Exploiting Cache in Multimedia

Rita Cucchiara
D.S.I. University of Modena, via Campi 213/b
41100 Modena, Italy
rita.cucchiara@unimo.it

Massimo Piccardi
D.I. University of Ferrara, via Saragat 1
44100 Ferrara, Italy
mpiccardi@ing12.unife.it

Andrea Prati
D.S.I. University of Modena, via Campi 213/b
41100 Modena, Italy
prati@dsi.unimo.it

Abstract

The paper explores cache strategies for multimedia. Although many architectural improvements have been designed for multimedia, the cache structure and the standard caching policies of general-purpose processors exhibit poor performance in exploiting the 2D spatial locality typical of programs handling and processing images. In this paper we propose a novel caching approach suitably tailored to the requirement of multimedia programs. Our proposal exploits hardware pre-fetching for allocating in cache blocks of data that satisfy the 2D spatial locality requirements. Results refer to a benchmark suite of multimedia program including MPEG decoding and image processing programs with different data dependency and access scheme to image data.

1. Introduction

Multimedia is now an impressive reality as a catalyst of different technologies, methods and techniques with the aim of managing, possibly in a strongly interactive manner, digital information such as images, video (from both synthetic and natural scenes), and audio.

The unbelievable multimedia spread has been possible due to the concurrency of two facts: the existence of successful standards such as MPEG [1] and the design of new computer architecture elements, specifically oriented to multimedia applications. As a matter of fact, differently from the common trend in designing high performance general-purpose computers, the multimedia market imposed the hardware manufacturers the integration of some novelties in standard PCs too, for answering multimedia demand.

The answer has been in two directions: first, the addition of multimedia processors or high performance DSPs in computing systems, in order to handle images and sounds concurrently with the normal CPU activity; second the integration inside the computational units of some SIMD improvement (such as Pentium MMX, HP MAX2 or Ultrasparc VIS) for improving the parallel computation on image pixels [2]. Systems for media processing based on a coarse grain parallelism are further proposed [3].

In general, the performance of new computer architectures for multimedia is evaluated with reference to the most common steps in image handling, such as compression and MPEG decoding. In a recent survey, Kuroda and Nishitani [4] compared standard high performance PCs and workstations, DSPs and multimedia processors using MPEG programs as a benchmark. As emphasized in [4], cache architecture, that played a prime role in the explosion of PCs power for general purpose applications, is partially insufficient if not useless for many multimedia tasks.

The most critical problem of cache in handling images is that image processing exhibit an intrinsic *2D spatial locality* instead of the classic 1D spatial locality typical of normal computation and vector processing [4, 5].

2D spatial locality arises since, whenever the CPU accesses a single data item, a high probability exists to access logically adjacent data items in both vertical and horizontal direction. The logical adjacency is due to the classical data type used for images, that references each image point by means of two indexes, but does not correspond to physical adjacency in memory: points of a same column and belonging to two adjacent rows, say a[i][j] and a[i+1][j] are stored in memory with a displacement given by the image size scaled by the point size in byte. Therefore with the standard cache architecture allocating blocks of adjacent bytes, only a 1D *horizontal spatial* locality is taken into account. This problem causes a high amount of cache misses that affect performance especially in PCs having very different access time for on-chip and off-chip memories. In particular, together with the compulsory misses due to the first access to the data item, a large amount of capacity and conflict misses arise [4].

MPEG decoding is one typical task that suffers of this performance degradation. Nevertheless, as it will be discussed in the following, most of the operations required are not data dependent, and addresses are predictable. For this reason, it should take advantage of some techniques such as software or hardware *cache pre-fetching* Hardware pre-fetching mechanisms are particularly studied as a method for decreasing cache misses by predicting the existence of spatial locality at execution time and providing data in cache before these data are effectively accessed.

In this paper we present novel hardware pre-fetching techniques specially oriented to managing images in multimedia application, by exploiting 2D spatial locality.

Performance of different techniques are evaluated with reference to classes of computation common to multimedia applications, and in particular with MPEG decoding and other image processing programs. Results are very promising and shows that computer architecture for multimedia application should include novelties not only in the computational units but in the cache organization too.

2. Related works

A very active research field concerns the evaluation of cache organization and cache access. Cache architecture has high performance only when its structure and access strategy mirror data type structure and data access locality, and now the idea of exploiting more than one cache for matching different and conflicting requirements is well accepted.

Since data and instruction caches have been split in most architecture, several researches refer the performance improvement that can be achieved by further splitting scalar and vector data types or splitting caches for temporal locality and spatial locality [7,8]. This can be used in some multimedia application when singles item of large arrays as images are evaluated once and not reused [9]. As many authors suggest the use of separate cache for vector data in order to adopt aggressive cache access scheme [7,8], we suggest the adoption of a special cache for image data, since this choice, together with a specific cache replacement strategy, reduce cache misses in the very frequent access to image data.

Many researches on caches concern cache software and hardware pre-fetching strategies [6, 10, 11]. In [6] a combined hardware/software solution is proposed. Software pre-fetching is very promising on non data-dependent applications, whenever the access order can be evaluated at compile time [11]. In [6] two important aspects for the realization of *stream pre-fetching* (i.e. two-dimensional data pre-fetching) are outlined: the former one is the *detection* of stream data present in an application, the latter one is the *synchronization* of stream pre-fetch in order to generate pre-fetch request on at a proper moment in time. In the case of software solution, pre-fetch instruction are added in order to obtain correct detection of data to be pre-fetched [9].

Software pre-fetching has the drawback of an increased execution time due to pre-fetch instructions [6]. Although some improvements [11] for this problem, software pre-fetching is not enough effective if the data access is not regular and statistically predictable, as in some processes on images such as labelling or trace contour, that are characterized by a data-dependent propagative access scheme [4].

Hardware pre-fetching schemes sound more interesting since they do not force the compiler to be related to the actual cache architecture; moreover the pre-fetching scheme is computed at run time and can match the data dependency requirements of many image processing and multimedia algorithms. Hardware pre-fetching consists of a logic unit that computes when a block has to be pre-fetched by looking-ahead and predicting which block will be used in the next future [9].

In order to justify the adoption of pre-fetching, a precise performance evaluation and comparison on different pre-fetch techniques for a given class of application or data type is mandatory. Some works explore pre-fetch techniques for image processing or multimedia [4, 5, 12]. In particular [4] states that standard cache pre-fetch mechanisms are not suitable for handling images characterized by 2D spatial locality. Accordingly, we propose new schemes that explicitly exploit 2D cache locality in multimedia applications and compare them to other proposed pre-fetching methods.

3. The working set: algorithms and data types

A great effort is now oriented to the definition of benchmarks for multimedia [2, 12], in order to create a common working set for performance evaluation: as well, we believe that these benchmark are currently too limited since they include only coding/decoding algorithms such as some tasks of JPEG and MPEG standards [1]. Instead, we believe that a complete benchmark suite, at least for the image and video part of a multimedia benchmark should include also basic algorithms of image processing, analysis and understanding that are used (and will be more and more used in the future) for retrieving visual information from pictorial data bases. In previous works we analysed prefetching scheme for image processing only with some results on a simple image database [5].

In this work we extend the analysis by comparing different algorithm classes: *MPEG-2 decoding, convolution, edge tracing.* The choice of this suite is due to the aim of covering different computational models and different data access schemes on images.

MPEG-2 decoding has been selected as an example mixture of different computational models and data access on more bidimensional data structures, as in [2]. The most interesting aspect from the data access point of view is that MPEG-2 works on image blocks (8 x 8 pixels) or macroblocks (16 x 16 pixels), thus imposing a strong 2D data locality. This step is the highest time consuming task, and generally introduces a very large amount of cache misses: obviously a large part are compulsory misses, if no pre-fetch is used; moreover, due to the quite large mask size, the large 2D locality causes a number of cache conflicts as well.

Convolution is the basic and most widely used algorithm for low-level image processing. It has been taken as the example of the class of raster-scan algorithms (including many algorithms for image compression, filtering, segmentation and feature extraction): they are mainly composed of a nested loop on rows and columns iterated containing the instruction stream that has to be executed on each single data item and its neighbour (of a given K size). Although their near-neighbor 2D spatial locality, the raster-scan algorithms should work well on standard caches since at least the temporal locality can be highly exploited. The initial compulsory misses concern points belonging to the required K rows: if no capacity misses arise, the points fetched and allocated in caches will be used also for the following points in the row; in this case, the number of misses is the same as for the simple one-directional raster scan. In addition, cache performance in terms of miss rate can be strongly improved with data pre-fetching techniques, and in particular with *sequential pre-fetching* that takes advantage of the spatial locality by pre-fetching consecutive cache blocks.

Finally, the case of edge tracing, has been taken into account as an example of image processing tasks characterized by an *unpredictable* 2D spatial locality, since the data access is mainly data-dependent and not performed in raster-scan manner. We selected a standard edge tracing algorithm that scans objects' contours and can be used for object matching or computing objects' signature for content-based retrieval. From the point of view of data access, this class of algorithms show the same bidimensional spatial locality of the raster-scan ones and at the same time the impossibility of predicting the data access. Moreover, the temporal locality is difficult to emphasize, because points involved in the neighbor computation may be used in the future, but may be after long computation and thus capacity misses probably occur.

Table 1. Workload.

Algorithm	Data source	Program	Image size	Total number of references	N. of image references	N. of scalar references
MPEG-2 decoding	Video of 69 frames	Mpeg2dec.c	256 x 192	81736432	24629902	57106530
Convolution	DARPA Test3 image	Convol.c	512 x 512	9401964	7022700	2379264
Edge tracing	DARPA Test3 image	Trace.c	512 x 512	1118438	619040	499398

In this case, using a standard cache architecture that fetches and stores only physically adjacent blocks, locality can't be fully exploited. In this context, standard cache optimization techniques are useless: blocking strategies (that are very useful in matrix operation and in convolution) can not be performed since no a-priori data partitioning is allowed; use of large block sizes, often used for reducing cache misses, doesn't catch the vertical spatial locality; even software or hardware sequential pre-fetching doesn't overcome the outlined drawbacks.

Table 1 summarize the workload that we are compared, by indicating for each program the main parameters and the number of memory references, separated for image data and scalar ones. For shake of generality, convolution and edge tracing algorithms are extracted from the DARPA Image Understanding benchmark (working on 512 x 512 images).

4. 2D block pre-fetching

It is well known that spatial locality is exploited in standard cache architectures by caching blocks (or lines) of adjacent data, that is data that are stored in adjacent cells in main memory. This is shown in Fig 1.a, where the cache contains block a[i], which TAG is the address reference of the a[i] datum and its adjacent data.

In programs like MPEG or image processing, when the a[i][j] datum (an image pixel) is referred, there is a high probability to use both a[i][j+1] (horizontally adjacent) and a[i+1][j] (vertically adjacent) in the next future. Therefore, an image-oriented cache architecture should store 2D cache blocks in place of the classical one-dimensional blocks. A *2D block* is a bidimensional set of pixels belonging to adjacent image rows; its size and form factor should be chosen so as to catch the maximal statistical hit rate. Since 2D adjacent blocks are not consecutive in main memory, the block in cache needs the addition of a form factor tag (FF in Fig 1.b) for maintaining a precise correspondence between cache and main memory. FF takes information on the size (because images of different size are used in one program, such as luminance and chrominance images in MPEG), the block form and the displacement between rows (the N*b value, being the image size N and the data type b).

This architecture is not easy to design and implement, since it requires a complex logic for accessing and replacing 2D blocks, as well as the factor form tag. In alternative, the approach we propose does not controvert the basic cache architecture, and uses as many 1D cache blocks as the rows of the 2D data block (each one with an individual tag) as in Fig. 1.c; nevertheless, it exploits a hardware pre-fetching technique for achieving the same results. When a miss happens, the line containing the missed reference is allocated in cache (a[i][j] in Fig. 1.c), and the other block lines are pre-fetched. Since the other block lines have a constant displacement depending on the image size, the information of the *stride* for the line to be pre-fetched can be computed at run-time with a very simple hardware mechanism as indicated in Fig. 1.c: an *Image Reference Table* maintains information on the displacement with respect to the physical address in order to find the next block line to be pre-fetched for each allocated image. This table can be precompiled at compile time, whenever the image size is known, or can be maintained at run-time as a sort of small cache storing the last references on image data. Many similar proposals discusses history tables or reference prediction tables [6, 10]. Our proposal can be viewed as a pre-fetch mechanism with constant stride [10], with the difference that the stride is not algorithm or code-dependent but is dependent on the data structure, i.e. the image. In this sense, it is very general, because it only needs that images are marked as bidimensional data at compile time

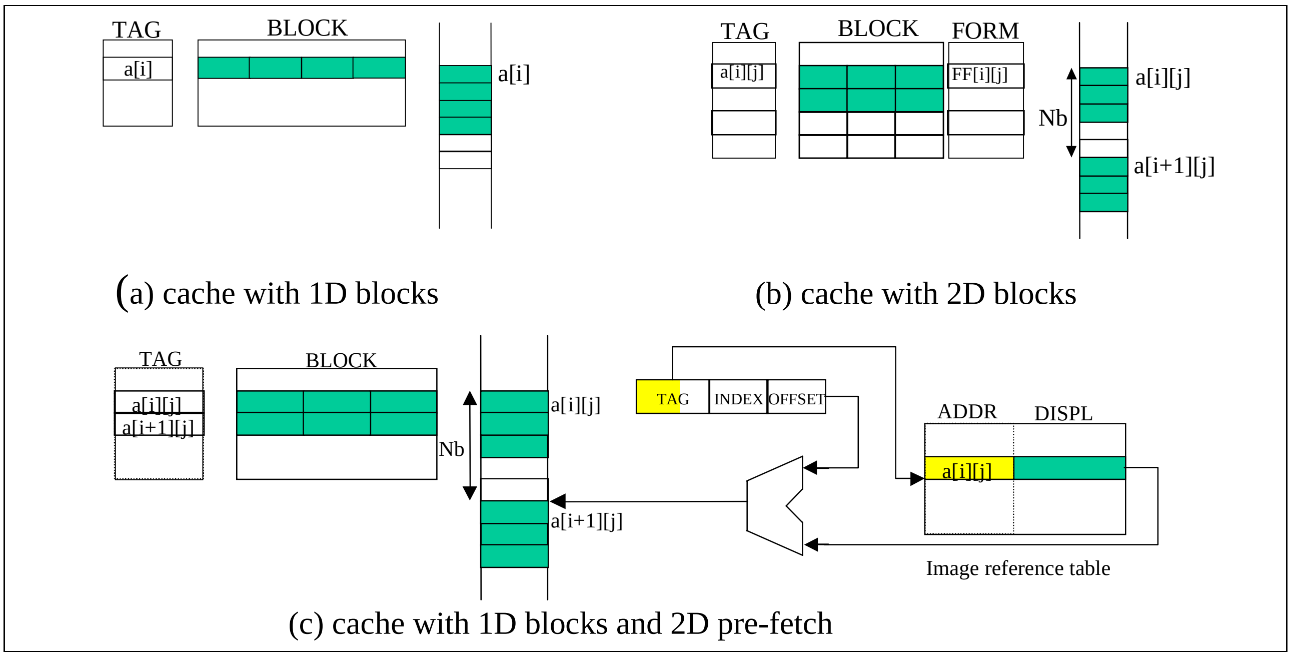

Fig. 1. 2D block.

5. Performance results

The first results we discuss concern the MPEG decoding program: Table 2 reports the number of cache misses measured by simulating program's execution on the MPEG video with different pre-fetching strategies: no pre-fetch, OBL (one block lock-ahead, i.e. sequential pre-fetch), and 2D pre-fetch.

Results are reported in Table 2 for a B line size of 16 and 32 bytes, two-way set associativity, and 8, 16, 32 KB S cache size; the 2D pre-fetch adopts 2 x B blocks, while OBL, that fetches two consecutive lines, results in an overall 1 x 2B block. In this simulation, *only image references are cached* (image data).

Table 2. Number of misses with respect to the S cache size, *image* data

B = 16

Type of fetching	S = 8K	S = 16K	S = 32K
No pre-fetch	**881105**	**827390**	**810758**
OBL pre-fetch	**710813**	**519709**	**485864**
2D pre-fetch	**478936**	**434046**	**416222**

B = 32

Type of fetching	S = 8K	S = 16K	S = 32K
No pre-fetch	**811343**	**578535**	**527403**
OBL pre-fetch	**805144**	**492537**	**390541**
2D pre-fetch	**583413**	**336957**	**286326**

Table 2 shows that the miss rate achieved with 2D pre-fetching is significantly lower than the one achieved with OBL, and is very close to half of the miss rate without pre-fetching; this result proves that for MPEG a 2 x B block is statistically preferable to 1 x 2B (OBL) at a parity of transfer bandwidth between main and cache memories. Similar results are achieved with higher associativity, and are not reported in the paper.

Since the results seem to justify the 2D approach to pre-fetching, we have performed other experiments with a greater number of lines in the block. Table 3 reports the number of misses for caches with 4 lines in the block (4 x 16 blocks for 2D, and 1 x 4 * 16 blocks for OBL – for sake of briefness, we still call OBL the "horizontal" pre-fetching, even if more lines are pre-fetched).

Table 3. Cache misses vs. form factors. S = 16 KB

Type of fetching	No. of misses
No pre-fetch 1 x 16	**827390**
OBL 1 x 2 * 16	**519709**
OBL 1 x 4 * 16	**520942**
2D 2 x 16	**434046**
2D 4 x 16	**223147**
2D 2 x 32	**336957**

For 2D pre-fetch, the number of misses decreases by the half with 4 x 16 with respect to 2 x 16; instead, for OBL, the number of misses decreases only slightly with 1 x 4 *16 against to 1 x 2 * 16 (32 KB cache) , or even worsen (16 KB cache); thus, a greater number of lines is not justified for horizontal pre-fetching, since it requires double the memory traffic. The performance of a 4 x 16 cache are also better than a 2 x 32 cache, at parity of size.

These results highlight the nature of the MPEG algorithm, which is mainly based on 8 x 8 block processing, with many intra-block accesses and some inter-block ones. The high probability of inter-block access, greater for the horizontal direction but substantial also for the vertical direction, is the reason for the usefulness of a bidimensional caching scheme, and explains the good performance of a 4 x 16 form factor .

Table 4.a. Cache misses, non-image data (B=16)

Type of fetching	S = 8K	S = 16K	S = 32K
No pre-fetch	**19241**	**6034**	**4010**

Table 4.b. Split vs. unified caches, all data (B=16)

Split cache (S/2 image, S/2 non-image)

Type of fetching	S = 16K	S = 32K
No pre-fetch	**900346**	**833424**
OBL pre-fetch	**730054**	**525743**
2D pre-fetch	**498177**	**440080**

Unified cache

Type of fetching	S = 16K	S = 32K
No pre-fetch	**949526**	**878569**
OBL pre-fetch	**683989**	**555337**
2D pre-fetch	**545207**	**476323**
2D pre-fetch for images, OBL for non-images	**531941**	**469409**

A complete program execution involves also many data references to non-image data. In fact, when all data references have to be cached, either a unified cache could be used for both image and non-image data, or a split cache could be used in alternative. The former approach provides higher allocation flexibility, since all the cache space can be used for each kind of data; the latter avoids interference and potential pollution between data characterized by different forms of locality [7] (temporal, 1D spatial, or no locality, vs. 2D locality). Indeed, in both approaches different form factors could be exploited in order to better mirror the proper form of locality. Table 4 compares the results achieved by simulating unified and split caches.

The split cache has been simulated by equally dividing the overall cache space between image and non-image data; different sizes could be adopted, anyway, since the miss rate on non-image data is much lower than that on image references (see Table 4.a). Table 4.b shows that in most cases the split cache allows a less number of misses, and this is especially true for the case of 2D pre-fetch. In addition, Table 4.b shows that 2D pre-fetching performs best for both kind of caches.

For the unified cache, two pre-fetching strategies have been used: 2D pre-fetch for all data, versus 2D pre-fetch for images and OBL for non-images; Table 4.b shows that the latter technique slightly reduces the number of misses.

These results prove that there is an evident trade-off between cache design and performance: the best results are achieved with a split cache, since a substantially-different access statistic underlies image and non-image data. If a unified cache is used for sake of simplicity, a 2D pre-fetching strategy results more efficient than traditional OBL pre-fetch; this results is not obvious, since the total number of non-image accesses is much greater than image ones (about $57 * 10^6$ vs. $24 * 10^6$), but the much lower miss rate of the non-image references makes 2D pre-fetch preferable as unified pre-fetching strategy. Moreover, if two separated pre-fetch strategies can be used, the adoption of 2D pre-fetch for images and OBL for non-images achieve a further improvement.

The previous analysis has been devoted to the MPEG program, as a well-accepted benchmark of architectures for multimedia [2]. In this paper we extend the benchmark suite image processing and analysis algorithms.

Table 5. Number of misses for image references (B = 16, S = 16K, DARPA benchmark Test3).

Type of fetching	Convolution	Edge Tracing
N. of references:	**7022700**	**619040**
No pre-fetch	**32704**	**72508**
OBL pre-fetch	**16352**	**39400**
2D pre-fetch	**16352**	**37864**

Table 5 shows that convolution has a much greater number of image references with respect to edge tracing (about 7 * 10^6 vs. 6 * 10^5), but the number of misses is significantly higher for the latter program, due its irregular memory access scheme (the miss rate is more than one order of magnitude greater); thus, its impact on performance is even stronger.

In order to assess overall results, we have scaled the number of image references and cache misses to the same size and number of frames of the MPEG video (it seems to be more reasonable, in the hypothesis that in a multimedia application a sequence of frames are processed by the same algorithms) Table 6 reports the compared results.

Table 6. Number of misses for image references (MPEG, Convolution and Edge Tracing for a sequence of frames).

Type of fetching	MPEG	Convolution (scaled)	Edge Tracing (scaled)
N. of references:	**24629902**	**90913002**	**8103849**
No pre-fetch	**827390**	**423373**	**938659**
OBL pre-fetch	**519709**	**211687**	**510057**
2D pre-fetch	**434046**	**211687**	**490172**

As it can be seen, the number of misses is surprisingly roughly comparable for all programs, although the total number of references is very different. This results is relevant since it highlights the importance of optimizing each program with respect to cache performance.

With these parameters, the convolution program exhibits only compulsory misses: its number of misses is independent on the pre-fetching strategy and depends only on the amount of pre-fetched data. Thus, minimizing the number of misses can be achieved with standard interleaving of pre-fetching activity with cache access, exploiting at most the memory bandwidth.

Instead, for both MPEG decoding and edge tracing, 2D pre-fetch allows a decreased number of misses; in both cases, 2D pre-fetch seems to better catch the locality of algorithms, although the very different nature of their memory access scheme.

6. Conclusions

The results outlined in the previous section have been obtained with a simulator that performs allocation on misses, both for read and write, exploiting an exact LRU substitution policy and divides data and instruction caches. With our modified version, pre-fetch strategies, too, can be simulated. We remind that we simulated pre-fetch on miss only. Pre-fetch can occur either on reference or on miss. In the first case it is activated on each memory reference and, in the best theoretical case, it could eliminate any compulsory miss. In this work we investigated pre-fetch on miss: it consists only of deciding which block has to be fetched in addition to the one required by the miss and thus it could be viewed as a modified block replacement scheme. Therefore, by definition, this strategy can reduce but not nullify the cache misses. Nevertheless, these results can be extended to the case of cache on reference straightforwardly

In this paper we have proved how programs working on images can improve their performance if a specifically oriented caching policy is adopted. In particular we have proposed to exploit the 2D locality that is typical of most image processing and multimedia algorithms. To this aim, we have proposed a novel 2D pre-fetching technique. The good results suggest to continue the investigation of special-purpose cache strategies for resolving the matching between data structure and algorithms in multimedia applications.

Bibliography

[1] L. Chiariglione, “Impact of multimedia standards on multimedia industry”, Proceedings of IEEE, v. 86, n. 16, 1998, pp. 1222-1227.

[2] K. Diefendoff, P.K. Dubey, “How multimedia workloads will change processor design”, IEEE Computer,Sept 1997.

[3] J.A. Watlington, V.M. Bove Jr., “A system for parallel media processing”, Parallel Computing 23 (1997), pp. 1793-1809

[4] I. Kuroda, T. Nishitani, “Multimedia processors”, Proceedings of IEEE, v. 86, n. 6, 1998, pp. 1203-1221.

[5] R. Cucchiara, M. Piccardi, “Cache pre-fetching for image processing”, Proc. of IEEE MTAC'98, Third Annual Multimedia Technology and Applications Conference, Sept. 1998, Anaheim, CA, USA.

[6] P. Struik, P. van der Wolf and A. D. Pimentel, ``A Combined Hardware/Software Solution for Stream Prefetching in Multimedia Applications'', in Proc. of the 10th Annual Symposium on Electronic Imaging , pp. 120--130, San Jose, USA, January, 1998.

[7] A. Gonzalez, C Aliagas, M. Valero, “A data cache with multiple caching strategies tuned to different types of locality”, Proc. of ACM Int. Conf. on Supercomputing, Barcelona, Spain, 1995, pp. 338-347.

[8] V. Milutinovic, B. Markovic, M. Tomasevic, M. Tremblay, “The split temporal/spatial cache: initial performance analysis”, Proc. of the SCIzzL-5, Santa Clara, CA, USA, 1996.

[9] R. Lee, J. Huck, “64 bit and multimedia extension in the PA-RISC 2.0 architecture”, Proc. of IEEE COMPCON 96, 1996, pp. 152-160.

[10] C T.F. Chen, J. L. Baer, “A performance study of hardware and software data prefetching schemes”, Proc. of 21st Int. Symp. Computer Architecture, 1994, pp. 223-232.

[11] T. Mowry, M.Lam, A Gupta, “Design and evaluation of compiler algorithm for prefetching” in SIGPLINE Notices, Sept. 92, pp. 62-73.

[12] D. Zucker, M.J. Flynn, R. Lee, “A comparison of hardware prefetching techniques for multimedia benchmark”, Proc. of IEEE Multimedia 96, pp. 236-244.