International Journal of Engineering & Technology, 7 (2.19) (2018) 80-83

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Request Schedule Oriented Compression Cache Memory

G.D.Kesavan[1*], P.N.Karthikayan[2]

[1] *Assistant Professor*
Department Of Computer Science And Engineering, School Of Computing,
Vel Tech Rangarajan Dr. Sagunthala R&D Institute Of Science And Technology,
Avadi, Chennai-600 062, Tamil Nadu, India.
[2] *Assistant Professor*
Department Of Information Technology And Engineering, School Of Computing,
Vel Tech Rangarajan Dr. Sagunthala R&D Institute Of Science And Technology,
Avadi, Chennai-600 062, Tamil Nadu, India.
* *Gdkesav@Gmail.Com*

Abstract

Using cache memory the overall memory access time to fetch data gets reduced. As use of cache memory is related to a system's performance, the caching process should take less time. To speed up caching process, there are many cache optimization techniques available. Some of the cache optimization process are Reducing Miss Rate, Reducing Miss Penalty, Re-ducing the time to hit in the cache etc. Re-cent advancement paved way for compressing data in cache, accessing recent data use pattern etc. All the techniques focus on increasing cache capacity or replacement policies in cache resulting in more hit ratio. There are many cache related compression and optimization techniques available which address only capacity and replacement related optimization and their related issues. This paper deals with scheduling the requests of cache memory as per compressed cache organization. So that cache searching and indexing speed gets reduced considerably and service the request in a faster manner. For capacity and replacement improvements Dictionary sharing based caching is used. Through this scheme multiple requests are foreseen using pre-fetcher and are searched as per cache organization, promoting easier indexing process.The benefit comes from both compressed storage and also easier storage ac-cess.

Keywords: *Dictionary based cache compression, cache compression, cache optimization*

1. Introduction

Cache memory is small memory placed near processor and are used to store recently accessed data mem-bers, so that the data members need not be fetched from lower level memories. If the data members in cache are available on a request, the request is a hit. Else if the data members in cache are not available on a request, the request is a miss.

The cache memory is a faster memory and size of cache should be limited so that the time to access a cache and process data should be lesser than that taking from lower memories. The caching operations are cache ll and cache replacement. Cache ll takes data from lower memory and places in the available location. The process happens when cache is empty or when a new victim location is found. Cache Replacement is the process of choosing a victim whenever a cache is full.

Finding similarities between data members in memory and representing them in a optimal way to gain memory fractions with easier access is called as Data Compression in Memory. Storing data in a compressed way, needs decompression to read its actual contents. Compression can be done at any level of memory in Memory Hierarchy.

When Memory compression used in main memory reduces the size or number of paging requests. While that used in cache memory reduces the number of line requests. Using memory compression capacity of memory increases, memory is utilized effectively and also page fault rate gets reduced. Storing Spatially local data closely, the method reduces energy and bandwidth demands. The disadvantage of compressing data are additional encoding and decoding delay.

The paper is organized by describing Compression using Dictionary, Role of Scheduling in Cache, Working Mechanism, Future Works.

2 Compression Using Dictionary

2.1 Dictionary Based Compression

Cache memory is divided into cache lines. Repetitions in data members within a cache line is examined and placed inside a small referential memory space called dictionaries. The number of

distinct dictionary elements if n, then number of bits representing the data is log(n). There are many variations of dictionary based compression. Some Compression uses a dictionary look up for small and repeated values.

2.2 Shared Dictionary Based Compression

The Dictionary can be shared among lines if there is commonality of data across lines. The method com-pacts multiple blocks based on data contents not by compression factor. Usually Dictionary based cache compression encodes cache with dictionary, which is neighbor cache sharable, stores 4 byte chunk of cache block and helps reduce decompression latency.

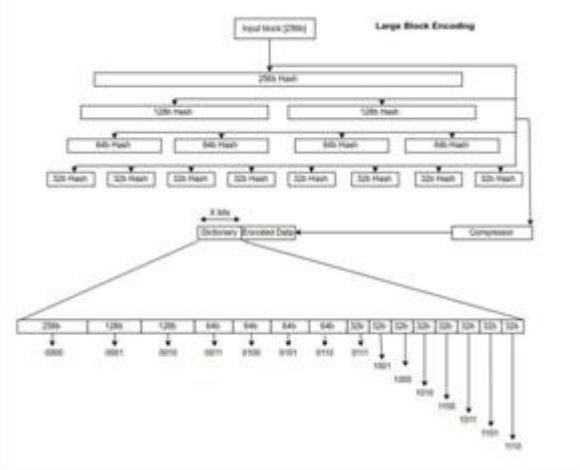

Figure 1: Compression by Dictionary Encoding

The dictionary is of variable sized from 256b to 32 bit hash. If the values in dictionary if matches the 256b wholly, then the encoded bits are rep-resented in place of them. else leaving MSB, the data is checked if it matches. Else it goes on till 32 bit hashes. Whenever the data matches the coded bits are represented.

Dictionary sharing is used because Single super block tags is a aligned continuous group of tags, having compression factor information and a status bit to indicate data in compressed or uncompressed form. Compacting using compression factor may allocate to incomplete super block with same compression fac-tor or new super block which waste space in LLC layer. By this methodology a block gets compacted along with its neighbors based on data not on compression factor data, improving the cache layouts.

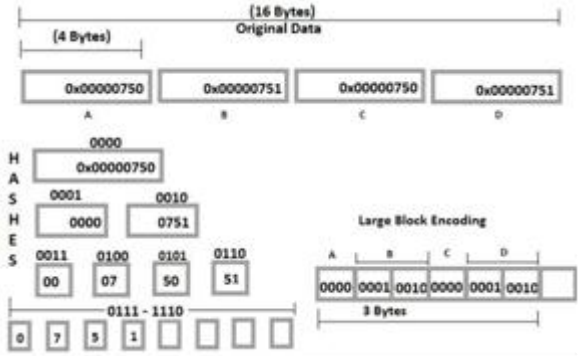

Figure 2: Compression by Large Block Encoding

Two encoding schemes are used to exploit multiple shared blocks. DISH does not limit the size of com-pressed block but extracts 4-byte chunks of cache block and compress using them. Compact the multiple com-pressed if 4 byte chunks are same. If the number of cache blocks are more than chunks then the data is treated as incompressible and makes appropriate entry in the dictionary. Fixed width pointers used to point to dictionary entries. Same issue can be addressed by valid/Invalid bit for each set of pointers and sets.

2.3 Dynamic Dictionary Based Compression

The Dictionary can be made while storing the data in cache and may vary from one cache line to another.

Enhance the compaction of multiple contiguous blocks into single block by not using compression factor but based on their data contents. B2 can be placed in data entry 1 as it has enough memory space. If placed B2 will be adjacent to spatially local B1.

The skewed compressed cache(SCC), decoupled compressed cache and Yet another compressed cache(YACC) have minimal storage and latency over-heads by compressing blocks and compact compressed blocks of similar compression factor into single data en-try. DISH encodes the cache blocks with a dictionary, sharable by neighboring cache blocks, without changing cache layout but improving spatial locality. The YACC is used as a baseline scheme without compression factor based compaction. The blocks are divided into 4 byte chunks. DISH uses dictionary for compression and compaction.

CPACK+Z compression is used which maintains the dictionary for repeated patterns and a pointer points to which data item from dictionary it belongs to. The term Z denotes a single bit is represented when the data is completely Zero. Base delta immediate compression is used when upper half of the data chunk have repetition but not the whole chunk. There are two schemes of representation in DISH. Scheme I is used for cases where the number of similar data contents across multiple contiguous cache blocks is less. Scheme II is used for cases where the number of repeated upper data bits across multiple contiguous cache blocks are more.

For a 64 byte block, the data is divided into 4 byte chunks. Distinct 4 bye chunks from data blocks are entered in dictionary. The number of dictionary elements if are eight then three bits are used to represent them. After framing dictionary, the data elements are represented by their appropriate pointers and a valid bit in the dictionary. If the number of dictionary elements are less than eight then scheme I works.

The next block who has same dictionary values as previous are compacted together. Thus enabling inter block data localities. The compression locality is used adjacent cache blocks tend to compress to same size. So use of same dictionary and packing them together using the same super block tag into a single data entry. Non contiguous blocks can also be compacted together. Figure 18 shows compaction of blocks by scheme I where a dictionary of eight distinct elements are formed and three bit pointers point the dictionary elements. The same dictionary if possible can be taken for subsequent blocks.

For a 64 byte block, the data is divided into 4 byte chunks. When the number of dictionary elements are more then the upper bits common are noted down. In this scheme, four 28 bit dictionary which have upper half same among data blocks are noted down. The same way two bits choose the dictionary value and lower four bits are stored in their appropriate place.

This is scheme II. When the number of dictionary here exceeds four, then the block is termed in-compressible. In-compressible blocks are allocated in last level cache with compaction factor as one. The tag entry for a su-per block contain super block tag, compression factor, block id and coherence state. The address bits contains super block tag, index, block id and o set. When the number of dictionary elements increases, the refer bits will increase lead to more space than actual block size. Such cases are termed as in-compressible blocks. In-compressible blocks can be checked if their

upper Most significant bits of four byte chunk, maximum of 28 bits, in the blocks are repeating. Such repeating data can be compressed using the Scheme II method. Sometimes the dictionary may have one or two dictionary entries. This is because when most of data chunks have same data in it. They are called as Incomplete Dictionary. When not knowing which scheme to use or arbitrarily choosing the scheme, set-dueling monitor help in deciding. It uses a 32 leader cache sets that uses scheme I and a 32 leader cache sets that uses scheme II. When there is a tie, the winning scheme is determined by follower sets. This method of choosing is termed as Adaptive DISH.

When decompressing, corresponding data bits are fetched and assembled as 4 byte chunks and then based on the tag the blocks gets retrieved. A valid bit is associated with each block of compacted data to be changed when write back of data is done. Practical implementation needs registers to indicate dictionary elements and for conditional execution. The encoding bit 0 for Scheme I and 1 for Scheme II in don't care place of YACC is used to differentiate Scheme I and Scheme II. If there is a block B1 fetched from DRAM, super block tags are checked for related B0 in cache. When there is a Block B0, its dictionary is checked with B1. When the super block has space and dictionary matches, the B1 is compacted in that space. Else if there is no space, separate cache line is allocated.

To compare dictionaries of two blocks, we replicate the compression hardware and preset the B0 dictionary for B1. This helps to and B1 compressibility, dictionary of B1 if same as B0 But this consume higher energy when third block or fourth block is compacted inside a super block. The write backs need to retrieve the dictionary, update the dictionary after compression. Sometimes the write backs may make compactable block in-compressible. This can be handled by invalidating the data accordingly and reallocating with same compaction process. When a miss in a super block tag may get data
from DRAM but next accessible block may be avail-able. There will be no extra access latency to retrieve those blocks. The dictionary elements are placed in compressor for compression. Decompression happens when there is a tag hit. The advantages of DISH are it avoids wastage of memory space by compacting different sized compressed data together based on their relatedness and no additional overhead for meta data storage.

3 Role of Scheduling in Cache

With use of Pre-fetching data members missing in cache can be fetched from lower memory prior to re-quests. This considerably minimizes the miss rate. As most of the data items accessed are spatially or tem-porally local. Spatial local items are closer data items which are accessed in previous requests. Similarly tem-poral data items are same data items that are accessed after few time slots. Pre-fetching expected data items are advantageous but may make system complex and also requires considerable additional energy associated cost.

3.1 Pre-fetching

Key features of prefetching are used for compression based on higher order bits. Prefetching information is gathered and all such blocks are compacted. Compact block are formed by multiple prefetched blocks and are placed in LLC. The compacted blocks should not be placed in di erent address locations. So without addi-tional lookups, we modify cache organization in LLC by di erent hash function.

Compression of data is done within blocks. Stream prefetching tracks many access tracks at a time and request many prefetches at a time. Stride prefetching gets stride memory length from past program counter activity and predicts future using it. Spacial memory streaming uses code based correlation to nd past pro-gram counter's data locality using which prefetching is done. Compression is based on BDI where data is represented by relative di erence from the reference.

4 Working Mechanism

4.1 Fetch Request History

A circular queue organized small memory which stores the main memory fetch details is called as Fetch Request History. The memory is similar to the direc-tory. Initially when the cache is empty compulsory misses happens. After these Compulsory misses, the fetched data item's memory addresses are noted in the History table. This history is used to nd stride dis-tance reference among requests previously made. The History table gets updated on each line request from Main Memory. If the Fetch Request History is full, the memory works as circular queue.

4.2 Predictors

The details of Fetch Request History table provides the address of data items that are requested previously. Also when there is a miss irrespective of prefetching, the related entries on Prefetch Request History are removed. This prevents misses due to useless prefetching. Using the hit data a stride distance in main memory is calculated and that distance is used to predict the future requests. The prediction is based on the probability of the data item to be accessed in future. These requests are then fed to the prefetcher.

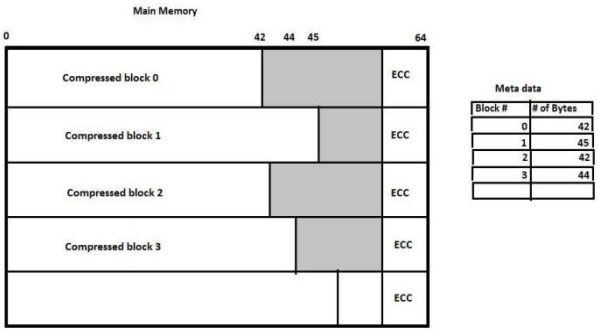

Figure 3: ECC for uncompressed data

4.3 Prefetcher

The Prefetcher collects the addresses of data items that are suppposed to be future requests. Then correspond-ing elements if not available already are requested. The prefetcher checks for retundancy as most of the data members used now are highly probable to be accessed in near future.

5 Future Work

Instead of scheduling based on the size of data el-ements, and grouping them as based on relatedness, some other factors can be imposed for a better asso-ciativity among them.

References

[1]A. Sha ee, M. Taassori, R. Balasubramonian, and A. Davis. Memzip: Exploring unconventional benfits from memory compression. In

2014 IEEE 20th International Symposium on High Perfor-mance Computer Architecture (HPCA), pages 638{ 649, Feb 2014.

[2]David J. Palframan, Nam Sung Kim, and Mikko H. Lipasti. Cop: To compress and protect main mem-ory. In Proceedings of the 42Nd Annual Interna-tional Symposium on Computer Architecture, ISCA '15, pages 682{693, New York, NY, USA, 2015. ACM.

[3]Tri M. Nguyen and David Wentzla . Morc: a manycore-oriented compressed cache. In Proceed-ings of the 48th International Symposium on Mi-croarchitecture, pages 76{88, 12 2015.

[4]Angelos Arelakis, Fredrik Dahlgren, and Per Sten-strom. Hycomp: A hybrid cache compression method for selection of data-type-speci c compres-sion methods. In Proceedings of the 48th Interna-tional Symposium on Microarchitecture, MICRO-48, pages 38{49, New York, NY, USA, 2015. ACM.

[5]G. Pekhimenko, T. Huberty, R. Cai, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. Exploiting compressed block size as an indicator of future reuse. In 2015 IEEE 21st International Symposium on High Performance Computer Archi-tecture (HPCA), pages 51{63, Feb 2015.

[6]K. Raghavendra, B. Panda, and M. Mutyam. Pbc: Prefetched blocks compaction. IEEE Transactions on Computers, 65(8):2534{2547, Aug 2016.

[7]Somayeh Sardashti, Andre Seznec, and David A. Wood. Yet another compressed cache: A low-cost yet e ective compressed cache. ACM Trans. Archit. Code Optim., 13(3):27:1{27:25, September 2016.

[8]B. Panda and A. Seznec. Dictionary sharing: An e cient cache compression scheme for compressed caches. In 2016 49th Annual IEEE/ACM Interna-tional Symposium on Microarchitecture (MICRO), pages 1-12, Oct 2016.

.