A Novel Pair of Replacement Algorithms on L1 and L2 Cache for FFT

Ms. Richa Gupta
Reader, Computer Science and Engineering
K C Bansal Technical Academy, Indore, India
richa_tg@rediffmail.com

Dr. Sanjiv Tokekar
Prof. & Head, Electronics and Telecom. Engg.
Institute of Engineering and Technology, DAVV
Indore, India
sanjivtokekar@yahoo.com

***Abstract*— Processors speed is much faster than memory; to bridge this gap cache memory is used. This paper proposes a preeminent pair of replacement algorithms for Level 1 cache (L1) and Level 2 cache (L2) respectively for the Fast Fourier Transform (FFT). The access patterns of L1 and L2 are different, when CPU does not get the desired data in L1 then it refers to L2. Thus the replacement algorithm which works efficiently for L1 may not be efficient for L2. With the memory access pattern of FFT, the paper has simulated and analyzed the behavior of various existing replacement algorithms on L1 and L2 respectively. The replacement algorithms which are taken into consideration are: Least Recently Used (LRU), Least Frequently Used (LFU) and First In First Out (FIFO).**

This paper has also proposed new replacement algorithms for L1 (FFTNEW1) and for L2 (FFTNEW2) respectively for the same application. Simulation results shows that by applying the proposed pair of replacement algorithms miss rates are considerably reduced.

Key words: Level 1 Cache (L1), Level 2 Cache (L2), Replacement Algorithms, Access Pattern, Fast Fourier Transform (FFT).

I. INTRODUCTION

To bridge the speed gap between main memory and processor, cache memory is used. Cache memory works with the principle of locality. The principle of locality refers that CPU does not requires all the code/data at a time. The principle of locality can be spatial or temporal [1, 2, 3].

Whenever a block is requested from CPU, first of all it is searched on L1 if the required page is found in L1 it is a hit else a miss. When L1 is saturated and it is a miss then a block from L1 is to be evicted to create a space for the required page. Various replacement algorithms, such as LRU, FIFO, LFU [4, 5] etc are used to select the victim page. L1 is having better temporal locality than L2, as L2 is accessed when a miss occurs on L1. The pages which are often used are likely to be used again i.e. hot pages should remain in L1 and cold pages should be taken off and are placed in the main memory. Whenever a page is evicted from L1 it will be placed on L2. The nature of pages which reside on L2 should be neither too cold nor too hot i.e. moderate. Place for hot pages is L1 and that for cold pages is in the main memory [3]. Most of the algorithm tries to keep hot pages in the cache but from the above discussion it is clear that this is not the requirement for L2 cache. Thus the replacement algorithm which is suitable for L1 may not be suitable for L2.

Furthermore various algorithms and applications such as Matrix Multiplication, Fast Fourier transform, Networks, Databases etc., will have varying accesses to the memory thus resulting in varying principal of locality. Thus same replacement algorithm may not be suitable for various applications and algorithms, as different applications may have different access patterns.

Replacement algorithms on L1 & L2

Initially if any block is referenced then it will suffer with compulsory miss. If a reference suffers a miss because of saturated cache, then replacement algorithm will evict a block. The replacement algorithms are based on some criteria as mentioned earlier may be recency, frequency etc. The replacement algorithms which are taken into account are Least Recently Used (LRU), First in First out (FIFO), Least Frequently Used (LFU).

Analyzing the access pattern of L2 we have discussed that L2 is having poor temporal locality as compared to L1. If the required block is not in L1 it is searched in L2 and then in main memory. It means that L2 will also suffer with initial misses. L2 is larger than L1, so after the initial misses the probability of the data to remain in L2 is high.

For analyzing the behavior of various replacement algorithms on L2, size of L1 is fixed and size of L2 is varied from double the size of L1. Replacement algorithm used on L1 is LRU, while on L2 the

replacement algorithms LRU, FIFO, LFU are applied to analyze the behavior of these algorithms on L2.

Literature Review

Various studies have been done for the cache replacement algorithms for single level cache. Basically the replacement algorithms can be divided in three criterions of recency, frequency and mixture of both recency and frequency. The recency based algorithms are such as Least Recently Used (LRU) [4,6], Least Recently Used k references (LRU-k) [7], this approach basically keeps track of the times of last k references of the popular page, with this information it statically estimates inter arrival time of references of the page. Most Recently Used (MRU) [4, 6] etc. The frequency based algorithms are such as Least Frequently Used (LFU) [4, 6], least Frequently Used k references (LFU-k) [8], this algorithm is basically improvement of LFU, in this amount of references is calculated with some parameters such as speed, acceleration, Frequency Based Algorithm (FBR) [9] this algorithm selects the victim using combination of reference frequency and block age. Algorithms using both the criterion of recency and frequency are such as Least Frequently Recently Used (LFRU) [10], this algorithm decides, how much more weightage be given to recent history than to older history, Low Inter-reference Recency (LIRs) [11], this algorithm evaluates Inter Reference Recency (IRR) for deciding the victim, 2Q is an improvement of LRU and LRU-2, which solves correlated reference problem [12], Second Chance Frequency- Least Recently Used (SF-LRU) [13], it uses both the basic algorithms LRU and LFU with the concept of second chance and calculates recency frequency control value to decide the victim. etc.

Much work has not been done for second level cache replacement algorithm. A comprehensive study of second level cache management was given by Zhou *et al.*, which emphasizes that access pattern of second level cache is different than first level. More specifically it presents a new algorithm Multi queue (MQ) to effectively manage second-level buffer caches. It was evaluated on nine-replacement algorithms, which were basically designed for single level [3]. Michael *et al*. proposes a policy "Karma" which uses application hints to partition the cache and to manage each range of blocks with the policy best suited for its access pattern [14].

Wayne *et al.* shows that an opportunity exists to close the gap between Optimality (OPT) and the LRU algorithms, they present a replacement algorithm based on the detection of temporal locality in the L2 cache, the block to be taken out is chosen by considering both its priority in the LRU stack and whether it exhibits temporal locality or not [15]. Nikolas *et al* has given the concept of cache conscious hash table, which shows improvement both in time and space [16]. Aleksandar *et al*. illustrates performance evaluation of cache design issues such as cache size and organization, block size and replacement policy in embedded processor. It suggests suitable replacement policy for data cache, instruction cache and unified cache [17].

We are focusing our research on suitable replacement algorithm for FFT. James *et al* proposes an algorithm for FFT which maximizes the use on in-cache operations [18]. Bevan suggests that cache improves the effectiveness, if the memory access pattern exhibits sufficient locality. It has proposed energy efficient, single chip 1024-point FFT Processor. It has used FFT algorithm which offers good locality over large span of computations [19]. Sakr *et al* works on learning and predicting memory access Patterns of FFT with the help of Time Delay Nueral Network (TDNN), which in turn reduces number of faults [20]. M. Frigo *et al* has given optimal algorithms for matrix transpose, FFT with multiple levels of caching [21]. Naga K. Govindaraju *et al* cache efficient algorithms for scientific computation using Graphics Processing Units (GPUs). It exploits data parallelism and high memory bandwidth in GPU. It has worked for sorting, FFT, Matrix Multiplcation and shows that FFT algorithm is able to achieve 10* performance improvement over Xeon or Opteron processors [22]. Jizhu Lu *et al* have described various implementation issues and have compared the performance with state – of-the-art FFT implementations on Intel and AMD with Cell BE Processor [23]. Bushra *et al*. suggests replacement policy should take into account cache performance as well as traffic generated by the cache. The author focuses on several dynamic replacement policies with the motivation to reduce the traffic last level of cache to main memory, while not increasing the number of misses [24]. Jinwoo et *al* focuses on microprocessor architecture that implement processor-in memory, stream processing and tiled processing. It has described about Coherent side-lobe canceller (CLSC), in which most of the computation time is spent on FFT and IFFT operations, thus for performance improvement appropriate FFT algorithm has been used for each architecture [25].

Based on the above study our goal is to work on pair of replacement policies which works efficiently on L1 and L2 cache for FFT memory access pattern.

Based on above discussion it works on the following criterion:

(i) To find out the performance of different replacement algorithms on L1 among existing algorithms taken into consideration (LRU, LFU, FIFO).

Finding out the preeminent pair of replacement algorithms on L1 and L2 respectively among

(ii) existing algorithms taken into consideration (LRU, LFU, FIFO).

(iii) This paper also proposes pair of new replacement algorithms for L1 (FFTNEW1) and for L2 (FFTNEW2).

II. MATERIALS AND METHODS

1. Analysis of Replacement algorithms on L1

To analyze the behavior of the replacement algorithms mentioned above; reference string of FFT is generated. For simulation a model of 128 point Fast Fourier transform (FFT) is taken. For each replacement algorithm, Miss Rate is calculated for varying size of L1.

With the variation of size of FFT the required maximum size of L1 will vary. Considering N point FFT N references will be for the input and N/2 references will be for various constants, thus total different references will be N+N/2. Thus there will be at least N+N/2 initial misses. If the cache size is N+N/2 then it will not suffer with any other misses. In the analysis the cache size is varied till N+N/2 i.e. up to 192 as we are considering 128 point FFT.

2. Analysis of Replacement Algorithm on L1 and L2

Similarly for analyzing the behavior of mentioned replacement algorithm on L2 the same setup is used as in L1. Here the L1 is fixed to $1/4^{th}$ (32) of the required memory (192) and L2 is varied from 64 till N+N/2.

3. Proposed Replacement Algorithms on L1 and L2

After analyzing the access pattern of FFT and comparing all the results, new replacement algorithms for L1 and L2 are developed.

Considering the calculation of FFT it can be realized that while calculation constant W0 is used maximum number of times. As W0 =1, thus while accessing we are not accessing the location of W0.

Along with this, we have realized, with the previous analysis that for lower cache size LRU is performing better and for higher cache size FIFO is performing far better than LRU and LFU. After analyzing this behavior for various point FFT's, we realized a common criterion. For example if the FFT is 128 than up to cache size 128 LRU will perform better and after that FIFO's performance is better; for FFT 64 up to 64 LRU is better and after that FIFO is better; and so on . Thus for generalization we had taken that if the FFT point is N than up to N LRU better and after that FIFO's performance is better. On behalf of this criterion we developed a new replacement algorithm NEW1FFT for This replacement algorithm takes benefit of both LRU and FIFO for N point FFT, till cache size N it behaves like LRU and after that it behaves like FIFO.

Same kind of behavior was also recognized for L2 and based on the same criterion another replacement algorithm FFTNEW2 has been developed.

4. Fixing Cache size and varying the FFT Size

An additional criterion for the analysis is taken as fixing the size of L1 and L2 and varying the FFT size and count the number of misses. Here we are fixing the size of L1 to 32, L2 to 128 and varying the FFT point from 8 to 1024.

III. RESULTS

1. Performance Analysis of FFTNEW1 on L1

The results of various replacement algorithms on L1 are as shown in Fig. 1, which gives the miss rates for different cache sizes.

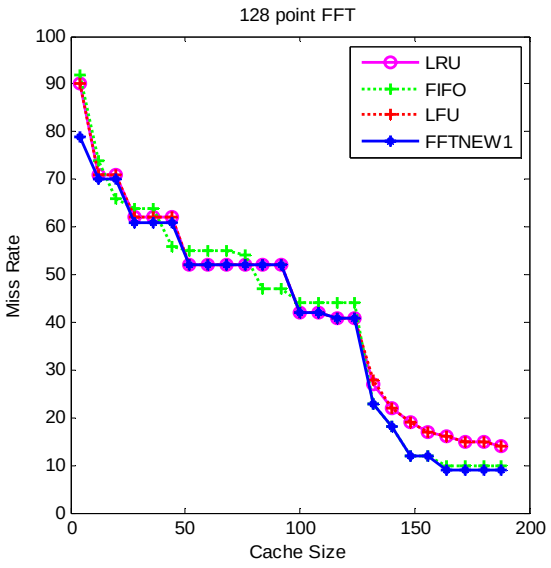

Figure-1. Comparison of Proposed Algorithm FFTNEW1 on L1.

2. Performance Analysis of Pair of Replacement Algorithms on L1 and L2

The following pairs were taken into consideration. As the performance of the replacement algorithm LRU and LFU performance is same thus the pair with LFU is not represented.

CASE I: Replacement algorithm used on L1 is LRU, while on L2 the replacement algorithms LRU, FIFO,

LFU are applied to analyze the behavior of these algorithms on L2. The results are as shown in Fig.2.

CASE II: Another analysis has been done by applying FIFO on L1 and LRU, FIFO, and LFU on L2 under the same criterion. The results are as shown in Fig. 3.

CASE III: In this analysis the proposed replacement algorithm FFTNEW1 is applied on L1 followed by LRU, LFU, and FIFO on L2. The results are as shown in Fig. 4.

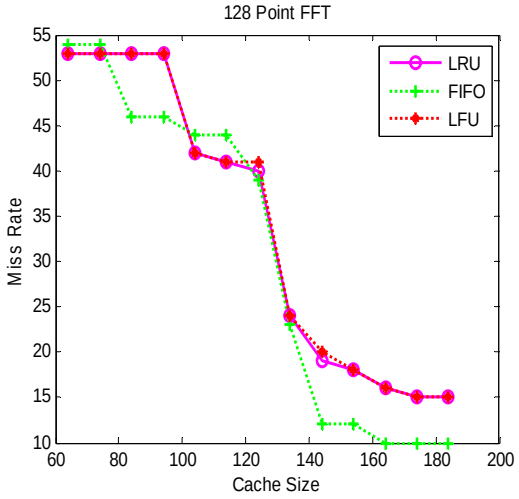

Figure 2. (L1 Size: 32, L2 Varied from 64) L1-LRU

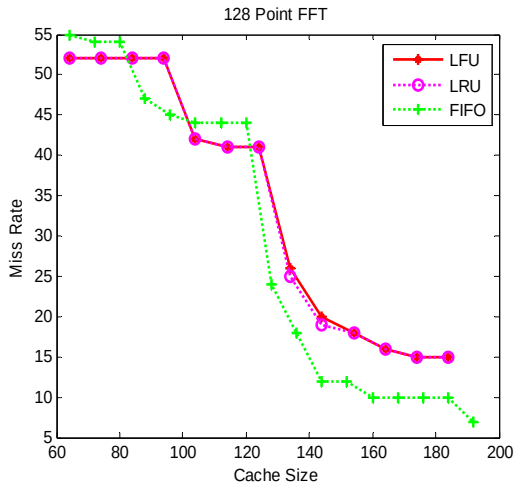

Figure 3. (L1 Size: 32, L2 Varied from 64) L1- FIFO

3. *Analysis by fixing cache size and varying FFT Size*

The comparison of the proposed algorithm FFTNEW1 with others is as shown in Fig. 5. for varying FFT Size Fig.6 compares the various pair of replacement algorithms FFTNEW1 on L1 followed by LRU, LFU, FIFO and FFTNEW2 on L2.

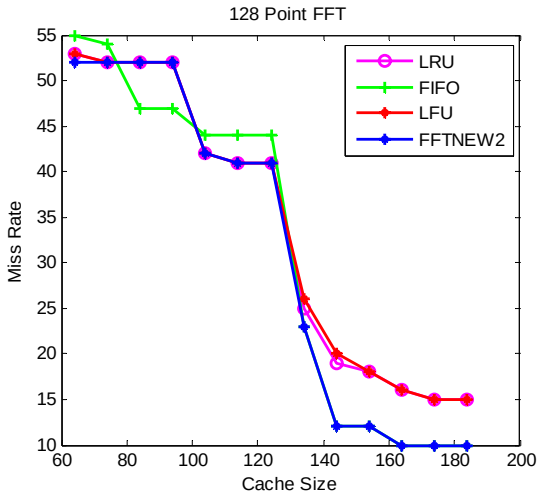

Figure 4. (L1 Size: 32, L2 Varied from 64) L1- FFTNEW1

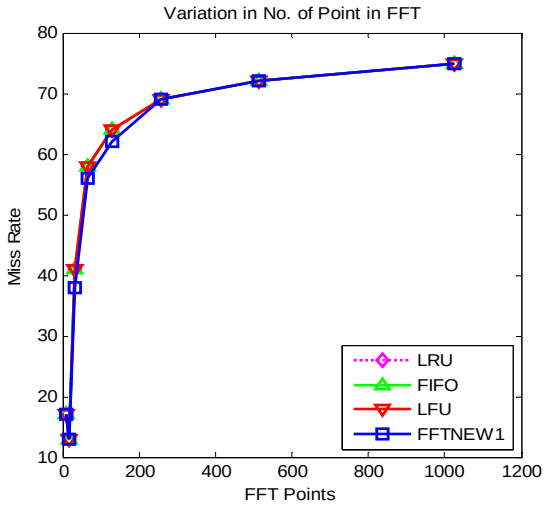

Figure 5. Comparison of the replacement algorithms LRU, LFU, FIFO with FFTNEW1 on L1 for varying FFT size

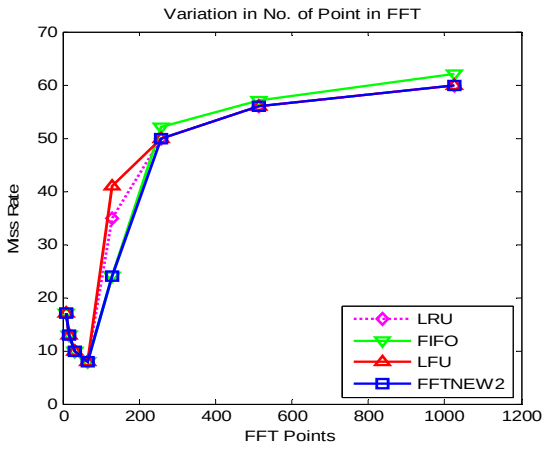

Figure 6. Comparison of the proposed pair FFTNEW1 and FFTNEW2 with other pair of replacement algorithms on L1 and L2

IV. DISCUSSION

1. Analysis of Replacement algorithms on L1

The result is as shown in Fig.1. From this figure it can be concluded that for FFT, on L2, the performance of LRU and LFU replacement policies are same.

It can also be realized that among existing algorithms LRU is performing better while for larger cache size FIFO is performing better.

Further more simulation result shows that the proposed algorithms performance is better than existing ones as shown in Table 1.

TABLE 1 : COMPARISION OF PROPOSED REPLACEMENT ALGORITHM FFTNEW1 WITH OTHERS

L1 Cache size / *ALGO*	*4*	*44*	*84*	*124*	*164*	*184*	*192*
LRU	90	62	52	41	16	15	7
FIFO	92	56	47	44	10	10	7
LFU	90	62	52	41	16	15	7
FFTNEW1	79	61	52	41	9	9	7

2. Performance Analysis of Pair of Replacement Algorithms on L1 and L2

Our motivation is to find out pair of replacement algorithms for L1 and L2, which is efficient for FFT access pattern.

CASE I : In this case we have done comparison for the pair LRU-LRU; LRU-LFU; LRU-FIFO. With the help of Fig 2 it can be realized that the pair LRU-LRU is performing better for lower cache size while the pair LRU-FIFO is giving better results for higher cache size.

CASE II : In this we have done comparison for the pair FIFO-LRU; FIFO-LFU; FIFO-FIFO. The results are shown in Fig. 3. The same kind of result are obtained in this case too i.e. the pair FIFO-LRU is performing better for lower cache size while the pair FIFO-FIFO is giving better results for higher cache size.

CASE III : In this the proposed algorithm FFTNEW1 is applied on L1. Here we are comparing the algorithm FFTNEW1 on L1 followed by LRU, LFU, FIFO and FFTNEW2 on L2. The results for these pairs are as shown in Fig. 4.. From the result it can be realized that the overall proposed pair FFTNEW1 on L1 and FFTNEW2 on L2 is performing better than other pairs. Finally the results in tabulation form of all the pairs considered are as shown in Table 2. From theTable2 it can be seen that overall for almost all the cache size the proposed pair is performing better.

TABLE 2 : COMPARISION OF PROPOSED PAIR OF ALGORITHMS WITH OTHER PAIRS

L1 = 32, L2 Varied from 64						
CACHE SIZE / ALGORITHM		**64**	**104**	**144**	**184**	**192**
L1	**L2**					
LRU	**LRU**	**53**	**42**	**19**	**15**	**7**
FIFO	**LRU**	**52**	**42**	**19**	**15**	**7**
NEWFFT1	**LRU**	**52**	**42**	**12**	**10**	**7**
LRU	**FIFO**	**54**	**44**	**12**	**10**	**7**
FIFO	**FIFO**	**55**	**44**	**12**	**10**	**7**
NEWFFT1	**FIFO**	**55**	**42**	**12**	**10**	**7**
LRU	**LFU**	**53**	**42**	**20**	**15**	**7**
FIFO	**LFU**	**52**	**42**	**20**	**15**	**7**
NEWFFT1	**LFU**	**53**	**42**	**20**	**15**	**7**
NEWFFT1	**NEWFFT2**	**52**	**42**	**12**	**10**	**7**

3. Analysis by fixing cache size and varying FFT Size

This analysis is depicted with the help of Fig.5 which compares the performance of FFTNEW1 on L1. From the Fig. 5 it can be realized that for very small and very large cache size performance of all the algorithms is almost same, while for moderate cache size of L1 FFTNNEW1 is performing better than others.

Fig. 6 analyzes the behavior of replacement algorithms on L2. Here it can be analyzed that overall the performance of the Pair FFTNEW1-FFTNEW2 is better than other pairs considered in this paper.

REFERENCES

[1] John L Hennessy, David A Patterson, *"Computer Architecture: A Quantitative Approach"* 2nd edition, 1996.

[2] Kai Hwang, "Advanced Computer Architecture: *Parallelism, Scalability, Programmability"*, 1st edition**,** 1992.

[3] Y.Zhou, Z.Chen and K.Li, *"Second Level Buffer cache Management"*, IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol.15, No. 7, pp.505-519, July 2004. IGMOD Int'l Conf. Management of Data, pp. 297-306, May 1993.

[4] Abraham Silberschatz and Peter Baer Galvin, *"Operating System concepts"*. Addison Wesley, 1997.

[5] A.Dan and D. Towsley, *"An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes"*, in Proceedings of ACM SIGMETRICS, Boulder, Colorado, United States, 1990, pp. 143--152.

[6] M.J. Bach, *"The Design of the UNIX Operating system"*. Engle wood Cliffs, Nj: Prentice –Hall, 1986.

[7] E.J. O'Neil, P.E. O'Neil and G. Weikum, *"The LRU-K page replacement algorithm for Database Disk Buffering"*

Proc.ACM SIGMOD Int'l Conf. Management of Data, pp. 297-306, May 1993.

[8] Leonid B. Sokolinsky, *"LFU-k: An Effective Buffer Management Replacement Algorithm"*. Database Systems for Advances Applications, 9th International Conference, DASFAA, pp.670-681, 2004.

[9] J.Robinson and M. Deevarakonda, *"Data Cache Management Using Frequency Based Replacement"*, In Proc. ACM SIGMETRICS Conf., pp. 134--142, 1990.

[10] D.Lee, J Choi, J-H Kim, S.L. Min, Y. Cho, C.S. Kim and S.H. Noh, *" On the Existence of a spectrum of policies that Subsumes the Least Recently Used and Least Frequently Used Policies"*, Proc. ACM SIGMENTRICS Int'l Conf. Measurement and Modeling of computer Systems, SIGMENTRICS Performance Evaluation Rev., vol.27, no-1, pp. 134-143, May 1999

[11] S.Jiang and X.Zhang, "LIRS: An efficient Low Inter-Reference Recency Set Replacement Policy to Improve Cache Performance", Proc. SIGMENTRICS, PP. 31-42,2002.

[12] T.Johnson and D.Shasha, *"2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm"*, Proc. Very Large Databases Conf., pp 430-450, 1995.

[13] Jaafar Alghazo, Adil Akaaboune, Nazeih Botros, *"SF-LRU Cache Replacement Algorithm"* Proc. in International Workshop on Memory Technology, Design and Testing (MTDT'04) - Volume 00,pp.19-24,2004.

[14] Michael Factor, Assaf Schuster, Gala Yadgar, *"Multilevel Cache Management Based on Application Hints"*,Technion-Computer Science Department Technical Report CS-2006.

[15] Wayne A. Wong and Jean –Loup Baer, *"Modified LRU policies for Improving second level Cache Behaviour"*, High Performance Computer Architecture (HPCA),pp-49-60, 2000

[16] Nikolas Askitis, Ranjan Sinha, "*HAT-trie: a cache-conscious trie-based data structure for strings"*,ACM Intenational Conference Proceeding Series; Vol. 244, pp 97-105,2007

[17] Aleksandar Milenovic, Milena Milenkovic, Nelson Barnes, " A Performance of Memory Hierarchy in Embedded Systems." System Theory, 2003. Proceedings of the 35th Southeastern Symposium on Publication,16-18 March 2003, pp. 427- 431

[18] James E. Raynolds, Lenore R. Mullin, "Optimizing the Fast Fourier Transform over memory hierarchies for embedded digital systems: a fully in-cache algorithm. Under Review, Journal of Computational Physics, http://trr.albany.edu/documents/TR00004, 2006.

[19] Bevan M. Baas, " A Low Power, High –Performance, 1024-point FFT Processor." Solid-State Circuits, IEEE Journal of , Volume: 34, March 1999 pp. 380 – 387.

[20] M.F. Sakr, C.L. Giles, S.P. Levitan, B.G. Horne, M. Maggini, D.M. Chiarulli, "On-Line Prediction of Multiprocessor Memory Access Patterns", IEEE International Conference on Volume 3, Issue , 3-6 Jun 1996 Page(s):1564 - 1569 vol.3

[21] M. Frigo, C.E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science* (FOCS 99), p.285-297. 1999

[22] Naga k. Govindaraju, Dinesh Manocha, " Cache efficient Numerical Algorithms using Graphics Hardware." Parallel Computing, v.33 n.10-11, p.663-684, November 2007

[23] Jizhu Lu, Acie Nobles, Michael Perrone, " Accelerating FFT Performnace using the Cell BE Processor." IBM Research Report RC24244, 2007.

[24] Bushra Ahsan, Mohmed Zahran, " Managing Off-Chip Bandwidth: A case for Bandwidth-Friendly Replacement Policy." in The 2nd Workshop on Managed Multi-Core Systems (MMCS'09), held in conjunction with ASPLOS 2009.

[25] Jinwoo Suh, Eun-Gyu Kim, Stephen P. Crago, Lakshmi Srinivas, and Methew C. French, "A Performance Analysis of PIM, Stream Processing, and Tiled Processinh on Memory-Intensive Signal Processing Kernals." ISCA03, San Diego, CA, June 2003.