International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6, August 2019

# A Variable Processor Cache Line Size Architecture

**S Subha**

***Abstract:*** *Processor caches have fixed line size. A processor cache defined by tuple (C, k, L) where C is the capacity, k associativity and L line size has fixed values for the parameters. Algorithms to have variable processor cache line size are proposed in literature. This paper proposes algorithm to have variable cache line size based on the miss count for any application. The line size is varied by increasing or decreasing line size based on the miss count for any time interval. The algorithm can be used in running any application. The SPEC2000 benchmarks are used for simulating the proposed algorithm for cache with one level. The average memory access time is chosen as performance parameter. A performance improvement of 12% is observed with energy saving of 18% for chosen parameters.*

*Keywords: Average Memory Access Time, Cache line size, Energy, Set associative cache*

## I. INTRODUCTION

The central processing unit is the heart of the computer. The processor cache improves the program execution time by providing faster data and instruction access. The tuple (C, k, L) denotes a processor cache where C is the capacity; k is the associativity and L the line size. Usually all the three parameters are fixed at processor design time. Three types of processor caches are defined namely direct mapped, set associative and fully associative based on line placement. A line is placed in fixed place in direct mapped cache. It can occupy any of the k possible locations in k-way set associative cache. It can occupy any place in fully associative cache of n blocks. The cache has sets. The direct mapped cache and k-way set associative cache has S sets. The fully associative cache has one set. Any address is mapped to a mod S set number with a div S tag value in direct mapped and set associative cache. An address with data is placed in fully associative cache of n blocks in any vacant block. If all the ways of mapped set in set associative cache or blocks in fully associative cache are filled a replacement algorithm is used to place the incoming line. Usually the least recently used (LRU) algorithm is used for this. The line size L is the amount of data fetched from memory to the cache.

Modern processors have multilevel caches. Caches can be inclusive or exclusive or two type [2, 8].

Revised Manuscript Received on August 30, 2019.

* Correspondence Author

**S Subha*,** Department of IT and Engineering, SITE, Vellore Institute of Technology, Vellore, T.Nadu, India

In exclusive cache a line is present in one cache level. In inclusive cache a line at level i is present in level i and higher cache levels. In two type data cache, a line is treated as inclusive or exclusive based on access pattern [8].

Algorithms to have variable cache line size are presented in literature. In the work [3] an algorithm to vary line size by factor of two based on miss ratios is proposed by the authors. The authors found that the miss rate can be improved in this method. A method to have variable cache line size by inspecting the program behavior was proposed in [7]. The performance improved by 78% with the model for the chosen program segment. The cache line size is not necessarily power of two in this mode. However the cache is treated a s array with defined bounds for various variables in the program segment. The works [5, 6] proposes variable block size for loops and matrix multiplication.

This paper proposes variable cache line size where the line size is multiple of default line size. The lines are in set associative cache. The proposed algorithm starts with default line size. When the misses exceed threshold value, the line size is incremented by one. The existing lines are invalidated and the program continues with new line size. For calculating the set number and the tag value, the default line size is taken. However based on the current line size, the line is fetched from main memory, the status of cache ways is updated to indicate the validity of the cache way. If the line size cannot accommodate the lines in a set, the line size is decremented, the ways reinitialized and the program continues. The proposed model was simulated on cache system with one cache level with SPEC2000 benchmarks using Simple scalar Toolkit. The average memory access time (AMAT) is used for performance measurement. The proposed model is compared with set associative cache of same size. The performance improved by 12% for the simulated cache. An improvement in hits was observed. An improvement in energy savings of 18% was observed from the simulation.

The organization of the paper is as follows. Motivation is in Section 2, proposed model in section 3. A mathematical model is derived in section 4, simulations in section 5 followed by conclusion in section 6. The paper concludes with references.

## II. MOTIVATION

Consider level one cache. Let it be 4-way set associative cache of 1024 sets. Let the cache line size be 32 bytes. Denote S = 1024. Consider the SPEC2000 benchmarks. These are run on this cache with the following placement/replacement policy.

1. Start
2. Let miss = 0 hits = 0 premises = 0
3. Right shift by five bits the address for block size
4. Calculate the set number = a % 1024 and tag number = a div 1024
5. If line is present in set number increment hits, line is accessed update the LRU counter and stop
6. Fetch the line of linesize from main memory to the LRU of the set number, access the line, update LRU counter, increment the miss and stop. If the line size cannot be accommodated in the set number go to step 12
7. If miss – premises > 200 do steps 8-13
8. Invalidate the entire cache entries.
9. Let premises = miss
10. Increment linesize
11. Fetch the linesize data to the setnumber if enough space is available in the set if possible else do step 12.
12. Decrement the linesize, invalidate the entries, go to step 11
13. Repeat the steps 3-11 till end of data
14. Stop

The above algorithm increments the waysize if the number of misses exceeds two hundred, cleans the cache entries and fetches data with new waysize. In case the new waysize exceeds the set capacity to accommodate new lines, the waysize is decremented and the procedure is repeated. The setnumber and tagnumber are calculated using the initial cache parameters. Simulations were done with SPEC2000 benchmars. The baseline is 4-way set associative cache of 32B linesize. The results are shown in Table 1

Table-I: AMAT Comparison

| name | AMAT_p(cycles) | AMAT_t(cycles) |
|---|---|---|
| 256.bzip2 | 13.69521937 | 13.77376461 |
| 181.mcf | 6.90136226 | 7.979818365 |
| 197.parser | 5.386246648 | 8.089264248 |
| 300.twolf | 6.68712608 | 7.551074673 |
| 255.vortex | 4.592063756 | 5.142038851 |
| 175.vpr | 5.249314756 | 5.946893559 |
| AVERAGE | 7.085222144 | 8.080475718 |

As seen from Table1 the AMAT improved by 12.32%. This is the motivation of the paper.

## III. PROPOSED MODEL

Consider level one processor cache. Let it be w-way set associative cache with S sets and line size L. Initially the line size is equal to one way. The following algorithm places line in the cache with varying line size based on the miss rate.
Algorithm Variable Line Size: Given cache with S sets, line size L, for w-way set associative cache, the algorithm adjusts the line size based on miss rate

1. Start
2. Let miss = 0 hits = 0 prev_miss = 0 linesize = L
3. Right shift by log linesize bits the address for block size
4. Calculate the setnumber = a % S and tagnumber = a div S
5. If line of size linesize is present in setnumber increment hits, access the line update the LRU counter and stop
6. Fetch the line of linesize bytes from main memory to the LRU of the setnumber, access the line, update LRU counter, increment the miss and stop. If the linesize cannot be accommodated in the setnumber go to step 12
7. If miss – prev_miss > 200 do steps 8-13
8. Invalidate the entire cache entries.
9. Let prev_miss = miss
10. Increment linesize by one
11. Fetch the linesize data to the setnumber if enough space is available in the set if possible, increment miss, update LRU, access the line else do step 12.
12. Decrement the linesize, invalidate the cache entries, go to step 11
13. Repeat the steps 3-11 till end of data
14. Stop

The algorithm varies the linesize based on miss count. The cache entries are invalidated at each change of linesize value. The algorithm runs for the entire trace. Each access probes for match. If hit, increment the hit counter, access the line, update LRU entry for the line of linesize bytes. Else, fetch line of linesize bytes to the LRU entry of the mapped set. If the line cannot be placed due to lack of space, the linesize is decremented and the process repeated after invalidating the cache entries. If the line can be placed, the miss count is incremented with book keeping of LRU counter. The line size is incremented when the miss count exceeds two hundred from previous linesize change. The number of sets is assumed to be constant for address mapping. The algorithm has the linesize varying from one way to w ways. The proposed model assumes that critical line first is used for line access. The proposed model has line size as m*default line size, m>= 1. The line size need not be power of two. It differs from [7] in that the linesize is multiple of cache ways. There is no need for bound registers to be maintained.
The proposed model requires register to hold current linesize. This is shown in Fig.1

Fig. 1. Proposed architecture

## IV. MATHEMATICAL ANALYSIS OF PROPOSED MODEL

Consider w-way set associative cache with line size L and S sets. Let $C_{prop}$ be proposed model . Let conventional set associative cache be $C_{trad}$ .

Let there be R references. Consider one level cache system. Let there be $h_1$ hits in $C_{trad}$ model. Let the L1 access time be $t_1$ cycles and miss penalty be m cycles. The $C_{trad}$ model AMAT is given by

$$AMAT(C_{trad}) = \frac{1}{R}\left(h_1 t_1 + (R - h_1)m\right) \tag{1}$$

For the proposed model let there be $H_1$ hits in level one cache and miss penalty be M. Let the cache be initialized as per the algorithm variable_line_size cinit times. Let it take $t_{init}$ time to initialize the cache. The AMAT of the proposed system is given by

$$\frac{1}{R}\left(H_1 t_1 + (R - H_1)M + cinit * t_{init}\right) \tag{2}$$

A performance improvement is observed if

$$\frac{1}{R}\left(h_1 t_1 + (R - h_1)m\right) >= \frac{1}{R}\left(H_1 t_1 + (R - H_1)M + cinit * t_{init}\right) \tag{3}$$

Consider the power consumed in the proposed model. Let the maximum number of enabled sets for trace in $C_{prop}$ be propsets and maximum number of enabled sets in $C_{trad}$ be tradsets. Let it take E J/way energy for the cache operation. The energy consumed in $C_{prop}$ is

$$E(C_{prop}) = Ew * propsets \tag{4}$$

For $C_{trad}$ the equation for energy consumed is given by

$$E(C_{trad}) = Ew * tradsets \tag{5}$$

An improvement in Energy consumption is seen if

$$Ew * propsets <= Ew * tradsets \tag{6}$$

## V. SIMULATION

Simplescalar Toolkit is used to simulate the proposed model on SPEC2000 benchmarks. The simulation parameters are shown in Table 2. The proposed is compared with set associative cache with one cache level of same configuration. The results of the simulation with AMAT comparison is shown in Table 1. The proposed model is called proposed and the base set associative cache is called traditional in this paper. As seen from the Table 1 performance improvement of 12% is observed. The miss count is presented in Table 3.

Table-II: Simulation Parameters

| S.No | Parameter | Value |
|---|---|---|
| 1 1 | L1 size of set associaitve cache | 1024 sets |
| 2 2 | L1 associativity of set associative cache | 4 |
| 3 5 | Default Line size | 32 bytes |
| 4 6 | L1 access time in proposed model | 3 cycles |
| 5 8 | Miss penalty in proposed model | 50 cycles |
| 6 | Time to reinitialize cache | 3 cycles |
| 7 | Energy per set | 5J |

Table-III: Miss count comparison

| name | prop | trad |
|---|---|---|
| 256.bzip2 | 106341 | 107124 |
| 181.mcf | 329 | 420 |
| 197.parser | 1647 | 3513 |
| 300.twolf | 354 | 437 |
| 255.vortex | 408 | 549 |
| 175.vpr | 419 | 549 |

The maximum enabled sets for proposed and traditional caches along with improvement in energy consumption is given in Table 4.

Table-IV: Energy Comparison

| name | enabled sets_p | enabled sets_t | E_prop | E_trad | %E |
|---|---|---|---|---|---|
| 256.bzip2 | 1024 | 512 | 20480 | 10240 | -100 |
| 181.mcf | 201 | 389 | 4020 | 7780 | 48.32905 |
| 197.parser | 528 | 512 | 10560 | 10240 | -3.125 |
| 300.twolf | 179 | 394 | 3580 | 7880 | 54.56853 |
| 255.vortex | 193 | 452 | 3860 | 9040 | 57.30088 |
| 175.vpr | 199 | 426 | 3980 | 8520 | 53.28638 |
| | | | average | | 18.39331 |

As seen from Table4 there is average energy improvement of 18% for the chosen parameters.

## VI. CONCLUSION

Processor caches defined by tuple (capacity, assocaiativity, line size) have fixed values at design time. Processor cache model with variable cache line size is proposed in this paper. The proposed model starts with given line size of the cache. The cache address mapping takes the default line size in calculating the set number and tag number. The cache is level one cache system. Based on the miss count the line size is varied as multiple of default line size. If there is no enough space in mapped cache set , the line size is decremented. The cache lines are reinitilaized for each cache line size change. The proposed algorithm is compared with set associative cache of similar configuration. The simulation is done with SPEC2000 benchmarks with Simplescalar Toolkit. A performance improvement of 12% with energy saving of 18% is observed for the chosen parameters.

## ACKNOWLEDGMENT

The author thanks Santa Clara University, CA, USA for providing SPEC2000 benchmarks and Simplescalar Toolkit.

## REFERENCES

1. Alan Jay Smith, "Cache Memories", Computing Surveys, Vol.14, No.3, September 1982, pp. 473-530
2. Alan Jay Smith, “Line (Block) Size Selection in CPU Cache Memories”, *IEEE Trans. Computers*, Vol. C-36, No.9, September, 1987, pp. 1063 -1075.
3. Alexander.V.Veidenbaum, Weiyu Tang and Rajesh Gupta, “Adapting Cache Line Size to Application Behaviour”, *Proceedings of ICS, ’99*
4. David A. Patterson, John L Hennessey: Computer System Architecture : A Quantitative Approach, 3rd edition, Morgan Kaufmann Publishers Inc., 2003, ch. 5
5. S. Subha and Weijia Shang, “Variable Block Size Architecture for Matrix Multiplication”, Proceedings of Obcom 2006, pp. 187-191
6. S. Subha, “Variable Block Size Architecture for Loops”, Proceedings of the Fifth International Conference on Information Technology: New Generations , 2008, pp. 1144-1145
7. S. Subha, “Variable Block Size Architecture for Programs”, Proceedings of ITNG 2009, pp.1640-1641
8. S. Subha, “A Two-Type Data Cache Model”. Proceedings of 2009 IEEE International Conference on Electro/Information Technology, 2009, pp. 476-481.

## AUTHORS PROFILE

S. Subha has done her Ph.D in computer Engineering in processor caches from Santa Clara University, CA, USA. Her research interests are in processor cache memories, computer arithmetic. She has teaching experience of seventeen years, software industry experience of six years. She is presently working in Vellore Institute of Technology, Vellore, India. She has successfully guided /co-guided four research scholars in area of computer architecture, parallel processing, cloud computing at VIT, Vellore. She has authored/co-authored fifty journal papers in international journals, thirty nine international conference publications. She has worked as reviewer of international journals for past five years