Hardware Prefetching Techniques for Cache Memories in Multimedia Applications

R. Cucchiara
DSI, University of Modena and Reggio Emilia
Via Campi 213/b
41100 Modena, Italy
rita.cucchiara@unimo.it

M. Piccardi
DI, University of Ferrara
via Saragat 1
44100 Ferrara, Italy
mpiccardi@ing.unife.it

A. Prati
DSI, University of Modena and Reggio Emilia
Via Campi 213/b
41100 Modena, Italy
prati@dsi.unimo.it

Abstract

The workload of multimedia applications has a strong impact on cache memory performance, since the locality of memory references embedded in multimedia programs differs from that of traditional programs. In many cases, standard cache memory organization achieves poorer performance when used for multimedia. A widely explored approach to improve cache performance is hardware prefetching that allows the pre-loading of data in the cache before they are referenced. However, existing hardware prefetching approaches partially miss the potential performance improvement, since they are not tailored to multimedia locality. In this paper we propose novel effective approaches to hardware prefetching to be used in image processing programs for multimedia. Experimental results are reported for a suite of multimedia image processing programs including convolutions with kernels, MPEG-2 decoding, and edge chain coding.

1. Introduction

Cache memory performance plays a more and more critical role in computer systems, since the difference in speed between processor and main memory tends to increase rather than the contrary.

Among the various techniques used to improve cache memory performance, prefetching has been one of the most studied and apparently promising. Prefetching techniques can be mainly classified according to their potential software or hardware implementation, although some techniques may take advantage of a combined software/hardware implementation [3]. The main difference is that software techniques assume that compiler is aware of the underlying cache organization and that program code can be modified at compile time with the insertion of specific prefetch instructions; the placement of prefetch instructions is based on a compile-time prevision of memory references. For this reason, software prefetching is not suitable for data-dependent executions. Instead, hardware techniques do not require modification of program code, while at the same time they exploit a run-time prevision of memory references, therefore allowing even data-dependent prediction. The main disadvantage of hardware prefetching is that it requires hardware resources and that all functionalities must be performed at run time, where speed requirements are mandatory in order not to compromise cache effectiveness.

The design of new computer architecture is justified by performance improvement on application workloads chosen as benchmark. In this work, we have addressed a multimedia workload since multimedia applications are spreading and they must orient the design of modern computer platforms [5]. In particular, we have addressed multimedia image processing, where we have included algorithms like the widespread MPEG-2 decoding used for decompression of audio/video streams and typical image processing operations like convolution for image filtering and edge chain coding, used as a pre-processing step in many image analysis tasks. We have omitted evaluation on sound data (like MP3 decompression or speech recognition), since they exhibit typical array spatial locality and standard prefetching techniques perform well enough. Algorithms have been selected according to their spread and their different data addressing schemes: while convolution is dominated by a regular data addressing scheme which can be predicted a priori, edge chain coding is heavily *data dependent*, in the sense that the address sequence of data references depends on the image and cannot be statically predicted. MPEG-2 exhibits a combination of regular address scheme and data dependency.

One of the most critical aspects affecting prefetching effectiveness is that image processing exhibits an intrinsic *2D spatial locality* instead of the classic one-dimensional

(1D) spatial locality typical of normal computation and array processing [23][7]. Typical hardware prefetching techniques are not suitable in this context: techniques based on one-block-lookahead [25] exploit only 1D spatial locality, while adaptive techniques do not match data dependency of some image processing algorithms.

In this paper, we present novel hardware prefetching techniques oriented to multimedia image processing matching the 2D spatial locality. We compare their performance with that of other existing hardware prefetching techniques in terms of avoided cache misses and total number of prefetches introduced.

The rest of the paper is structured as follows: Section 2 addresses related works on hardware prefetching techniques. In section 3, the hardware prefetching strategies are presented, including novel techniques aimed at exploiting the 2D spatial locality. Section 4 describes the multimedia image processing algorithms used for the tests. In section 5 experimental results and performance analysis are proposed in terms of eliminated cache misses, while section 6 accounts for prefetching costs. The conclusion presents final considerations on the profitability of the presented approach.

2. Related works

Many recent works have been focussed on hardware prefetching techniques, since recent improvements in hardware technology has made more feasible the run-time functionalities required by hardware implementation [28][4][27]. The basic technique is known as one-block-lookahead [25] (also called *always prefetch* in a more recent work [27]). With this technique, every time a reference to block i is made, a lookup in the cache is issued for block i + 1; if the block is absent, it is prefetched. The implicit assumption is that block i + 1 has a high probability to be referenced in the near future, and that scheduling its prefetching on the first reference to block i grants adequate timeliness (timeliness expresses the property that a prefetched block is not loaded too early, so as to risk substituting useful blocks already present, while at the same time allowing prefetch completion before the block is actually referenced [14][19]). Several more complex approaches have been proposed, taking into account different forms of adaptivity in order to achieve a better prevision than the one represented by block i + 1. The main idea of adaptivity is to induce block reference probability by recent references history. A basic algorithm computes the *stride*, i.e. the address difference between the last two memory references made by a same instruction, and adds it to the address of the last memory reference to obtain block prevision. Chen and Baer in [4] propose the use of a Reference Prediction Table (RPT) for achieving reference prediction and a state machine to assert if the prediction can be trusted (correct) or not (incorrect). In the state machine proposed in [4], two successive successful predictions must be made in order to enter a steady state of prediction from the initial no-prediction state. This idea has some similarity with the two adaptive techniques compared in this work, where the stride is considered reliable for reference prevision if it is stable since the last two references [9], aiming at filtering isolated strides that are for instance frequent in MPEG decoding. Some authors propose more complicated form of adaptivity that we judged too demanding in hardware resources to be feasible and therefore didn’t consider in this paper [19][17]. Tse and Smith in [27] propose a system-level analysis of prefetching impact on system performance by way of an accurate cycle-by-cycle execution simulation. All system aspects relevant to performance are considered in detail, including cache and block size, associativity, main memory latency, single or dual-ported tag and data arrays, split or non-split bus transactions, and others. This accurate cycle-by-cycle simulation allows the computation of the actual impact of memory access on the execution time. However, other metrics [14][3][28] like the number of cache misses eliminated are useful for an initial evaluation of prefetching techniques because they focus on cache performance independently of system parameters.

Multimedia image processing algorithms exhibit a substantial amount of a form of spatial locality called two-dimensional (or 2D) spatial locality [8][23][7] that has not been taken into account in traditional cache architectures. Therefore, evaluation of prefetching performance must be extended also to adequate multimedia benchmarks. In previous works we explored cache exploitation for simple image processing tasks [6]; we extended it to MPEG in [7]. A recent work from Zucker, Flynn and Lee [28] proves the effectiveness of hardware prefetching on MPEG algorithms; the authors propose the use of adaptive techniques based on a Stride Prediction Table (SPT), together with a multiple-way stream memory. Instead, in this work we mainly want to explore if novel, non-adaptive prefetching techniques based on 2D spatial locality may lead to significant performance improvement on a larger set of multimedia image processing programs.

Other works related to our research are those on split caches [12][24]. Recent research reports the performance improvement that can be achieved by splitting scalar and vector data types or splitting caches for temporal locality and spatial locality [12][24][6]. According to this trend, in this work we propose the adoption of a special cache for image data together with a dedicated prefetching policy.

3. The hardware prefetching techniques

In multimedia image processing, two main properties can be exploited to achieve high prefetching effectiveness:

1. *locality* of image data access;
2. *access predictability* of typical algorithms.

In particular, in this work we have compared different prefetching schemes, including simple schemes like one-block-lookahead, basic adaptive prefetching and other original ones designed to suitably mirror these properties, and measured them on different workloads. The compared schemes can be divided into two categories, namely static and adaptive prefetching schemes. <u>Static prefetching schemes</u> are:

a) ***OBL*** (one-block-lookahead);
b) ***OBVL*** (one-block-vertical-lookahead);
c) ***neighbor,***

while <u>adaptive prefetching schemes</u> include:

a) (basic) ***adaptive***
b) ***ada_2/OBL*** (2-step adaptive with OBL)
c) ***ada_2/OBVL*** (2-step adaptive with OBVL)

OBL is the most commonly adopted pre-fetching scheme as it is simple and can be very effective with scalar and vector data since it is able to exploit the spatial locality by looking ahead the basic line; for this same reason it proves very effective also on raster-scan (pixel-by-pixel access in the order of spatial locality) image processing algorithms.

In *OBVL prefetching*, proposed by the same authors of this paper, 2D spatial locality is exploited [7]. This kind of locality arises since a high probability exists to access logically adjacent pixels in both vertical and horizontal directions. The logical adjacency does not correspond to physical adjacency in memory: pixels in a same column 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 pixel size in byte [6]. In OBVL prefetching, the line just below the one currently referenced is prefetched if absent from the cache. The rationale is to explore if this block may be more useful than the following horizontal block prefetched by OBL with respect to the following in vertical order.

In this paper we propose a further exploration of 2D spatial locality by the *neighbor* prefetching, where all lines containing pixels in the 3 x 3 neighborhood of the currently referenced pixel (called 8-connected lines) are checked for presence in the cache, and their blocks prefetched if required. This technique carries an intrinsic increase of the lookup pressure in the cache, but it will be later shown that the number of blocks actually prefetched is comparable with those of the other techniques.

These techniques are all static in the prevision of the block(s) to be prefetched, and mainly aim to explore an a-priori probability of access intrinsic in multimedia image processing. Nevertheless, adaptivity may prove useful, too, and therefore we explored various adaptive techniques. The basic adaptive prefetching is based on an instruction-based reference prediction table [28]: for each instruction making a reference to image data, the difference between the current and previous data addresses is computed (stride); the stride is added to the current address to form the prefetch address. The 2-step adaptive techniques trust the stride only if it is equal to the previous one; this is meant to filter isolated strides from being used for prefetch prevision [9]. We propose two variants: in case the last two strides are not equal or no prediction is present in the prediction table, a prevision is made on a static probability assumption: one-block-lookahead for *ada_2/OBL*, while OBVL prefetching for *ada_2/OBVL*.

The reference cache architecture on which the prefetching techniques have been tested is an image cache with 2-way set associativity, 16 byte block length and a 32 Kbyte size. These parameters are those of a typical L-1 cache; experiments with different cache parameters are also presented in this paper. Simulations have been performed by collecting all traces of program execution on a Sparc 10 platform using the Spy trace generator [15], and giving them to a cache simulator derived from ACME [1] that we modified in order to support and measure prefetching. All the compared techniques are based on prefetching *on reference*, and we assume that prefetching is completed before the next memory reference is issued; for the adaptive techniques we use a 128-entry Reference Prediction Table with a pseudo-random replacement policy. Although many system organizations allow variations from this scheme, these assumptions are not limiting for the scope of this paper, which is mainly to explore how locality can be more effectively exploited for prefetching in multimedia image processing.

4. The multimedia image processing benchmark

Performance of prefetching schemes in multimedia image processing can be influenced by several factors, such as the program type and the input images. The program type is the most critical factor since the type of locality embedded in the program is the main cause of success of a prefetching technique. For this reason we have considered different types of algorithms, very common in multimedia, which exhibit a very different locality.

The algorithms are:

a) image convolution
b) chain code computation
c) MPEG-2 decoding

Convolution is the basic algorithm for image processing; it consists of processing each image pixel by convolving pixels of its bidimensional neighborhood with a coefficient mask. In many papers addressing performance evaluation in image processing [6][7][28][26][2], convolution is included in the basic benchmark, since it is very common. Examples of programs based on convolution are filtering for noise cleaning, image enhancement, edge detection and template matching. Image is evaluated in raster-scan mode, and the algorithm execution and the data access is not data dependent; a substantial amount of strictly 2D locality is embedded.

The *Chain code* computation is included as a typical data-dependent program on images. It is a propagative algorithm in which data access is propagated in the image in two directions depending on the edge connectivity. Chain code computation starts choosing an initial edge pixel. Then, its 8-connected pixels are checked: if there is an edge pixel, it is linked to the edge chain and a defined code is computed for storing the direction change of the edge; then, the previous step is repeated from the linked pixel until the initial pixel is reached [10][22][20]. This algorithm has been included as an example of image computation that potentially does not benefit from standard cache techniques since it exploits an unusual and non-predictable spatial locality. Since computation is data dependent, prefetching performance varies with the input image. Chain codes are useful for describing edges of shapes in image, image compression and image description; they are possible shape coding techniques for MPEG-4 [16][21][11].

Finally, *MPEG-2 decoding* is the typical benchmark for multimedia processors, since it is currently the most used multimedia program [16][21][23][11][4]. MPEG-2 decoding consists of several steps: 1) variable-length decoding and inverse quantization of discrete cosine transform (DCT) coefficients; 2) Inverse DCT for computing original frames; 3) motion prediction by correlation of two or three consecutive frames. The most interesting aspect for data access is that step 2) works on image blocks (8 x 8 pixels) or macroblocks (16 x 16 pixels), thus imposing a strong 2D data locality. MPEG-2 decoding typically carries a high number of cache misses [4]: obviously, a large part are compulsory misses, if no prefetching is used; moreover, due to the relatively large mask size, the large 2D locality causes a number of cache conflicts as well. The number of misses depends on the format of image data and compression; for this reason many different examples of MPEG videos have been compared.

5. Performance analysis

Basic performance metrics for cache memories are the number of cache misses and the miss rate, which is a more general measure since it doesn't depend on the total number of references. In this section, we call *NM* and *MR* the number of misses without prefetching (*No_Pf*), and for each i-th prefetching method, NMP_i its related number of misses and MR_i the correspondent miss rate, respectively. Starting from the number of misses, we also define the *efficacy* of the i-th prefetching method as the ratio

$$\eta_i = \frac{NM - NMP_i}{NM}$$

Efficacy ranges between 0 and 1 and it tends to 1 when prefetching achieves high performance, that is its NMP_i (i.e. the number of misses that have not been eliminated) tends towards 0. NMPi, MRi, and ηi values are used in the rest of the section to compare the performance of the considered hardware prefetching techniques. However, the prefetching activity in general can affect the execution time since it occupies the bus and introduce a further lookup pressure on the cache; therefore the number of prefetches issued or the prefetching rate can be taken into account as measures indicating the prefetching costs, which will be discussed in section 6.

In the following, we report only the misses occurring when image data are accessed. This choice is due to two motivations. First, the number of accesses to image data in the programs considered for multimedia and image processing applications is considerably higher than the number of accesses to scalar data or other vector data; second, it is possible to use a separate cache for the image data, with associated prefetching techniques, in order to achieve better performance while granting less interference with data with different locality.

5.1 Convolution

Convolution is representative of the basic raster-scan near-neighbor computation [13]. In this case, all simulations confirm the expected abatement of misses adopting any reasonable prefetching strategy with respect to the non-prefetching case. The strong regularity of the algorithm and its uniform 2D spatial locality makes the number of misses almost *invariant* with the prefetching strategy. Table 1 shows results with the six prefetching methods for a convolution of a 512 x 512 image with a 5 x 5 mask. The miss rate is low even without prefetching (consisting only of compulsory misses) but is practically nullified by prefetching. The classic one-block-lookahead prefetching performs well since the data access follows the row-by-row data store direction. The neighbor method is able to eliminate all misses (apart from the first image reference) since it mirrors the data access locality; although it checks only the 8-connected lines, this is enough to pre-load data before their reference even for convolution with any mask size. All static methods show effective miss elimination (also the OBVL method proves adequate timeliness); as the data access is easily predictable, adaptive techniques exhibit high efficacy, too.

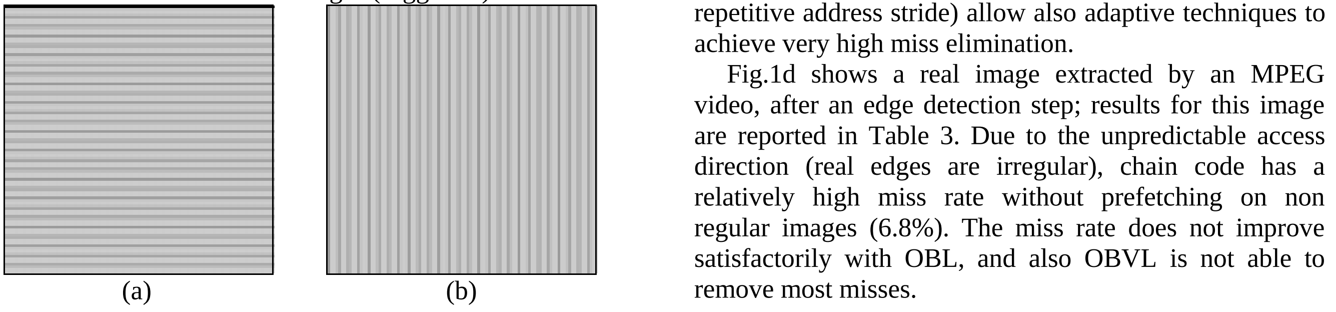
Table 1. Efficacy for Convolution

Convolution 5 x 5		**Nref=19.504.949**	
	NMPi	**MRi%**	**η%**
No_Pf	32738	0,16784%	0,00%
OBL	6	0,00003%	99,98%
OBVL	64	0,00032%	99,80%
Neighbor	2	0,00001%	99,99%
Adaptive	66	0,00033%	99,79%
ada_2/OBVL	35	0,00017%	99,89%
ada_2/OBL	35	0,00017%	99,89%

In the case of convolution programs, the advantage achieved with prefetching is so evident that its adoption is highly convenient – unless interference with other data could cause cache conflicts and decrease performance, thus confirming the usefulness of a separate cache for image data [6].

5.2 Chain code

Chain code is data dependent and propagative since it follows the edge direction. In order to stress the impact of data dependency, first we report results for some limit cases of synthetic images with horizontal, vertical, and both horizontal and vertical edges (Figg.1a-c).



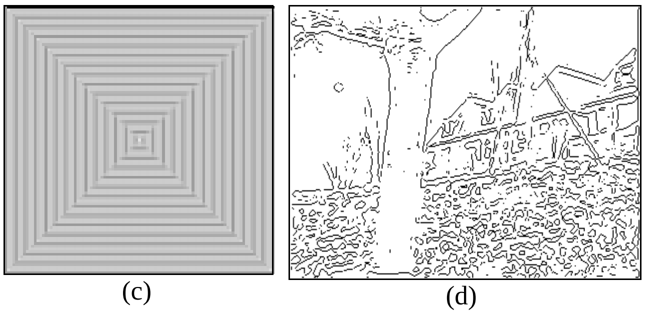


Figure 1. Images used in chain code: (a) horizontal.img, (b) vertical.img, (c) spiral.img, (d) frame of mei16l (after edge detection).

Table 2 compares results for the limit-case test images; let us consider static techniques firstly. In the image in Fig. 1a data access is mainly in the horizontal direction. Thus a basic technique like OBL works well, as shown in Table 2; instead, OBVL prefetching, privileging vertical direction, is not able to eliminate a substantial amount of misses. Instead, with the image in Fig. 1b, OBL shows unsatisfactory performance, while OBVL nearly eliminates all misses. Thus, these two static techniques exhibit their complementary behavior. As it is easy to foresee, both OBL and OBVL prefetching are disrupted by a pattern like the one in Fig. 1c, assessing poor performance. However, in all those three cases, neighbor prefetching achieves best performance, with nearly total miss elimination. The regularity of patterns (carrying a repetitive address stride) allow also adaptive techniques to achieve very high miss elimination.

Fig.1d shows a real image extracted by an MPEG video, after an edge detection step; results for this image are reported in Table 3. Due to the unpredictable access direction (real edges are irregular), chain code has a relatively high miss rate without prefetching on non regular images (6.8%). The miss rate does not improve satisfactorily with OBL, and also OBVL is not able to remove most misses.

Table 2. Efficacy for Chain on limit cases.

	Chain horizontal.img Nref=983271			**Chain vertical.img** Nref=981744			**Chain spiral.img** Nref=980471		
	NMPi	**MRi**	**η%**	**NMPi**	**MRi**	**η%**	**NMPi**	**MRi**	**η%**
No_Pf	32769	3,3326%	0,00%	180189	18,354%	0,00%	112265	11,450%	0,00%
OBL	260	0,0264%	99,20%	131073	13,351%	27,25%	72304	7,3744%	35,59%
OBVL	8002	0,8138%	75,58%	354	0,0360%	99,80%	39419	4,0204%	64,88%
Neighbor	4	0,0004%	99,98%	260	0,0264%	99,85%	5	0,0005%	99,99%
Adaptive	263	0,0267%	99,19%	647	0,0659%	99,64%	14	0,0014%	99,98%
Ada_2/OBVL	263	0,0267%	99,19%	487	0,0496%	99,72%	254	0,0259%	99,77%
ada_2/OBL	260	0,0264%	99,20%	902	0,0918%	99,49%	151	0,0154%	99,86%

Table 3. Efficacy for Chain on a real image.

Mei16l.img Nref=1520145			
	MNPi	**MRi**	η%
No_Pf	103476	6,8069%	0,00%
OBL	86489	5,6895%	16,41%
OBVL	12081	0,7947%	88,32%
Neighbor	335	0,0220%	99,67%
Adaptive	4466	0,2937%	95,68%
ada_2/OBVL	3609	0,2374%	96,51%
ada_2/OBL	5926	0,3898%	94,27%

Adaptive methods are interesting if edges are straight, which is not often the case in real images. In this case neighbor method is the best one again (two orders of magnitude better than OBL and one order better than the other techniques) in terms of miss rate, since in any case chain code computation accesses the 8-connected pixels in order to choose the edge direction to be followed.

5.3. MPEG-2 decoding

The MPEG-2 decoder is a complex program, in which locality and data access order are not easy to model. The access to image data is raster-scan and computation is mainly local (in blocks of 8 x 8); in part the access is random between different blocks. The locality of MPEG and the efficiency of standard caches have been evaluated in detail in [26].

MPEG frames are of three different types: I (only spatial compression in a JPEG style), P (forward-predicted), and B (forward-backward predicted); usually a whole movie is a periodical repetition of frame types, called the sequence pattern. For the test we have used different MPEG data, selected from standard movies with different sequence pattern, image size and number of frames (files are available at http://www.dsi.unimo.it). In particular, the first four (Frisco, Hulahoop, Pirates, MJ) have sequence pattern I I I I I I I… (also called MJPEG), while the other three (Mei16L, Genetic, Watrerski) are IPBBBPBBBPBBPBB movies.

Table 4 reports test results. In the upper rows, parameters characterizing each MPEG movie are reported: name, total number of memory references to image data, number of frames, and frame size; in the lower, framed rows, the miss rate with the different prefetching techniques (in bold the best results). The miss rate is rather high without prefetching (higher for the first four movies, lower for the other three, due to the different sequence patterns) and is substantially reduced by all the prefetching strategies. In general the best improvement is achieved with neighbor prefetching and ada_2/OBL.

Further considerations must address performance with different cache parameters, mainly cache size, block size, and associativity degree. In particular, in the performed experiments, cache size and block size have shown to strongly influence performance, in some cases resulting in a different ranking of the compared techniques; yet, the associativity degree didn't prove to be significant, thus we selected two-way set associativity as the most common case of implemented cache. A larger block size is advantageous when the computation embeds standard "horizontal" spatial locality, in cases similar to those in which OBL shows high performance; in the other cases, a larger block size at a parity of cache size could be a limitation and even decrease performance. In Table 5 we report a comparison of the number of misses for FRISCO MPEG movie, for a cache with increasing block size, in which the number of blocks is always the same (by proportionally scaling block size and cache size).

Table 5 shows that all the prefetching techniques take advantage of a large block size, since a substantial amount of horizontal spatial locality is embedded in MPEG decoding. However, the ada_2/OBL has a peak of misses with a 32 x 64 K cache; with this cache size, the neighbor prefetching we propose in this paper outperforms all other techniques in every test performed.

Table 4. Miss rate for MPEG decoding.

	FRISCO	**HULAHOOP**	**PIRATES**	**MJ**	**MEI16L**	**GENETIC**	**WATERSKI**
NR	1.566.778	1.228.855	8.601.655	4.638.778	47.474.371	24.629.902	47.702.979
No. of frames	51	40	280	151	61	69	80
Frame size	160x128	160x128	160x128	160x128	352x240	256x192	336x208
No_Pf	5,649%	5,656%	5,628%	5,632%	2,574%	3,292%	2,721%
OBL	0,703%	0,704%	0,700%	0,701%	0,191%	1,126%	0,264%
OBVL	4,169%	3,157%	0,033%	0,033%	0,954%	1,429%	1,093%
Neighbor	0,332%	0,342%	**0,005%**	**0,005%**	**0,060%**	0,704%	**0,126%**
Adaptive	0,530%	0,531%	0,528%	0,528%	0,545%	1,048%	0,688%
ada_2/OBVL	0,425%	0,349%	0,033%	0,033%	0,323%	0,540%	0,394%
ada_2/OBL	**0,063%**	**0,063%**	0,062%	0,062%	0,107%	**0,483%**	0,137%

Table 5. Number of misses for MPEG decoding with variable cache size.

FRISCO	8 x 16 K	16 x 32 K	32 x 64 K	64 x 128 K	128 x 256 K
No_Pf	195862	88504	15926	967	487
OBL	12001	11016	3955	300	77
OBVL	144768	65324	10835	406	94
neighbor	11927	5195	**39**	**15**	**14**
adaptive	22235	8298	3177	426	149
ada_2/OBL	**1722**	**983**	2151	340	76
ada_2/OBVL	17952	6660	2028	239	85

Furthermore, since also for data-independent algorithms (see convolution) and strongly data-dependent ones (see chain code) neighbor prefetching outperforms the other techniques, it consequently seems to be an attractive strategy for multimedia image processing.

6. Prefetching costs

In previous sections, we demonstrated the efficacy and the high performance achieved by prefetching techniques for typical workloads of multimedia applications. Results seem to encourage the adoption of specific prefetching schemes, in particular neighbor or ada_2/OBL, for coping with locality and access predictability of programs working on images.

Nevertheless a complete evaluation of prefetching should deal with an analysis of costs, both in terms of hardware resources and in terms of time spent for prefetching. In this context we model prefetching as a three step process:

1) *prefetching address generation* (PFAG) for computing the address of data to be prefetched
2) *prefetching inquire cycle(s)* (PFIC) for verifying the presence in cache of data to be prefetched (lookup time).
3) *prefetching memory cycle(s)* (PFMC) for loading the data from the next hierarchical memory level to cache memory.

The time required for the first step is proportional to the number of data that the method attempts to prefetch; this quantity can be assumed as $N_{PFAG} = P \cdot Nref$, where *Nref* is the number of memory references and *P* is the number of blocks the method requires to prefetch for each memory reference. In particular *P* is equal to eight for the neighbor method and is fixed to one for the other pre-fetching schemes. The hardware resources and times associated with the PFAG step depend on the prefetching method. Fixed OBL has a negligible hardware cost and prefetch address is almost immediate to compute, since it consists of calculating the next block-aligned address only. Fixed stride methods (OBVL and neighbor) need access to a reference table where address displacements are statically stored (the displacement is the number of bytes in an image row, decided at compile time, used for computing the next row address and the eight neighbor addresses, respectively). For adaptive methods, the cost of an instruction-based prediction table must be taken into account [7] and the time for PFAG is due to the access to this internal table. However, although the time required for the prefetching address generation is not null, it can be made negligible with respect to the other two steps by an adequate amount of hardware resources.

The second step of inquiry cycle(s) consists of an interrogation of the directory cache table with a tag comparison and requires one or more clock cycles, depending on the *P* parallelism degree. In general, the total time spent for inquire cycles is proportional to *Nref* for all prefetching methods apart from for neighbor: in this case it could be eight times higher in the worst case (unless a parallel access to directory cache is provided).

The longest time which must be accounted for is due to the third step of memory access. This time is difficult to model since it is strongly affected by the memory and bus architecture and the processor-memory interface. Advanced architecture solutions, including stream buffers, victim caches, non-blocking memories can reduce PFMC time [28][18]. However, this time is mainly proportional to the number of actual prefetches (called *Npf)*, that is normally less than the maximum number of prefetches considered by the method. Therefore $Npf \leq P \cdot Nref$. The *Npf* measure gives a basic idea of the possible cost of a prefetching method since, on the average, the time spent for accessing main memory due to prefetching will be proportional to it. Rather than using the *Npf* as the performance metric, we prefer the *prefetching rate PFR* (defined as the ratio *Npf/Nref* between the number of actual prefetches and the total number of references), since it is independent of the total number of references.

Table 6 reports the prefetching rate measured for multimedia image processing programs with real images. The prefetching rate can range between 0% and 100% for all methods apart from neighbor prefetching, where it theoretically ranges between 0% and 800%, due to the number of prefetches attempted. Conversely, all tests performed on image data confirm that the neighbor prefetching rate is comparable with those of all other methods, as reported in Table 6; it is also interesting to

Table 6. Prefetching rate.

	Convolution 5x5	Chain Mpeg	MPEG Frisco	MPEG Hulahoop	MPEG Pirates	MPEG MJ	MPEG Mei16l	MPEG Genetic	MPEG Waterski
OBL	0,1678%	7,53%	4,95%	4,96%	4,94%	4,94%	2,47%	3,20%	2,60%
OBVL	0,1678%	6,24%	1,53%	2,54%	5,66%	5,66%	1,69%	2,05%	1,75%
Neighbor	0,1685%	20,50%	5,43%	5,41%	5,76%	5,76%	2,88%	6,07%	3,34%
Adaptive	0,1683%	8,06%	5,82%	5,83%	5,81%	5,81%	2,88%	3,19%	2,90%
ada_2/OBL	0,1678%	12,53%	6,29%	6,30%	6,28%	6,28%	3,31%	4,73%	3,57%
ada_2/OBVL	0,1680%	7,06%	5,93%	6,02%	6,30%	6,31%	3,06%	3,78%	3,16%

note that in MPEG-2 decoding the number of blocks prefetched by the neighbor approach is even lower than the number required with the adaptive approach.

As stated in other works (with particular relevance in [27]), only a detailed cycle-by-cycle simulation can quantify the overall time needed for prefetching, and evaluate if the advantages achieved with the reduced number of misses can compensate the prefetching overhead. However, if a certain prefetching technique assesses both a lower number of misses and a comparable number of prefetches - as is often the case for neighbor prefetching - it is a candidate for better overall performance. Moreover, the most relevant observation is that when the number of actual prefetches is comparable with that of eliminated misses an overall performance improvement is expected, since prefetching time can be intrinsically better hidden in the execution time than the fetch-on-demand imposed by a miss. This is the case of all static techniques in the majority of MPEG executions (see Tables 4 and 6); and at the same time, best performance among these techniques are expected with neighbor prefetching, which assesses the highest number of eliminated misses.

7. Conclusion

In this paper, we have presented novel hardware prefetching techniques oriented to multimedia image processing matching the 2D spatial locality. We have compared their performance with other existing hardware prefetching techniques in terms of eliminated cache misses and actual prefetches. The reference architecture used is a dedicated cache size with parameters typical of a modern L1 cache. In the paper we have shown that neighbor prefetching we propose, based on a static assumption of 2D locality, generally perform better than the other methods compared (one-block-lookahead and some variations of adaptive schemes) in terms of number of eliminated misses. At the same time in most cases the number of actual prefetches introduced by neighbor prefetching is comparable with or even less than those carried by the other prefetching techniques, and therefore the prefetching costs can not be considered a serious drawback of neighbor prefetching. In the case of the static prefetching techniques, the number of prefetching issued is comparable to the number of avoided misses, and this is positive since prefetches can be more easily overlapped to execution than fetches on miss. Among the static techniques, neighbor prefetching is able to eliminate the greatest amount of misses of all and thus is promising of overall performance improvement.

References

[1] ACME Cache Simulator, http://atanasoff.nmsu.edu/~acme/acs.html.

[2] Baglietto, P. and Maresca, M. and Migliardi, M. and Zingirian, N., "Image processing on high performance RISC systems", Proceedings of the IEEE, v. 84, n.7, 917-925, 1996.

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

[4] Chen, T.F. and Baer, J.L., "Effective hardware-based data prefetching for high-performance processors", IEEE Trans. on Comp., v. 44, n. 5, 1995, pp. 609-623.

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

[6] Cucchiara, R. and Piccardi, M., "Exploiting Image Processing Locality in Cache Pre-fetching", Proc. of 5th International Conference on High Performance Computing HiPC '98, Dec. 17-20 1998.

[7] Cucchiara, R. and Piccardi, M. and Prati, A., "Exploiting Cache in Multimedia" Proc. of IEEE International Conf. on Multimedia Computing and Systems ICMCS 99, v. 1, 1999.

[8] Diefendoff, K. and Dubey, P.K., "How multimedia workloads will change processor design", IEEE Computer, Sept 1997.

[9] Eickemeyer, R. J., Vassiliadis, S., A load instruction unit for pipelined processor, IBM Journal of Research and Development, vol. 37, pp. 547-564, July 1993.

[10] Freeman, H., “On the encoding of arbitrary geometric configurations”, IRE Trans. Electron. Comput., vol EC-10, pp. 260-268, 1961.

[11] Gall, D., “MPEG: a video compression standard for multimedia application”, Comm. of ACM v. 34, n. 4, 1991, pp. 46-58.

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

[13] Haralick, R.M. and Shapiro, L.G. “Computer and robot vision”, v.1, Addison Wesley, 1992.

[14] Hennessy, J. and Patterson, D., “ Computer Architecture: A Quantitative Approach”, Morgan Kaufmann, 1990.

[15] Irlam, G., “Spa”, personal communication, 1992.

[16] ISO/IEC DIS 14496-2 “Information technology -- Coding of audio-visual objects -- Part 2: Visual”.

[17] Johnson, T.L. and Merten, M.C. and Hwu, W.W., “Run-time spatial locality detection and optimization”, Proc. of 30th International Symposium on Microarchitecture, Dec. 1-3 1997, pp. 57-64.

[18] Jouppi, N., “Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers”, Proc. 17th Int'l Symp. Computer Architecture, pp. 364-373, May 1990.

[19] Joseph, D. and Grunwald, D., “Prefetching using Markov Predictors”, IEEE Trans. on Computers, Special Issue on Cache Memory and Related Problems, Vol. 48, No. 2, Feb. 1999, pp. 121-134

[20] Kanedo, T. and Okudaira, M., “Encoding of arbitrary curves based on the chain code representation”, IEEE Trans. Commun., vol. COM-33, pp. 697-707, 1985.

[21] Katsaggelos, A.K. and Lisimachos, P.K. and Meier, F.W. and Ostermann, J. and Schuster, G.M., "MPEG-4 and Rate-Distortion-Based Shape-Coding Techniques" Proc, of the IEEE, Vol. 86, no. 6, 1998.

[22] Koplowitz, J., “On the performance of chain codes for quantization of line drawings”, IEEE Trans. on PAMI, vol. 3, pp. 180-185, 1981.

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

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

[25] Smith, A.J., “Cache memories”, Computing Surveys, v. 14, n. 3, 1982, pp. 473-530.

[26] Soderquist, P. and Leeser, M., “Memory traffic and data cache behavior of an MPEG-2 software decoder”, preprint for ICCD ’97, 1997.

[27] Tse, J. And Smith, A.J., “CPU cache prefetching: timing evaluation of hardware implementation”, IEEE Trans. on Comp., v. 47, n. 5, 1995, pp. 509-526.

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