Abstract
Improving miss penalty and cache replacement policy has been a hotspot for researches. The miss penalty may be defined as the number of cycles microprocessor is stalled for a memory access is called miss penalty and it is sum of
- Cycles to replace a block in the cache.
- Cycles to deliver the block to the processor.
The miss penalty problem reduces the overall efficiency of the processing environment. Several efforts have been made to improve the performance of the cache by reducing the miss penalty rate. In this paper, we discuss the work done by different researchers to reduce the miss penalty problem and propose a new idea that can be combined with the existing techniques to improve the performance and reduce the miss penalty problem.

Keywords
Miss Penalty, Cache, Microprocessor, Replacement Policy, Miss Penalty Rate

I. Introduction
In today’s computers, we have very fast microprocessors, but main memory, i.e. RAM is not as fast as compared to a microprocessor. This speed gap creates problems and the fastness of microprocessors is affected negatively. One way to overcome this problem is to insert a small high-speed buffer memory between the processor and the main memory. Such a buffer is generally referred to as cache memory or only cache for short. The cache interconnection [9] with RAM has been shown in the figure.

Fig. 1: Cache Interconnection

The microprocessor can take advantage of this enhancement by fetching data from the cache instead of main memory. Caching is one of the most popular and effective methods to eliminate the speed gap. When microprocessor access an element from cache and the element is not found on cache, a cache miss occurs, known as miss penalty. When a cache miss occurs then miss penalty first needs to take a block from a specific set. Suppose that the evicted block in the cache which is going to be replaced is dirty block, i.e. it is modified in the cache but not copied inside the memory.

Now, what normally happen is that the cache controller first copies the cache block in the memory in order to make a room for new block inside that specific set and after this the cache controller then moves the memory block inside that cache line. So, all replacement policies first transfer the evicted block to memory and then a block from memory is transferred to that cache line. This miss penalty problem reduces the overall efficiency in terms of speed of the processing environment. So far, a number of techniques have been developed to improve the performance of the cache by reducing the miss penalty rate.

II. Different Techniques For Reducing Cache Miss Penalty
In order to reduce the problem of miss penalty in cache memory, different techniques have been developed by researchers. Here we discuss important techniques developed for reducing miss penalty problem in cache so far.

A. Sequential Hardware Prefetching in Shared-Memory Multiprocessors
Sequential prefetching [1] took advantage of spatial locality to reduce the read stall time. Spatial locality, however, varies among different programs and also during different parts of the execution of a given program. This calls for a technique where the degree of prefetching is high at those moments when it is useful, and is small or even zero at those moments when it does not pay off. The proposed adaptive prefetching – another contribution of this paper – tunes the degree of prefetching based on a dynamic measure of prefetching effectiveness. The simulations show a considerable reduction of the number of read misses and of the read stall time for those applications where the spatial locality of read misses is high, and show little added traffic for those applications where the spatial locality of misses is small. The number of read misses and thus the read stall time was significantly reduced for cases where the spatial locality of read misses is small or even zero at those moments when it does not pay off.

B. Miss Penalty Reduction Using Bundled Capacity Prefetching in Multiprocessors
This Bundled Capacity Prefetching in multiprocessor [2] method is to simply categorize cache lines in communicating or non-communicating. The optimization proposed is to issue prefetch requests for non-communicating lines. No optimization has been carried out for communicating lines. The strategy reduces the capacity and cold misses by sequentially prefetching lines without causing problems with false sharing. The strategies create virtually longer cache lines for all non-communicating data and keep the cache lines short for communicating data. The hardware implementation cost of the prefetching scheme is very low. By combining the messages generated at each prefetch for all Read and Upgrade requests, the address traffic can be significantly lowered using bundled capacity prefetching.
C. Cache Conscious Data Layout Organization for Conflict Miss Reduction in Embedded Multimedia Applications

This technique has presented a new data layout organization technique [3] which is able to reduce conflict misses to a large extent. This technique has been automated as part of a source-to-source precompiled for multimedia applications, called ACROPOLIS. The results indicate a consistent reduction in the number of conflict misses for different drivers with different cache sizes and a performance-power trade off can be obtained for a given application.

D. Generational Cache Management of Code Traces in Dynamic Optimization Systems

This technique [4] showed that, large, interactive applications impose limiting constraints on a code cache management system. The salient features of such applications complicate management heuristics and strain the system due to the frequency and number of code traces generated at runtime. The technique explored generational code caches as a means for solving the problem of code cache management in dynamic optimizers. The motivation was based on an analysis of the lifetime of code traces residing in the code cache for various applications, and the observation that the majority of code traces were either very short- or very long-lived. It was found that, by replacing a single cache with multiple generational caches, we can decrease the miss rates and resulting overhead of nearly all benchmarks, large or small, often significantly.

E. Hiding Cache Miss Penalty Using Priority-based Execution for Embedded Processor

This technique [5] uses the latency time. The contribution of memory latency to execution time continues to increase, and latency hiding mechanisms become ever more important for efficient processor design. While high-end processors can use elaborate techniques like multiple issue, out-of-order execution, speculative execution, value pre-diction etc. to tolerate high memory latencies, they are often not viable solutions for embedded processors, due to significant area, power and chip complexity overheads. This technique proposes a hardware-software cooperative approach, called priority-based execution to hide cache miss penalty for embedded processors [5]. The compiler classifies the instructions into low-priority and high-priority instructions. The processor executes the high-priority instructions, but delays the execution of low-priority instructions. They are executed on a cache miss to hide the cache miss penalty. The technique was empirically evaluated on the Intel XScale compiler and micro architecture. Experimental results on bench-marks from Multimedia, MediaBench, MiBench, and SPEC2000 demonstrate average 17% performance improvements, hiding 75% cache miss penalty.

F. Bargain Cache: Using File-System Metadata to Reduce the Cache Miss Penalty

This technique [7] introduced a simple observation that non-sequential access of disk data is much slower than sequential access, thus the cache miss penalty has the great potential to be reduced if hard disk works at the sequential mode. And then introduced the popular block-level cache replacement algorithms, such as, Belady’MIN [6], FIFO [6], LRU [5], LFU [6], LRU-K [6], 2Q [10] etc. These algorithms tend to improve the hit ratio, rather than reduce the miss penalty. Finally, a novel and simple replacement algorithm Bargain Cache [7] was proposed, which uses file-system metadata to reduce the miss penalty and the mean access time. Experiment results show that Bargain Cache performs better than LRU and that it is an effective way for different workloads and cache sizes. There are two limitations. First, no comparison of complexity of Bargain Cache with that of other algorithms was computed. Second, only Bargain Cache and LRU algorithms were implemented in the simulation. It would be more convicive to compare these with more replacement algorithms.

G. A Software Strategy to Improve Cache Performance

In embedded systems, cost, power consumption, and die size requirements push the designer to use small and simple cache memories. Such caches can provide low performance because of limited memory capacity and inflexible placement policy. A way to increase the performance is to adapt the program layout to the cache structure. This strategy needs the solution of an N-P complete problem and a very long processing time. This technique [8] proposed a strategy to look for a near optimum program layout within a reasonable time by means of smart heuristics. This solution didn’t added code and used standard functionality’s of a linker to produce the new layout. The approach is able to reduce up to 70% the misses in case of a 2-kbyte direct access cache.

III. Proposed Technique

Here, we present a new technique to reduce miss penalty in set associative caches that make use of extra free cache line reserved in every set that can hold a block coming from RAM. In set associative caches blocks can be placed anywhere in the set in the cache. Block replacement policies plays an important role to reduce the miss penalty. In this paper the technique make use of Random block replacement policy because it is simple to build in the hardware. So, in set associative cache we can replace any cache line from a specific set. A set-associative scheme is a hybrid between a fully associative cache, and direct mapped cache. It is considered as a reasonable compromise between the complex hardware needed for fully associative caches (which requires parallel searches of all slots), and the simplistic direct-mapped scheme, which may cause collisions of addresses to the same slot (similar to collisions in a hash table). So first we see what happens on a miss penalty in set associative write back cache with a random replacement policy and that we will see how the new strategy to reduce the miss penalty works. Suppose that we have 4-way set associative cache shown in the following figure and suppose all the lines are full as shown in figure 2.

![Fig. 2: 4-way Set Associative Cache](image)
When processor fails to find an item in the cache and cache miss occurs then miss penalty first need to take a block from a specific set. Suppose that the evicted block in the cache which is going to be replaced is dirty block i.e. it is modified in the cache but not copied inside memory. Now what happen normally is that the cache controller first copies the cache block in the memory to make a room for new block inside that specific set and after this the cache controller than moves the memory block inside that cache line. So, all replacement policies first transfer the evicted block to memory and then a block from memory is transferred to that cache line.

But our idea is to reduce the miss penalty works as follows. There is one free cache line inside each set of 4-way set associative cache as shown in the figure 3.

Fig. 3: 4-way Set Associative Cache With Free Line

Now, when a cache miss occurs the block is first transferred from memory to the free line of a specific set and is made available to the processor, so that memory stall cycles can be reduced. After this, the evicted dirty block is transferred from cache to the main memory. The evicted block line is then marked as free cache line. In this strategy, the miss penalty is reduced as the block is made available to the processor in the first step instead of first transferring the evicted block and then moving block from memory to the cache.

IV. Conclusion
The miss penalty problem reduces the overall efficiency of the processing environment. Several efforts have been made to improve the performance of the cache by reducing the miss penalty rate. Improving miss penalty and cache replacement policy has been a hotspot of researchers due to the limitation of available memory compared to ever increasing capacity of the hard disks. Though many techniques have been supposed to reduce miss penalty rate still further improvements are possible. If we combine our proposed technique with critical word first and early restart miss penalty reduction technique then we can get more performance of cache. This technique can be used with any Block replacement policy and can be combined with any existing miss penalty reduction technique. Our proposed technique works fast because on a miss, the block from memory is first transferred to the free cache line and delivered to the processor and then evicted block is copied to the memory. However, this technique is a little costly because if cache line size is 1kb then the size of a 4 way set associative cache is 5 kb instead of 4 kb. This means that 1kb of cache memory is serving as an extra memory. Since cache memory cost is high, this technique requires more cost in terms of economy.

References
Miraj Gul received his M.Sc. degree in Computer Science from the University of Peshawar, Peshawar, Pakistan in 2003. He joined the Virtual University of Pakistan in 2004 as instructor. In Oct 2006, he joined Deptt of IT, Hazara University, Mansehra, Pakistan as lecturer and working till date. Now he is currently pursuing his MS degree. His research interests include Advance Architecture, Advance analysis of Algorithms, Dynamic Programming and Game Theory.

Noorul Amin received his M.Sc. degree in Computer Science from the University of Peshawar, Peshawar, Pakistan in 1996. He joined the Fazaia Degree College Risalpur, Pakistan in 1997 as Lecturer. In Oct 2006, he joined Deptt of IT, Hazara University, Mansehra, Pakistan as Assistant Professor and working till date. He is also serving as Head of Department of IT. He Completed his MS in Networking in 2011. Now he is currently pursuing his PhD study. His research interests include Advance Architecture, Networks, Networks Security and Cryptography.

Faisal Bahadur received his M.Sc. degree in Computer Science from University of Peshawar, Peshawar, Pakistan in 2001. He joined the Agriculture University of Peshawar, Pakistan in 2004 as Lecturer. In Oct 2006, he joined Deptt of IT, Hazara University, Mansehra, Pakistan as lecturer and working till date. He has completed his MS degree in Software Engineering from the Virtual University of Pakistan in 2012. Now he is currently pursuing his PhD degree. His research interests include Advance Architecture, Software Engineering and Thread Polling Techniques in Web.