On sharing aware Last-Level Cache Replacement Policy

Manish Kumar Bera
Indian Institute of Technology Kanpur
mkbera@iitk.ac.in : 150381

Abhishek Kumar
Indian Institute of Technology Kanpur
abhikr@iitk.ac.in : 150035

ABSTRACT

In this project we look at predicting future sharing of a block in an LLC cache fill. We primarily look at improving on the work done by Natarajan and Chaudhuri in [4]. We also look at the metric which is used to evaluate the performance of the predictor. **This project is done as part of course project for [CS622: Advanced Computer Architecture], under Prof Mainak Chaudhuri (Computer Science and Engineering, Indian Institute of Technology, Kanpur)**

1. INTRODUCTION

Most of the modern day systems have a cache hierarchy consisting of a shared last level cache. Handling last level cache block properly is crucial for the performance because a miss at this level would mean a trip is extremely slow memory. Most of the work on last level cache has focused decreasing the detrimental effect that competing independent threads have by evicting blocks from the working set of other threads. In this project, we wish to explore the possibilities of having a sharing aware LLC. We believe that a replacement policy that takes sharing behaviour into account will perform much better than a policy that is oblivious to sharing pattern. We carry out experiments on PARSEC data-set to bolster our case.

2. RELATED WORK

Natrajan and Chaudhuri [4] in their pioneer work showed how making the last level cache sharing aware can enhance the performance. They classify the filled blocks into three categories namely: shared, noreuse and private reuse. They observe that a majority of hits in LLC are enjoyed by shared cache blocks. Thus, a block management policy that gives priority to shared blocks over others is likely to perform better. The importance of a shared cache blocck also depends on the degree of sharing. They also discuss several ways to quantify sharing awareness. Natrajan and Chaudhuri provide a generic approach to design a sharing aware policy. The oracle policies presented by them highlights the performance gain that can achieved by sharing aware LLC.

SRRIP [2] and DRRIP [1] are two very famous block replacement policies that performed very good on benchmark applications. However, both of them do not take sharing information into account while making replacement decisions. Thus, they fall short of optimal policy significantly.

DOI: 10.1145/1235

3. METHODOLOGY

3.1 Prediction

Block size of 64B is assumed in all three caches. Three kinds of blocks are filled in cache: (a) No reuse fill (b) Private reuse fill (c) Shared fill.

In [4], the authors propose that we use the last w bits of sharing history of an LLC cache fill to predict whether the next LLC cache fill will be shared.

We propose that we also use the sharing history of the neighbours of the block for prediction.

3.2 Evaluation

3.2.1 Predictability index[4]

In [4] the authors use predictability index for evaluating the performance of the predictor. The authors define the predictability index of a shared block at address A for w-bit history as:

$$P_A(w) = \frac{1}{N_w} \sum_h \frac{\max(p_h, s_h)}{p_h + s_h}$$

where the sum is over all w-bit history patterns h captured by the sliding window and N_w is the number of such distinct patterns. They then go on to show the distribution of predictability index among the memory blocks. (Please see Fig 13 in [4])

3.2.2 Credibility of predictability index

We think that plotting this distribution is not a good representation of the performance of the predictor. This is because the metric does not take into consideration the probability distribution of the blocks(A) and history patterns(h). This can be seen in two ways, via examples:

1. Let it be the case that most of the blocks have predictability index around than 0.5 and a few of the blocks have predictability index 0.9. But it might be that the blocks with predictability index 0.5 are required very rarely, and the blocks with predictability

index 0.9 dominate the LLC trace. In this case the predictability index will judge the predictor's performance as bad, even though, we know that the predictor is confident in most of its predictions.

2. Even for a single block, the predictability index normalizes among the different patterns by taking a mean of predictabilities of all the patterns (predictability of a pattern is $\max(p_h, s_h)/(p_h + s_h)$; mean is taken by dividing by number of patterns). But it might be the case that some patterns (even though few in number), which have high predictability, occur more frequently, than the patterns (even though more in number) having low predictability.

3.2.3 Conditional entropy

We think that the relative entropy will be suitable for evaluating performance of a predictor. Relative entropy is defined in the following way[5]:

$$H(Y|X) = -\sum_{x\in\mathcal{X}, y\in\mathcal{Y}} p(x,y)\log p(y|x)$$

For our purpose, of evaluating a predictor, we define X as the sequence of of w bit history and Y as the term that we are trying to predict (the next term in the sharing history sequence).

Information theoretic measures are widely used in sequence prediction evaluation and analysis. [3] is a very nice paper on this, from which we derive our inspiration.

4. EXPERIMENTS

We ran our experiments on following configuration in Table 1

Table 1: configuration 1

cache	size	assoc	replacement policy
L1	32 KB	8 way	LRU
L2	128 KB	8 way	LRU
LLC (offline)	4 MB / 8 MB	16 way	MIN

But we noticed that the trace digested by the L1-L2 combination of upper level cache digests most of the trace, so we also ran on the following configuration in Table 2 (Note that in the below mentioned configuration, everything is same as above, the only difference is that the L2 cache is taken out)

Table 2: configuration 2

cache	size	assoc	replacement policy
L1	32 KB	8 way	LRU
LLC (offline)	4 MB / 8 MB	16 way	MIN

In these experiments, we first run the LLC trace using Belady's MIN algorithm, and then use the sharing history thus produced as an input to our predictor.

First of all we analyze the traits of the trace, similar to what is done in [4]. We tabulate the results in Tables 3, 4, 5, 6, 7 ,8, 9, 10. From these tables, one can observe that shared cache fills contribute a sizable amount towards the total number of cache fills. Also, the shared blocks experience a a significant number of hits out of the total number of cache hits. This reinforces our motive to find a method that exploits sharing attribute of the blocks.

We computed predictability index and conditional entropy for various combination of sequence lengths of block and neighbours. But in this report we present only a few for brevity.

We have tabulated the predictability index and conditional entropy metric for various sequence lengths of blocks and their neighbours in Tables 11, 12, 13, 14, 15, 16, 17, 18. **In the column *number of bits*, the tuple [a,b,c] means (a = number of bits of history of a block), (b = number of bits of previous neighbour, whose address is block-1), (c = number of bits of next neighbour, whose address is block+1)**. When referring to these tables, compare the row of a program with *number of bits* [2,0,0] with [2,1,1] and compare [4,0,0] with [4,2,2] for a better perspective on the difference in performance of prediction strategies.

Note: Instead of computing predictability index for each block separately, we compute the global predictability index by averaging over the global predictability indices of all patterns, which in turn is computed by considering the saring history of all blocks instead of just one.

Note that in most of the cases, the predictability index drops with introduction of bits of neighbouring blocks for prediction. One reason of this drop is that the predictability index is computed by taking the mean over all the patterns, and as the number of bits increases, the number of patterns increases by that many number of exponents. This might be the reason behind the superior 'apparent' performance of (history window size two bits) as compared to (history window size four bits) in [4] as shown in Fig 13 of the paper.

We see that for most of the cases, the conditional entropy decreases when we introduce sharing history bits of neighbouring blocks.

5. CONCLUSION AND FUTURE WORK

It is evident that introducing the extra sharing history bits from neighbouring blocks does enhance the performance of predictor, if conditional entropy is taken as the standard of performance comparison.

Future Work:

1. We need to see how the cache replacement policy performs when we use the predictor on the fly and use these predictions to make decisions regarding cache replacement.

2. We could also do the following: when we are to decide the victim for replacement, we would run MIN algorithm on the trace obtained so far and then give the sharing history thus obtained as the input to the predictor for making predictions. This would help us understand whether we are being able to fully utilize the information obtained till the given moment. If not,

then we can look at developing algorithms/heuristics/architecture to harvest the most out of the information obtained till the given moment.

6. REFERENCES

[1] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, and J. Emer. Adaptive insertion policies for managing shared caches. In *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 208–219, Oct 2008.

[2] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer. High performance cache replacement using re-reference interval prediction (rrip). *SIGARCH Comput. Archit. News*, 38(3):60–71, June 2010.

[3] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001*, pages 130–143, 2001.

[4] R. Natarajan and M. Chaudhuri. Characterizing multi-threaded applications for designing sharing-aware last-level cache replacement policies.

[5] Wikipedia contributors. Conditional entropy — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Conditional_entropy&oldid=868845247, 2018. [Online; accessed 16-November-2018].

7. TABLES

Table 3: LLC metadata: simsmall 4 MB

program	cache fills [shared, private, no-use]	hits [shared, private]	reuse [shared, private]
blackscholes	[2043, 2704, 2820]	[5861, 8328]	[3818, 2804]
bodytrack	[48655, 70023, 9503]	[3125235, 432647]	[3076580, 353121]
canneal	[580317, 1900670, 4184244]	[5667018, 17984915]	[5086701, 11900001]
facesim	[317651, 9344403, 93156899]	[38610846, 179132307]	[38293195, 76631005]
ferret	[106529, 398369, 96629]	[794624, 2761302]	[688095, 2266304]
fluidanimate	[142013, 148412, 1787287]	[1133799, 2418539]	[991786, 482840]
freqmine	[147617, 318385, 1025406]	[2563442, 2618482]	[2415825, 1274691]
raytrace	[101629, 2776597, 7984043]	[340137, 23402254]	[238508, 12641614]
streamcluster	[10874, 1482, 7130]	[7580113, 43631]	[7569239, 35019]
vips	[207393, 1192054, 1574240]	[2684407, 8476687]	[2477014, 5710393]

Table 4: LLC metadata: simmedium 4 MB

program	cache fills [shared, private, no-use]	hits [shared, private]	reuse [shared, private]
blackscholes	[6744, 9905, 3197]	[16130, 23119]	[9386, 10017]
bodytrack	[103327, 86219, 12651]	[10637049, 850272]	[10533722, 751402]
canneal	[2029208, 4141023, 14563082]	[13544676, 42771515]	[11515468, 24067410]
facesim	[317507, 9372050, 93202206]	[37999867, 178922901]	[37682360, 76348645]
ferret	[1078211, 2061530, 2541042]	[4666980, 19571562]	[3588769, 14968990]
fluidanimate	[419379, 605649, 3050798]	[2337002, 4955636]	[1917623, 1299189]
freqmine	[278446, 839089, 2131619]	[5988819, 6619364]	[5710373, 3648656]
raytrace	[198929, 2802461, 8388938]	[617006, 23863334]	[418077, 12671935]
streamcluster	[37573, 1748, 7246]	[66440569, 250741]	[66402996, 241747]
vips	[580254, 4242313, 5194255]	[7158927, 26683245]	[6578673, 17246677]

Table 5: LLC metadata: simsmall 8 MB

program	cache fills [shared, private, no-use]	hits [shared, private]	reuse [shared, private]
blackscholes	[2043, 2704, 2820]	[5861, 8328]	[3818, 2804]
bodytrack	[48657, 70019, 9487]	[3125259, 432623]	[3076602, 353117]
canneal	[370770, 1536798, 2182032]	[7212114, 16439819]	[6841344, 12720989]
facesim	[422637, 18262419, 67502963]	[38781104, 178962049]	[38358467, 93196667]
ferret	[69804, 334144, 36516]	[890548, 2665378]	[820744, 2294718]
fluidanimate	[144106, 77172, 1619898]	[1570775, 1981563]	[1426669, 284493]
freqmine	[224614, 359029, 693115]	[2845279, 2336645]	[2620665, 1284501]
raytrace	[172375, 2822580, 5809939]	[608997, 23133394]	[436622, 14500875]
streamcluster	[10874, 1482, 7130]	[7580113, 43631]	[7569239, 35019]
vips	[149398, 442163, 305579]	[2809265, 8351829]	[2659867, 7604087]

Table 6: LLC metadata: simmedium 8 MB

program	cache fills [shared, private, no-use]	hits [shared, private]	reuse [shared, private]
blackscholes	[6744, 9905, 3197]	[16130, 23119]	[9386, 10017]
bodytrack	[83103, 78488, 9126]	[10667993, 819328]	[10584890, 731714]
canneal	[1737154, 3833363, 9128675]	[16706141, 39610050]	[14968987, 26648012]
facesim	[423772, 18489357, 67415950]	[38319406, 178603362]	[37895634, 92698055]
ferret	[950509, 1717041, 836136]	[6718075, 17520467]	[5767566, 14967290]
fluidanimate	[394367, 502215, 2280094]	[2784508, 4508130]	[2390141, 1725821]
freqmine	[277263, 645849, 1718403]	[6771221, 5836962]	[6493958, 3472710]
raytrace	[244894, 2867376, 6201555]	[886321, 23594019]	[641427, 14525088]
streamcluster	[37573, 1748, 7246]	[66440569, 250741]	[66402996, 241747]
vips	[428004, 1668739, 1393702]	[7512007, 26330165]	[7084003, 23267724]

Table 7: L2 metadata: simsmall 4 MB

program	cache fills [shared, private, no-use]	hits [shared, private]	reuse [shared, private]
blackscholes	[2043, 3144, 2380]	[204796, 29879]	[202753, 24355]
bodytrack	[48655, 72883, 6643]	[8620691, 752466]	[8572036, 672940]
canneal	[580358, 1949238, 4136248]	[5956232, 21710986]	[5375874, 15625500]
facesim	[315045, 10381103, 92138037]	[44891052, 199111621]	[44576007, 96592481]
ferret	[112310, 414632, 89798]	[1362031, 9822177]	[1249721, 9317747]
fluidanimate	[141937, 152857, 1783176]	[1372898, 2561434]	[1230961, 625401]
freqmine	[148185, 417877, 928457]	[4336701, 4589411]	[4188516, 3243077]
raytrace	[102966, 2902074, 7860298]	[455422, 28923175]	[352456, 18160803]
streamcluster	[10874, 2426, 6186]	[18833826, 127682]	[18822952, 119070]
vips	[212004, 2634441, 281537]	[5375648, 12937948]	[5163644, 10021970]

Table 8: L2 metadata: simmedium 4 MB

program	cache fills [shared, private, no-use]	hits [shared, private]	reuse [shared, private]
blackscholes	[6744, 10730, 2372]	[832656, 112923]	[825912, 99821]
bodytrack	[103327, 89115, 9755]	[30983861, 2150261]	[30880534, 2051391]
canneal	[2029195, 4290935, 14415225]	[13952317, 50489010]	[11923122, 31782850]
facesim	[315333, 10431856, 92158957]	[44398159, 199853196]	[44082826, 97262383]
ferret	[1061076, 2158696, 2580105]	[6832943, 42406573]	[5771867, 37667772]
fluidanimate	[420173, 666967, 2993904]	[2707671, 5301217]	[2287498, 1640346]
freqmine	[282963, 1038141, 1956489]	[7776199, 12807670]	[7493236, 9813040]
raytrace	[202018, 3004350, 8186463]	[885354, 29410212]	[683336, 18219399]
streamcluster	[37573, 2631, 6363]	[67098070, 323154]	[67060497, 314160]
vips	[584578, 9184117, 750695]	[20923338, 39529378]	[20338760, 29594566]

Table 9: L2 metadata: simsmall 8 MB

program	cache fills [shared, private, no-use]	hits [shared, private]	reuse [shared, private]
blackscholes	[2043, 3144, 2380]	[204796, 29879]	[202753, 24355]
bodytrack	[48657, 72879, 6627]	[8620721, 752436]	[8572064, 672930]
canneal	[370770, 1564519, 2154442]	[7548699, 20118519]	[7177929, 16399558]
facesim	[422218, 19122234, 66653547]	[45120652, 198882021]	[44698434, 113106240]
ferret	[69990, 340013, 31151]	[1663164, 9521044]	[1593174, 9149880]
fluidanimate	[144036, 78211, 1618941]	[1870196, 2064136]	[1726160, 366984]
freqmine	[225009, 400150, 652414]	[4823384, 4102728]	[4598375, 3050164]
raytrace	[172499, 2902207, 5731977]	[769855, 28608742]	[597356, 19974558]
streamcluster	[10874, 2426, 6186]	[18833826, 127682]	[18822952, 119070]
vips	[150905, 667371, 103079]	[5530925, 12782671]	[5380020, 12012221]

Table 10: L2 metadata: simmedium 8 MB

program	cache fills [shared, private, no-use]	hits [shared, private]	reuse [shared, private]
blackscholes	[6744, 10730, 2372]	[832656, 112923]	[825912, 99821]
bodytrack	[83103, 81042, 6572]	[31147443, 1986679]	[31064340, 1899065]
canneal	[1737240, 3934240, 9028234]	[17173366, 47267961]	[15436126, 34305487]
facesim	[423327, 19372338, 66543331]	[44795774, 199455581]	[44372447, 113539912]
ferret	[960507, 1759131, 802177]	[9596039, 39643477]	[8635532, 37082169]
fluidanimate	[394359, 519239, 2263589]	[3216152, 4792736]	[2821793, 2009908]
freqmine	[278671, 788100, 1580509]	[9019162, 11564707]	[8740491, 9196098]
raytrace	[248457, 3019449, 6047176]	[1212319, 29083247]	[963862, 20016622]
streamcluster	[37573, 2631, 6363]	[67098070, 323154]	[67060497, 314160]
vips	[432472, 2839705, 374697]	[21350381, 39102335]	[20917909, 35887933]

Table 11: LLC analysis: simsmall 4 MB

program	number of bits [block, prev block, next block]	predictability index	conditional entropy
canneal	[2, 0, 0]	0.689631	0.331978
canneal	[4, 0, 0]	0.680574	0.327954
canneal	[2, 1, 1]	0.640371	0.303111
canneal	[4, 2, 2]	0.642248	0.316302
facesim	[2, 0, 0]	0.773624	0.02215
facesim	[4, 0, 0]	0.78931	0.021716
facesim	[2, 1, 1]	0.698446	0.011267
facesim	[4, 2, 2]	0.775507	0.006083
ferret	[2, 0, 0]	0.633627	0.58462
ferret	[4, 0, 0]	0.624666	0.581741
ferret	[2, 1, 1]	0.607519	0.816428
ferret	[4, 2, 2]	0.686672	0.789844
fluidanimate	[2, 0, 0]	0.829489	0.310176
fluidanimate	[4, 0, 0]	0.763312	0.308643
fluidanimate	[2, 1, 1]	0.798999	0.484697
fluidanimate	[4, 2, 2]	0.725215	0.512937
raytrace	[2, 0, 0]	0.793261	0.073887
raytrace	[4, 0, 0]	0.198315	0.073887
raytrace	[2, 1, 1]	0.679592	0.059547
raytrace	[4, 2, 2]	0.162106	0.067871
vips	[2, 0, 0]	0.877544	0.155809
vips	[4, 0, 0]	0.741857	0.153638
vips	[2, 1, 1]	0.800953	0.030054
vips	[4, 2, 2]	0.682572	0.021971

Table 12: LLC analysis: simmedium 4 MB

program	number of bits [block, prev block, next block]	predictability index	conditional entropy
canneal	[2, 0, 0]	0.696516	0.375507
canneal	[4, 0, 0]	0.651959	0.35048
canneal	[2, 1, 1]	0.627627	0.354415
canneal	[4, 2, 2]	0.606447	0.359942
facesim	[2, 0, 0]	0.779195	0.0219
facesim	[4, 0, 0]	0.806613	0.021489
facesim	[2, 1, 1]	0.703973	0.011086
facesim	[4, 2, 2]	0.783752	0.006659
ferret	[2, 0, 0]	0.705395	0.659915
ferret	[4, 0, 0]	0.657466	0.635745
ferret	[2, 1, 1]	0.645464	0.605543
ferret	[4, 2, 2]	0.630262	0.610399
fluidanimate	[2, 0, 0]	0.75858	0.363757
fluidanimate	[4, 0, 0]	0.780489	0.34983
fluidanimate	[2, 1, 1]	0.788236	0.414723
fluidanimate	[4, 2, 2]	0.731172	0.404254
raytrace	[2, 0, 0]	0.799244	0.102019
raytrace	[4, 0, 0]	0.199811	0.102019
raytrace	[2, 1, 1]	0.73302	0.090886
raytrace	[4, 2, 2]	0.178429	0.106951
vips	[2, 0, 0]	0.837441	0.112379
vips	[4, 0, 0]	0.789079	0.108886
vips	[2, 1, 1]	0.825789	0.038603
vips	[4, 2, 2]	0.733714	0.021359

Table 13: LLC analysis: simsmall 8 MB

program	number of bits [block, prev block, next block]	predictability index	conditional entropy
canneal	[2, 0, 0]	0.830345	0.416159
canneal	[4, 0, 0]	0.884673	0.414731
canneal	[2, 1, 1]	0.748565	0.3891
canneal	[4, 2, 2]	0.831261	0.390299
facesim	[2, 0, 0]	0.811987	0.034657
facesim	[4, 0, 0]	0.81649	0.034174
facesim	[2, 1, 1]	0.728059	0.01558
facesim	[4, 2, 2]	0.798771	0.008234
ferret	[2, 0, 0]	0.710305	0.62609
ferret	[4, 0, 0]	0.590914	0.626071
ferret	[2, 1, 1]	0.736119	0.532955
ferret	[4, 2, 2]	0.048473	0.284769
fluidanimate	[2, 0, 0]	0.825114	0.374179
fluidanimate	[4, 0, 0]	0.393716	0.37395
fluidanimate	[2, 1, 1]	0.81409	0.351587
fluidanimate	[4, 2, 2]	0.035156	0.0
raytrace	[2, 0, 0]	0.827661	0.138249
raytrace	[4, 0, 0]	0.206915	0.138249
raytrace	[2, 1, 1]	0.722488	0.115948
raytrace	[4, 2, 2]	0.18605	0.129382
vips	[2, 0, 0]	0.853184	0.396437
vips	[4, 0, 0]	0.780772	0.395415
vips	[2, 1, 1]	0.797441	0.088336
vips	[4, 2, 2]	0.737012	0.043798

Table 14: LLC analysis: simmedium 8 MB

program	number of bits [block, prev block, next block]	predictability index	conditional entropy
canneal	[2, 0, 0]	0.684547	0.395502
canneal	[4, 0, 0]	0.658945	0.38667
canneal	[2, 1, 1]	0.639918	0.365579
canneal	[4, 2, 2]	0.634998	0.385388
facesim	[2, 0, 0]	0.814465	0.034509
facesim	[4, 0, 0]	0.815762	0.034077
facesim	[2, 1, 1]	0.728645	0.015603
facesim	[4, 2, 2]	0.799575	0.008549
ferret	[2, 0, 0]	0.646784	0.693125
ferret	[4, 0, 0]	0.589632	0.665268
ferret	[2, 1, 1]	0.617247	0.544101
ferret	[4, 2, 2]	0.59473	0.51976
fluidanimate	[2, 0, 0]	0.805207	0.462598
fluidanimate	[4, 0, 0]	0.773062	0.45533
fluidanimate	[2, 1, 1]	0.812886	0.586915
fluidanimate	[4, 2, 2]	0.607753	0.646521
raytrace	[2, 0, 0]	0.784646	0.156643
raytrace	[4, 0, 0]	0.196161	0.156643
raytrace	[2, 1, 1]	0.729985	0.148497
raytrace	[4, 2, 2]	0.177029	0.179207
vips	[2, 0, 0]	0.843366	0.253629
vips	[4, 0, 0]	0.795252	0.249593
vips	[2, 1, 1]	0.820361	0.083084
vips	[4, 2, 2]	0.76404	0.043255

Table 15: L2 analysis: simsmall 4 MB

program	number of bits [block, prev block, next block]	predictability index	conditional entropy
canneal	[2, 0, 0]	0.68968	0.331963
canneal	[4, 0, 0]	0.68049	0.327933
canneal	[2, 1, 1]	0.6405	0.303111
canneal	[4, 2, 2]	0.642274	0.316331
facesim	[2, 0, 0]	0.773543	0.022085
facesim	[4, 0, 0]	0.788652	0.021651
facesim	[2, 1, 1]	0.699434	0.011219
facesim	[4, 2, 2]	0.774649	0.006097
ferret	[2, 0, 0]	0.623312	0.593605
ferret	[4, 0, 0]	0.609941	0.589456
ferret	[2, 1, 1]	0.607502	0.834089
ferret	[4, 2, 2]	0.647122	0.864934
fluidanimate	[2, 0, 0]	0.828333	0.310027
fluidanimate	[4, 0, 0]	0.769516	0.308479
fluidanimate	[2, 1, 1]	0.796737	0.485042
fluidanimate	[4, 2, 2]	0.757191	0.511904
raytrace	[2, 0, 0]	0.799693	0.074689
raytrace	[4, 0, 0]	0.199923	0.074689
raytrace	[2, 1, 1]	0.687486	0.061329
raytrace	[4, 2, 2]	0.161452	0.069781
vips	[2, 0, 0]	0.879329	0.150277
vips	[4, 0, 0]	0.743413	0.148027
vips	[2, 1, 1]	0.802402	0.029667
vips	[4, 2, 2]	0.684163	0.021916

Table 16: L2 analysis: simmedium 4 MB

program	number of bits [block, prev block, next block]	predictability index	conditional entropy
canneal	[2, 0, 0]	0.696543	0.375489
canneal	[4, 0, 0]	0.651958	0.350472
canneal	[2, 1, 1]	0.627625	0.354413
canneal	[4, 2, 2]	0.606457	0.359948
facesim	[2, 0, 0]	0.778977	0.021847
facesim	[4, 0, 0]	0.806033	0.021434
facesim	[2, 1, 1]	0.705336	0.011041
facesim	[4, 2, 2]	0.785411	0.006658
ferret	[2, 0, 0]	0.717129	0.650035
ferret	[4, 0, 0]	0.671006	0.6284
ferret	[2, 1, 1]	0.653094	0.599165
ferret	[4, 2, 2]	0.63834	0.606235
fluidanimate	[2, 0, 0]	0.756241	0.364454
fluidanimate	[4, 0, 0]	0.782043	0.349895
fluidanimate	[2, 1, 1]	0.789853	0.416294
fluidanimate	[4, 2, 2]	0.717615	0.400669
raytrace	[2, 0, 0]	0.795398	0.103725
raytrace	[4, 0, 0]	0.198849	0.103725
raytrace	[2, 1, 1]	0.733529	0.092291
raytrace	[4, 2, 2]	0.177901	0.107618
vips	[2, 0, 0]	0.832239	0.108583
vips	[4, 0, 0]	0.785174	0.105093
vips	[2, 1, 1]	0.826344	0.037642
vips	[4, 2, 2]	0.735392	0.021124

Table 17: L2 analysis: simsmall 8 MB

program	number of bits [block, prev block, next block]	predictability index	conditional entropy
canneal	[2, 0, 0]	0.830332	0.416136
canneal	[4, 0, 0]	0.884715	0.414709
canneal	[2, 1, 1]	0.748528	0.389054
canneal	[4, 2, 2]	0.831407	0.390235
facesim	[2, 0, 0]	0.811977	0.034672
facesim	[4, 0, 0]	0.817	0.034182
facesim	[2, 1, 1]	0.727298	0.015595
facesim	[4, 2, 2]	0.796772	0.008232
ferret	[2, 0, 0]	0.678921	0.625687
ferret	[4, 0, 0]	0.589448	0.625677
ferret	[2, 1, 1]	0.753761	0.534414
ferret	[4, 2, 2]	0.057552	0.399623
fluidanimate	[2, 0, 0]	0.825018	0.374072
fluidanimate	[4, 0, 0]	0.393692	0.373843
fluidanimate	[2, 1, 1]	0.8145	0.351273
fluidanimate	[4, 2, 2]	0.035156	0.0
raytrace	[2, 0, 0]	0.834226	0.138323
raytrace	[4, 0, 0]	0.208556	0.138323
raytrace	[2, 1, 1]	0.731033	0.116514
raytrace	[4, 2, 2]	0.176453	0.129876
vips	[2, 0, 0]	0.852874	0.388841
vips	[4, 0, 0]	0.779379	0.387747
vips	[2, 1, 1]	0.800545	0.085816
vips	[4, 2, 2]	0.735902	0.043509

Table 18: L2 analysis: simmedium 8 MB

program	number of bits [block, prev block, next block]	predictability index	conditional entropy
canneal	[2, 0, 0]	0.684554	0.395487
canneal	[4, 0, 0]	0.658943	0.38666
canneal	[2, 1, 1]	0.63992	0.365565
canneal	[4, 2, 2]	0.63504	0.385372
facesim	[2, 0, 0]	0.814326	0.03453
facesim	[4, 0, 0]	0.815871	0.034089
facesim	[2, 1, 1]	0.728	0.015617
facesim	[4, 2, 2]	0.799131	0.008562
ferret	[2, 0, 0]	0.641933	0.69963
ferret	[4, 0, 0]	0.58629	0.671355
ferret	[2, 1, 1]	0.615583	0.556171
ferret	[4, 2, 2]	0.593688	0.53317
fluidanimate	[2, 0, 0]	0.804956	0.462674
fluidanimate	[4, 0, 0]	0.777418	0.455374
fluidanimate	[2, 1, 1]	0.812996	0.58732
fluidanimate	[4, 2, 2]	0.597506	0.647749
raytrace	[2, 0, 0]	0.789609	0.158564
raytrace	[4, 0, 0]	0.197402	0.158564
raytrace	[2, 1, 1]	0.731353	0.150747
raytrace	[4, 2, 2]	0.176196	0.18068
vips	[2, 0, 0]	0.838602	0.246177
vips	[4, 0, 0]	0.793719	0.242138
vips	[2, 1, 1]	0.819372	0.080797
vips	[4, 2, 2]	0.77263	0.042393