An Overview of Compiler Based Cache Optimization Techniques

Authors
Virendra Kumar Gautam¹, Dr. P K Singh²
¹M. Tech Dept. of Computer Science and Engg. MMM University of Technology Gorakhpur, Uttar Pradesh
²Associate Professor Dept. of Computer Science and Engg. MMM University of Technology Gorakhpur, Uttar Pradesh
Email- vkgautam01@gmail.com, vkg.mmmut@gmail.com

ABSTRACT
Cache plays an important role in computer architecture. It provides data and instruction at a faster rate to increase the utilization of CPU. This paper comprises of concepts related to the memory hierarchy, cache memory, its organization and various techniques used to optimize cache to increase the performance of CPU. Computer architecture uses memory hierarchy to deal with the low memory bandwidth and latency of main memory access, at the end of this paper we are aware of cache and its various compiler based optimization in cache.

Keywords—Cache optimization; Cache miss; latency: bandwidth; memory hierarchy

1. INTRODUCTION
Cache memory is fast and expensive. Basically it is a buffer memory and fast. It kept those items which are most accessed by the programs. CPU performance is increase at 60 percent per year, whereas DRAM (main memory) performance is increase at average rate of 10 percent. So there is a large gap between the performances of these two, which makes CPU in ideal state. To hide the impact of this gap we use fast cache memory.

It improves the performance of program execution by keeping the recently used data. If the data is found in cache it is called cache hit. Otherwise cache miss occur.[1][2] On cache miss the requested memory reference is fetch from main memory to cache. A number of work is done to reduce this cache miss occurred.

Cache is characterized mainly by the three parameters.
1. Block size 2. Capacity 3. Associativity

Block size: On cache miss how many contiguous bytes are fetched.
Capacity: Number of bytes that cache can hold.
Associativity: Associativity means the number of unique places in the cache a particular block reside. In full associativity, block has exactly one place.

In memory hierarchy a very fast memory that is access by the processor without delay and work at the speed of microprocessor itself i.e. register. Processor kept some intermediate produced data and data that are used in future.[1] After the register, cache memory comes. The cache that is present in the processor is called L1 cache. Cache on separate chip next to processor is called L2 cache. Sometime L3 cache is also used on separate chip.
2. ARCHITECTURE AND PERFORMANCE EVALUATION OF CACHE

Sometimes L1 and L2 cache is on the same chip and L3 on separate chip. L1 cache is divided into two parts L1 (data) which keeps the data and second part L1 (instruction) keeps the instructions that are recently used or mostly used by the processor.

![Memory Hierarchy Diagram](image)

Fig.1. Memory Hierarchy

Generally latency of L1 cache is mainly one or two cycle. The L1 cache is backup by the L2 cache. L2 cache provides data with the latency 5 to 10 cycle. L3 cache is off the chip there size is large in comparison to L1 and L2 cache. L3 cache provides data with a delay of 10 to 20 cycles.

3. LOCALITY OF REFERENCES

Cache is small and has few spaces to hold data and code. Due to its small and limited size it can hold copies of recently used data. Performance of cache is improved when data is reused before being replaced by other. Principle of locality of reference in cache is reason for reducing the executing time. This principle state that recently used data has higher chances to use in future. Locality is subdivided into two parts

1. Temporal locality
2. Spatial locality

If recently used data are chances to again access in near future then the sequence of references shows temporal locality. A sequence of references shows spatial locality if data located close together in address space have chances to access.

4. ASPECT OF CACHE MEMORY

Cache line in cache stored the data. Contiguous block of main memory are present in cache line. If CPU requested data searched in cache and if it found in cache line then it is called cache hit. If not found in cache line it is called cache miss. The requested memory block containing the requested word are fetched from a lower memory and copied to cache line. Mapping techniques are used to bring block from main memory to cache [2]. In direct mapping the main memory block content is placed into one cache line. Direct mapped cache is used in past and still it is used for off chip caches. In full associative memory block content placed in any cache line. In c-way set associative the C represent the number of cache line. Problem arise here which block is placed into which cache line and which block has to be replaced. This is decided by replacement strategy. Today we use least recently used LRU and random.

5. MEASURING AND SIMULATING CACHE BEHAVIOR

Many profiling tools are available to check whether the code runs efficiently or not. It identify the performance bottleneck and to guide for code optimization [3]. To hide the existence of cache memory is one of main fundamental concept in any memory hierarchy. Many microprocessor manufactures add
dedicated register to CPU for counting certain events. These dedicated register are called hardware performance counters.

The information gathered by hardware performance counter is different for different platforms [4, 5].

The quantities that can be measure are:
1. Cache hit, cache miss
2. Pipeline stalls
3. Processor cycle
4. Instruction issues
5. Branch mispredictions

The profiling tools that are based on hardware performance counter are:
1. Performance counter library
2. The performance application programming interface (PAPI)
3. Digital continuous profiling infrastructure.

### 6. BASIC TECHNIQUES FOR IMPROVING CACHE EFFICIENCY

#### 6.1 Data access optimization:

Data access optimization are used to change the way in which iteration in a loop is executed, this is achieved by some modification in code. The reason behind code transformation is to improve temporal locality [6]. Which type of code transformation is used to increase performance is not an easy task compiler use heuristics to select the type of transformation [7].

#### 6.1.1 Loop interchange:

This transformation interchanges the two adjacent nested loops [6]. Loop interchange is used to improve register reuse, improve vectorization and parallelism [7].

![Fig.2. Access patterns for interchanged loop nests.](image)

For example

1. double sum;
2. double a[n,n];
3. //original loopnest:
   4. for (j=1 to n do)
   5. for (i=1 to n do)
   6. sum = sum +a[i,j];
   7. end for
   8. end for
4. for (i=1 to n do)
5. for (j=1 to n do)
6. Sum= sum+a[i,j];
7. end for
8. end for

Assume 4*8 array stored in row major order in memory. In row major order two elements of array are adjacent if second index is increasing order. (1,1) (1,2) (1,3)….. are said stored in row major order. In left code mentioned above access array in column wise (1, 1) (2,1) (3,1)…..so on. Cache fetches the array of
element in row wise because array is stored in row wise. So data in cache is no longer use. Every time we get cache miss. After interchange loop we see data are accessed in row wise and cache also stored data in row of array element. So no misses are found therefore performance increase.

6.1.2 Loop fusion
Loop fusion sometimes also known as loop jamming. It is a transformation that adds two different loops that have same iteration into a single loop. It works when loop has no flow, antidependent [9]. This will reduce the loop overhead.

Example- Loop fusion

6.1.3 Loop blocking
(code for dense matrix multiplication)

In left side of code we see the array b is loaded in the cache by first loop. Which is not remaining in cache, but the second loop reload the same data from main memory. If both are performed in single loop then only one global iteration performed to get the result. Loop fusion improve temporal locality and reduced miss rate.
\[ r = r + y[i]k^*z[k][j]; \]
\[ X[i][j] = X[i][j] + r; \]
\]

After blocking capacity misses reduce from \( 2N^3+N^2 \) to \( N^3/B+2N^2 \) [10]. Here B is called Blocking factor. This reduction is due to doing computation on B*B sub matrix that fits into the cache. Loop blocking exploits a combination of spatial and temporal locality. But it may suffer from conflict misses.

The smaller number of element is accessed in comparison to previous element access.

6.1.4. Data Prefetching

Above loop transformation reduces capacity misses while the program in execution. First time miss occur when execution begin is not addressed by above method. Prefetching makes the microprocessor to generate a data request before actual computation start. Prefetching is done either by hardware or software. Many microprocessor implement data prefetching instruction as a regular instruction. This prefetching instruction is basically a hint for processor to load specific data item. Sequential prefetching is simplest hardware based prefetching. It is overhead to system.

6.2. Data layout optimization

Data layout optimization changes the way in which data structure and variable present in memory. Its main aim is to avoid conflict miss and false sharing. Data layout optimization improves spatial locality of code. It involves the changing base address of variable, merging arrays, array dimension and modifying array sizes. This technique is used at compile time and some part in runtime. Array padding and array merging are mostly used technique.

7. PERFORMANCE EVALUATION

There are various techniques are available to optimize the cache. We compare some of the techniques for different parameters which are shown in Table 1. The simplest way to reduce miss rate is increase the block size but they will produce higher hit rate and also increase conflict misses. All optimization techniques categorized into compiler based, hardware based and operating system based. Muti-level cache is more efficient in improving overall system however design of multi-level cache is very complex.
Table 1 Performance of Cache Optimization Techniques On Some Parameters

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Miss rate</th>
<th>Hit time</th>
<th>Miss penalty</th>
<th>Hit rate</th>
<th>Access time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger Block size[11]</td>
<td>Decrease compulsory miss, increase conflict misses</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>Slow</td>
</tr>
<tr>
<td>Loop interchange</td>
<td>Reduce</td>
<td>high</td>
<td>-----------</td>
<td>high</td>
<td>-----------</td>
</tr>
<tr>
<td>Loop fusion</td>
<td>Reduce</td>
<td>high</td>
<td>-----------</td>
<td>high</td>
<td>-----------</td>
</tr>
<tr>
<td>Loop blocking</td>
<td>Reduce</td>
<td>high</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>Data prefetching</td>
<td>Reduce</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>LR+5LF[13]</td>
<td>Reduce cache misses with greater amount than LRU, FIFO and LFU at L1 cache and same for L2</td>
<td>Same as LRU and LFU</td>
<td>Same as LRU and LFU</td>
<td>Same as LRU and LFU</td>
<td>Same as LRU and LFU</td>
</tr>
<tr>
<td>Higher Associativity[14][15]</td>
<td>Reduce capacity, conflict miss</td>
<td>High</td>
<td>Less</td>
<td>Low</td>
<td>Fast</td>
</tr>
<tr>
<td>Multilevel cache[14]</td>
<td>Cache coherence Miss takes place</td>
<td>High</td>
<td>Low for L1</td>
<td>High</td>
<td>Slow</td>
</tr>
</tbody>
</table>

8. CONCLUSION

In this paper, we discussed various compiler based cache optimization techniques. We realize that one technique is not best choice for all cases. Each technique is used for specific purpose. Each techniques has certain advantage and disadvantage. For example using larger cache size reduce miss rate but at the cost of higher hit time. So we have to use all the techniques in such a way that overall system performance will enhance. In future we intend to design cache configuration for multicore processor and techniques to optimize the performance.

REFERENCES