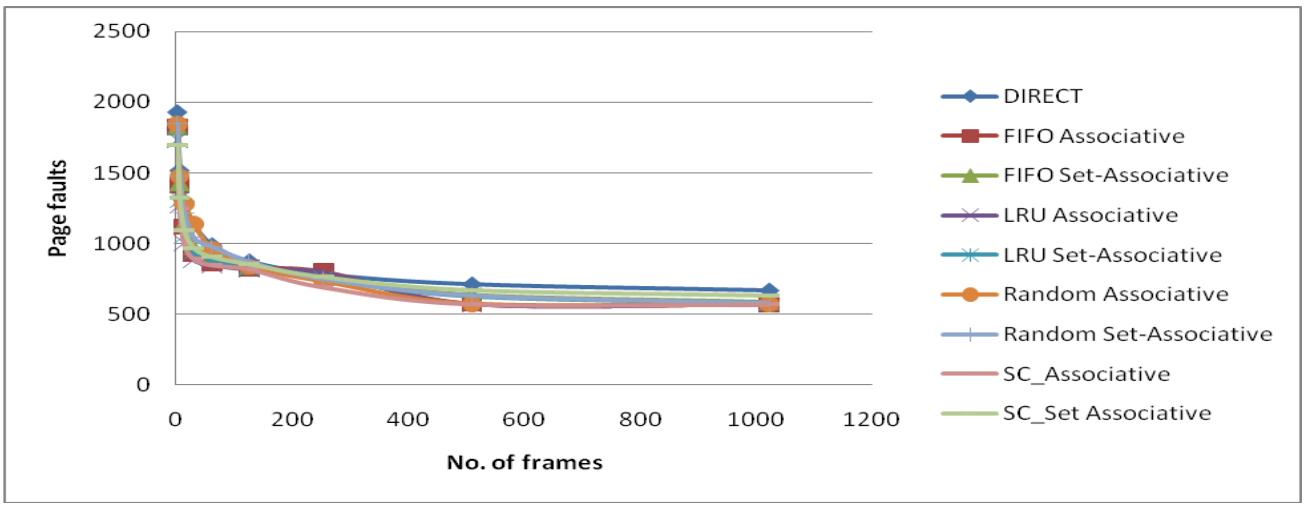


Figure 11 (for swim application)

Table 12 (Using gcc application)

No. of frames	DIRECT	No. of Page faults using FIFO Associative	No. of Page faults using FIFO Set Associative	No. of Page faults using LRU Associative	No. of Page faults using LRU Set Associative	No. of Page faults using Random Associative	No. of Page faults using Random Set Associative	No. of Page faults using Second Chance Associative	No. of Page faults using Second Chance Set Associative
3	1931	1826	1826	1737	1737	1846	1846	1699	1699
7	1519	1415	1428	1272	1310	1475	1478	1253	1324
15	1302	1121	1145	1007	1054	1278	1216	995	1095
31	1141	928	977	886	938	1140	1041	893	968
63	993	858	905	849	888	961	977	849	903
127	878	828	838	819	830	836	870	820	852
255	787	803	756	796	752	731	747	693	761
511	716	575	636	573	630	574	631	574	668
1023	672	571	585	571	583	571	578	571	629



5. FUTURE SCOPE

- Simulation can be used for multiprogramming model of operating system.
- Simulation study can be developed to consider complete traces to produce more realistic results.
- Large size of main memory and cache memory can be considered in simulation study.
- This kind of simulation study can be used to compare latest page replacement techniques in cache memory in future.
- Further simulation study can be used to include multiple cache memory levels and execution environment.
- Simulation study proposed in the report provides useful conclusions for the design of future replacement algorithm

6. CONCLUSION

It is observed that traces bzip, swim and gcc exhibits different memory access pattern that's why producing different number of page replacement. In all the cache architectures the gcc produces the largest number of page replacement perhaps due to irregular memory access pattern while bzip produces minimum number of page replacement, initially but swim trace gives a rapid decrease in the number of page replacement with increase in number of frames. It is observed that LRU page replacement in associative and set associative gives better performance than other policies in other cache architectures for all the memory traces that's why it can be concluded that SECOND CHANCE associative and LRU associative performance is better than FIFO and RANDOM policies. It is also observed that the performance of SECOND CHANCE associative in cache model is slightly better than LRU in associative cache model.

7. ACKNOWLEDGEMENTS

I am thankful to Mr. Ruchin Gupta and my guide Mr. Deepak Chaudhary for their support and cooperation.

8. REFERENCES

[1]. Jan Reineke Daniel Grund "Relative Competitive Analysis of Cache Replacement Policies" LCTES'08, June 12–13, 2008, Tucson, Arizona, USA. Copyrightc 2008 ACM.

[2]. S. Jiang, and X. Zhang, "LIRS: An Efficient Policy to improve Buffer Cache Performance",IEEE Transcations on Computers, pp. 939-952, 2005.

[3] S. Albers, S. Arora, and S. Khanna, "Page replacement for general caching problems," *Proceedings of the* 10*th Annual ACM–SIAM Symposium on Discrete Algorithms*, pp. 31–40, 1999.

[4]. Hameed, F., L. Bauer and J. Henkel, 2013. "Dynamic cache management in multi-core architectures through run-time adaptation. Proceedings of the Design, Automation and Test" in Europe Conference and Exhibition, Mar. 12-16, IEEE Xplore Press, Dresden, pp: 485-490. DOI: 10.1109/DATE.2012.6176518

[5]. S.Irani, "Page Replacement with Multi-Size Pages and Applications to Web Caching," *Proc.29th Ann, ACM symp*. *Theory of Computing*, pp. 701-710, 1997. [6] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "an Optimality Proof of the LRU-K page Replacement Algorithm." *J.ACM*, vol. 46, no.1, pp. 92-112, 1999.

[6].Debabala Swain, Bijay K Paikray, Debabrata Swain,"AWRP: Adaptive Weight Ranking Policy for Improving Cache Performance", Journal of Computing, vol-3, Issue-2, February 2011.

[7]. Kaveh Samiee and GholamAli Rezai Rad, "WRP: Weighting Replacement Policy to Improve Cache Performance," International Symposium on Computer Science and its Application, IEEE, 2008.

[8].Yogesh Niranjan, Shailendra Tiwari "Design and Implementation of Page Replacement Algorithm for Web Proxy Caching", Int.J.Computer Technology & Applications,Vol 4 (2),221-225 IJCTA | Mar-Apr 2013.

[9].C. Aggarwal, J. L. Wolf, and P. S. Yu. "Caching on the WorldWideWeb," In IEEE Transactions on Knowledge and Data Engineering, vol. 11, pp. 94-107, 1999.

[10].Nimrod Megiddo, Dharmendra, S. Modha IBM Almaden Research Center Outperforming LRU with an Adaptive Replacement Cache Algorithm, Published by the IEEE Computer Society, 0018-9162/04/$20.00 © 2004 IEEE.

[11]. S.M. Shamsheer Daula, Dr. K.E Sreenivasa Murthy, G Amjad Khan "A Throughput Analysis on Page Replacement Algorithms in Cache Memory Management", International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 Vol. 2, Issue 2, pp.126-130.

[12].John Dilley, Martine Arlitt and Stephane Perret "Enhancement and Validation of Squid's Cache Replacement Policy" Internet Systems and Applications Laboratory HP Laboratories Palo Alto HPL- 1999-69, May 2009.

[13].Michael Factor, Assaf Schuster, Gala Yadgar, "Multilevel Cache Management Based on Application Hints" Technion- Computer Science Department Technical Report CS-2006.

[14]. Amit S. Chavan, Kartik R. Nayak, Keval D. Vora, Manish D. Purohit and Pramila M. Chawan "A Comparison of Page Replacement Algorithms" *IACSIT Vol.3, No.2, April 201*.

[15]. A. S. Tanenbaum and A. S. Woodhull, *Operating Systems: Design and Implementation*. Prentice-Hall, 1997.

[16] Vinit A. Kakde, Sanjay K. Mishra, " Effective Web Cache Algorithm," International Journal of Electronics, Communication & Soft Computing Science and Engineering (IJECSCSE) Volume 1, Issue 1.

[17]http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.cortexr/index.html

[18] Development of a Virtual Memory Simulator to Analyze the Goodness of Page Replacement Algorithms Fadi N. , Sibai, Maria Ma, David A. Lill

[19]. Debabrata Swain, Bancha Nidhi Dash " Analysis and Predictability of Page Replacement Techniques towards Optimized Performance"IRCTITC 2011 Proceedings published in International Journal of Computer Applications® (IJCA).