Vol. XVI & Issue No. 10 October - 2023
INDUSTRIAL ENGINEERING JOURNAL

PERFORMANCE IMPROVISATION INSIDE CACHE TILE USING MAPPING ALGORITHM

Ms. Suma sannamani
Dr. Manjudevi

Abstract

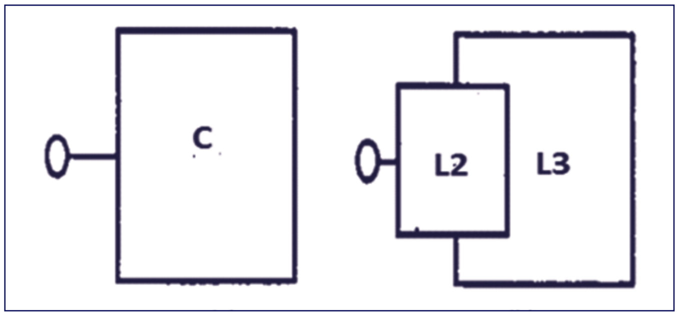
There is huge demand for improvisation of cache memory performance considering multiprocessors. Cache memory arranged in tile fashion. These cache tiles are placed in multiple levels. Here work involves survey of existing search algorithm to draw conclusions and to develop algorithm which reduces number of searches required. There by improvisation performance parameter latency. Latency reduction is achieved by avoiding unnecessary search through all location.

Keywords: Cache memory, Search algorithm, latency.

INTRODUCTION

Many applications including mobile and processors demand cache with high performance. Uniform cache architecture performance fig 1(a) is poor due to wire delay problem. Multiple level cache fig 1(b) helps to improvise performance by providing parallel access.

Figaure 1(a) UCA Figaure 1(b)ML-UCA

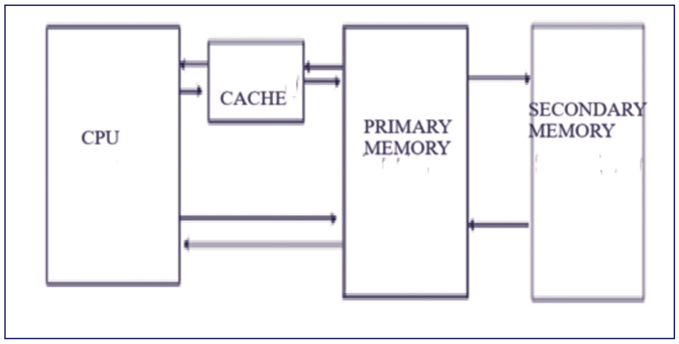


Cache is small and temporary memory that exist in between processor and main memory as shown in figure 2.

Multiple Levels of Memory:

1) Register – Registers exist on chip. These are fastest memory. Common registers available are accumulator, PC-program counter, Stack pointer/address register etc.
2) Cache memory – This is the memory we are working on. Here in cache data is stored temporarily with the advantage smaller access time. Hence it is the fastest memory.
3) Main Memory – It is a volatile memory; Size is large compared to cache. Less fast compared to cache.
4) Secondary Memory – It is non-volatile in nature. Speed is less compared to RAM but it has the advantage of storing data permanently. Key points to be considered during search inside cache are: 1) Hit: The address to be searched present in cache. 2) Miss: The address to be searched not found in cache.

Figure 2: CPU with multiple levels of memory



Cache Mapping: Data mapping is categorised into three types in cache memory. Direct, Set-Associative and Associative mapping. Here the design is made using direct mapping technique. Direct-Mapping is simplest way of mapping. It maps each main memory block into only specified cache memory line. The previous block is trashed when another main memory block needs to be loaded to same place. Address of cache is divided into tag and index field. Tag is stored in cache. The remaining part of address is same as main memory address.

i = n modulo m

i=cache memory line number

n= RAM block number

m=Total number of cache lines

BACK GROUND WORK

Acharya [1] presents cache trie search algorithms. There are main three important features that exist for this algorithm. B tree hash table vector using data structure is developed for each node. considering cache characteristics and the fan-out of the node, data structure is decided. Changing the data structure design in dynamic fashion, they adapt to changes in the fanout. The layout and size of data structure is depends on size of symbols in alphabet. Brodal [2] presents a cache-oblivious algorithm. In this algorithm there is no multiple cache hierarchy. Design is considered with only one level of memory. The design does not consider or it does not have knowledge of memory hierarchies. Design is analyzed in a two-level I/o model. Hence result or algorithm can be applied to multi-level hierarchies. search tress has search cost O is log $_B$N I/O s and it has search cost same as B-trees. Real-time indices are the aspects of modern search engines. These get incorporated the change in content within seconds. These search engines help to reduce user latency and back-end load. CIP – the cache invalidation predictor [3] is an architectural component. Researcher discuss cache memory replacement techniques. LRU-Least recently used technique fails to work with freshness over time and CIP gives better results with 97 percent of queries with fresh results. T. C. Xu [4] work on mapping algorithm considering big data applications. Data processing is on demand. Data needs to be processed very fast. Work is to explore characteristics of data applications with multi core processor and shared cache. Performance metrics considered is latency. Belady's algorithm [5] is an optimal solution. This algorithm is infeasible as it needs understanding of the future. The researcher justifies that the algorithm is efficient way, as it introduces a unique way for efficiently simulate Belady's behavior and to represent the long history information they have used a known sampling technique that is needed achieve high accuracy.

This paper [6] explain a algorithm is to implement, deployment experience of cache stack. Cache traffic is portioned into disjoint categories by cache stack and analyze the benefit for cache from each subset. Knapsack problem is formulated to match the best admission policy to each cache set. Tree-based backtracking search [7] is a technique to find solution for distributed constraint optimization problem. Here there is effective reuse of historical search. Hence, reducing the overall overhead. This paper proposes Retention Scheme (RS) for tree-based synchronous backtracking search. Javier marino [8] presents dynamic cache partitioning. Divides L2 bank into a private and shared partition. Placing private data closure to core processor. Depending on the work load proposed architecture self adjust the sharing partition.

Cache indexing policy [9] reduces the conflict misses. Indexing policy spreads the references to the all-cache memory sets. Power consumption is the performance parameter considered. Merino, V [10] proposed Enhanced Shared-Private Non-Uniform Cache Architecture (ESP-NUCA), which is suitable for high performance processor and it is also cost effective. During run time to determine the cache memory configuration the approaches are introduced. These approaches are based on algorithms to test cache configuration in variety of orders [11, 12, 13,14, 15]. Energy consumption estimations with mathematical model are used to determine cache performance during runtime [11,12,13,15] or to find cache miss rate[14]. Reconfiguration is made according to selected configuration candidate. Then comparison is made between previous one and reconfigured cache. Conclusion is drawn.

DESIGN

Considering physical address bits as 12 bits. One cache tile has 32 locations. There exist eight blocks. Each block contains four bytes.

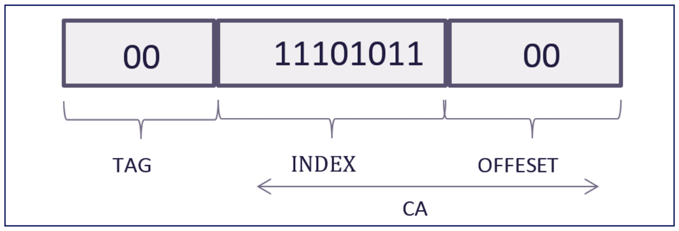
Offset bits are $\log_2$ (number of words per block) = $\log_2(4)$=2

Index bits are $\log_2$ (number of blocks in cache) = $\log_2(256)$ =8

Tag bits=Physical address-offset bits-Index bits=2

As each tile has 32 location, always upper 5 bits of Index remain same. Hence, it reduces 32 location address comparison to 1 comparison. Hence this search algorithm by using direct mapping improves performance.

Figure 3: Physical address



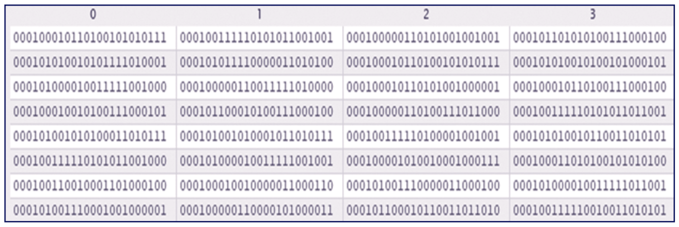
TAG: TAG bits

CA=INDEX bits + offset bits

DATA=Data bits of cache

RESULTS

Figure 4: Stored memory of a tile

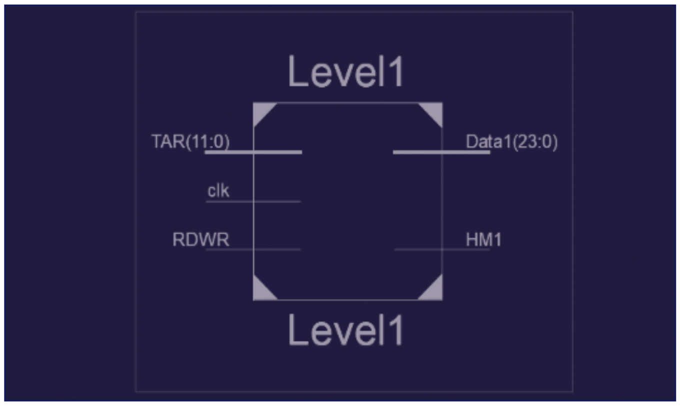


0	1	2	3
000100010110100101010111	000100111110101011001001	000100000110101001001001	000101101010100111000100
000101010010101111010001	000101011110000011010100	000100010110100101010111	000101010010100101000101
000101000010011111001000	000100000110011111010000	000100010110101001000001	000100010110100111000100
000100010010100111000101	000101100010100111000100	000100000110100111011000	000100111110101011011001
000101001010100011010111	000101001010001011010111	000100111110100001001001	000101010010110011010101
000100111110101011001000	000101000010011111001001	000100001010010001000111	000100011010100101010100
000100110010001101000100	000100010010000011000110	000101001110000011000100	000101000010011111011001
000101001110001001000001	000100000110000101000011	000101100010110011011010	000100111110010011010101

Figure 4 shows memory picture of the cache tile with twenty four bit data stored. There are eight blocks with each block containing four locations. Figure 6 Search algorithm experiencing hit in tile1 with HM_Le1 going high. Hence data reflected on DATA bits otherwise it will be unknown.

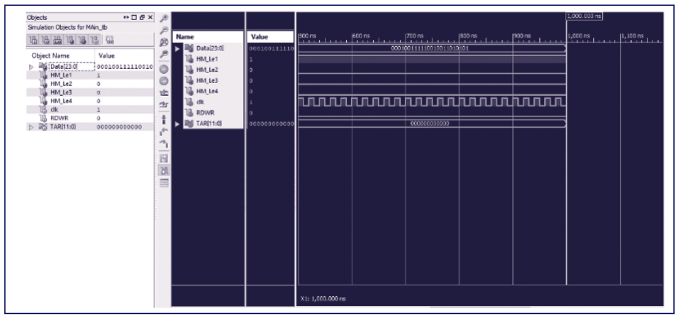
RTL schematic of tile is shown in figure 5. Address to be searched is sent through TAR _Test Address Register. RDWR represents weather operation related to read or writing data from or to cache memory. Data1 will have 24-bit data if search experience hit otherwise, it remains empty. HM1 represents weather tile experience hit/miss.

Figure 5: RTL Schematic of a tile



TAR: Test Address Register
clk: Clock
RDWR: Read/Write operation
HM1: Hit or Miss in Tile
Data: Data if tile has experienced hit
Tool used: Xilinx ISE

Figure 6: Simulation results



Object Name	Value
Data[23:0]	000100111110010
HM_Le1	1
HM_Le2	0
HM_Le3	0
HM_Le4	0
clk	1
RDWR	0
TAR[11:0]	000000000000

HM_Le1:1=hit,0=miss

Data: Data from location where hit occurred

TAR: Search address

Figure 6 depicts search through tile_1 experiencing hit . Hence HM_Le1 is high and data from location where hit has occurred is reflected on to data lines.

CONCLUSION

The design of cache tiles helps to avoid unnecessary search. The design uses direct mapping technique. The design of tile is such that it removes thirty one comparisons of index bits and helps to improve the performance by reducing latency. Performance improvisation by reducing latency is the very important when it comes application of cache memory which is achieved with minimum Gate delay of 0.575 ns.

ACKNOWLEDGEMENT

The authors would like to thank Visvesvaraya Technological University, Belagavi and the Department of ECE, Oxford College of Engineering, Bangalore for the assistance provided for this work.

REFERENCE

1. *Acharya, Anurag, Huican Zhu, and Kai Shen. "Adaptive algorithms for cache-efficient trie search." Workshop on Algorithm Engineering and Experimentation. Springer, Berlin, Heidelberg, 1999.*
2. *Brodal, GerthStølting. "Cache-oblivious algorithms and data structures." In Scandinavian Workshop on Algorithm Theory, pp. 3-13. Springer, Berlin, Heidelberg, 2004.*
3. *Bortnikov, E., Lempel, R., Vornovitsky, K. (2011). "Caching for Realtime Search". In: ,* et al. Advances in Information Retrieval. ECIR ٢٠١١. Lecture Notes in Computer Science, *vol 6611. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-20161-5_12.*
4. *T. C. Xu and V. Leppänen, "A cache- and memory-aware mapping algorithm for big data applications," 2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC), 2015, pp. 110-115, doi: 10.1109/ICDIPC.2015.7323015.*
5. *Akanksha Jain Calvin Lin.2016 " Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement" . In Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16). IEEE Press, 78–89.*
6. *Tzu-Wei Yang, Seth Pollen, Mustafa Uysal, Arif Merchant, and Homer Wolfmeister, Google . "Cache Sack: Admission Optimization for Google Datacenter Flash Caches "USENIX Annual Technical Conference USENIX Association ,2022. https://www.usenix.org/conference/atc22/presentation/yang-tzu-wei*
7. *Wang, Jie* ; *Chen, Dingding* ; *Chen, Ziyu* ; *Liu, Xiangshuang* ; *Gao, Junsong," Completeness Matters: Towards Efficient Caching in Tree-Based Synchronous Backtracking Search for DCOPs",28th international conference on principles and practice of constraint programming,pp :39:1-39:17*
8. *Javier Merino, Valentín Puente, Pablo Prieto, José Ángel Gregorio," SP-NUCA: A Cost Effective Dynamic Non-Uniform Cache Architecture", ACM SIGARCH Computer Architecture News 64 Vol. 36, No. 2, May 2008 ,pp-64-71.*
9. *Ros, P. Xekalakis, M. Cintra, M. E. Acacio and J. M. Garcıa, "Adaptive Selection of Cache Indexing Bits for Removing Conflict Misses," in IEEE Transactions on Computers, vol. 64, no. 6, pp. 1534-1547, 1 June 2015, doi: 10.1109/TC.2014.2339819.*
10. *Merino, V. Puente, and J. Gregorio, "ESP-NUCA: A low-cost adap- tive non-uniform cache architecture," in Proc. 16th Int. Symp. High Per- formance Comput. Architecture (HPCA'10), 2010, pp. 1–10.*
11. *Ann Gordon-Ross, Frank Vahid, and Nikil D. Dutt. 2009. Fast configurable-cache tuning with a unified second-level cache. IEEE Trans. VLSI Syst. 17, 1 (2009), 80–91*
12. *Osvaldo Navarro and Michael Hubner. 2014. An adaptive victim cache scheme. In Proceedings of the 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig'14). IEEE, 1–4.*
13. *Osvaldo Navarro, Tim Leiding, and Michael Hubner. 2015. Configurable cache tuning with a victim cache. In Proceedings of the 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC'15). IEEE, 1–6.*
14. *Chuanjun Zhang, Frank Vahid, and Roman Lysecky. 2004. A self-tuning cache architecture for embedded systems. ACM Trans. Embed. Comput. Syst. 3, 2 (2004), 407–425.*
15. *Bruno A. Silva, Lucas A. Cuminato, Vanderlei Bonato, and Pedro C. Diniz. 2015. Run-time cache configuration for the LEON-3 embedded processor. In Proceedings of the 28th Symposium on Integrated Circuits and Systems Design (SBCCI'15). ACM, New York, NY, Article 42, 6 pages. DOI:https://doi.org/10.1145/2800986.2801026.*

AUTHORS

Ms. Suma sannamani, Research Scholar, Visvesvaraya Technological University, Jnana Sangama, Belagavi – 590 018, Karnataka, India
Email: sumabs2014@gmail.com / 088926 31066

Dr. Manjudevi , Professor & Head, Department of Electronics and Communication Engineering, The Oxford College of Engineering, Bommanahalli, Hosur Road, Bengaluru – 560 068, Karnataka, India
Email: manju3devi@gmail.com / 094487 61979