Multimedia Tools and Applications, 20, 159–177, 2003

Improving Data Prefetching Efficacy in Multimedia Applications

RITA CUCCHIARA rita.cucchiara@unimo.it
ANDREA PRATI prati@dsi.unimo.it
Dipartimento di Scienze dell'Ingegneria, Università di Modena e Reggio Emilia, Via Vignolese, 905-Modena, Italy

MASSIMO PICCARDI mpiccardi@ing.unife.it
Dipartimento di Ingegneria, Università di Ferrara, Via Saragat, 1-Ferrara, Italy

Disk followed

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, which allows the pre-loading of data in the cache before they are referenced. However, existing hardware prefetching approaches unable to exploit the potential improvement in performance, 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 MPEG-2 decoding and encoding, convolution, thresholding, and edge chain coding.

Keywords: cache performance, hardware prefetching techniques, multimedia benchmark, MPEG decode, image processing

1. Introduction

Multimedia is one of the fastest growing key businesses of the computer market both in terms of software tools and applications, and hardware and computing platforms. Multimedia technology includes communication and processing support for managing text, images, graphics, binary data, audio, video and animation. These types of information exhibit very different requirements in the exploitation of computer resources and call for specific approaches for optimizing the computation and communication performance. However, the incredible spread of multimedia has been made possible by the design of new computer architecture elements, specifically oriented to efficiently handle and process these multimodal information types. As a matter of fact [5], differently from the common trend in designing high performance general-purpose computers, the multimedia market imposed on the hardware manufacturers the integration of some novelties in standard PCs too. Answers have focused on various items: first, the integration of multimedia peripherals (DVD, high-performance disks, CD-ROMs, frame-grabbers, . . .), followed by the addition of multimedia processors [20] or high performance DSPs, in order to handle images and sounds concurrently with the normal CPU activity; finally, the enlargement of the instruction set architecture and the re-design of the internal computational units of general-purpose CPUs

(such as Pentium MMX, HP MAX2, Ultrasparc VIS) in order to improve the parallel computation on image pixels [9]. Nevertheless, other critical elements are in some way still neglected. In particular, as is well emphasized in [9], traditional cache memory architecture, which played a prime role in the explosion of PCs power for general purpose applications, is partially insufficient if not useless for many multimedia tasks. Cache architecture has been optimized for matching the spatio-temporal locality of scalar and vector data, with adequate parameters (size, associativity, placement and allocation policies); these optimizations are satisfactory for general-purpose data and also some forms of multimedia data, such as audio data which are usually mapped on 1-d arrays and exhibit conventional locality. Conversely, a similar optimization for access to image data has not yet been achieved.

In this context, this paper aims to analyze cache architecture behaviour, comparing advanced techniques for cache performance enhancement and proposing new architectural solutions for exploiting cache memory in multimedia image applications. Among the various techniques used to improve cache memory performance, *prefetching* has been one of the most studied and apparently promising [4]. Data prefetching aims to provide the first level of the memory hierarchy with data before they are requested by the current instruction, so as to eliminate or limit the waiting time for accessing data in slower storage levels. Prefetching techniques can be classified mainly according to their software or hardware implementation [9]. Software techniques modify the program code by the insertion of prefetch instructions at compile time. A drawback in this approach is that it limits program portability on different architectures since prefetch instructions depend on the specific architecture. In addition, the placement of prefetch instructions is based on a compile-time prediction of memory references and thus software prefetching is less suitable for data-dependent programs. Instead, hardware techniques do not require modification of the program code, while at the same time exploiting a run-time history of memory references, thus allowing better data-dependent prediction. The main drawback of hardware prefetching is that it requires hardware resources and all functionalities must be performed at run-time, where speed requirements are mandatory so as not to compromise cache effectiveness. In this work we will focus on hardware prefetching techniques since they are not invasive on software and the current level of circuit integration allows practical feasibility.

In this paper we present experimental results achieved in the execution of a multimedia benchmark with different hardware prefetching techniques. The benchmark includes a variety of image and video-based multimedia programs since they are the most critical from the point of view of memory access and in order to cover a wide spectrum of memory access schemes. Performance evaluation is based on the of miss rate as performance metric, but also takes into account the prefetching costs. The experimental results allow us to state that all multimedia programs working on images can receive important benefits from a substantial amount of cache prefetching (achieving an improvement of one or two magnitude orders in terms of cache misses); among hardware prefetching schemes, we show that the basic *always prefetch* is not always highly performing for image processing and video management. Instead, adaptive methods, even though more complex to implement, have a better behaviour in general. Moreover, the new prefetching technique we propose, the *neighbour prefetching*, outperforms the other methods in many cases.

The rest of the paper is structured as follows: Section 2 presents related works on cache improvement for multimedia and hardware prefetching techniques. Section 3 defines the suite of multimedia image processing algorithms used for the tests. In Section 4, the hardware prefetching strategies are presented, including novel and original techniques we propose aimed at exploiting the spatial locality. Section 5 describes the methodology used for performance measurement, also covering prefetching costs. In Section 6 experimental results and performance analysis are presented for a reference cache architecture, while the effects of changing the cache size are reported in Section 7. An evaluation of prefetching costs is presented in Section 8. The conclusion presents final considerations on the benefits of the presented approach.

2. Related works

Our work relies on a very intense research activity covering two main aspects: first, the definition of multimedia benchmarks representative of multimedia workload; second, specific architectural improvements for multimedia. With respect to the first issue, it is possible to find several recent works. They cannot be considered well assessed yet, since multimedia applications are still in rapid evolution. Existing multimedia benchmarks include video compression (MPEG), audio processing, and imaging or image processing programs [21]. However, they are oriented to overall performance analysis and do not focus on memory performance specifically. The only related works addressing memory performance for multimedia use as benchmark MPEG or video compression only [9, 20, 27]. In this work, we extend these benchmarks with image processing and analysis programs, currently exploited in many multimedia applications such as MPEG-4 and content-based retrieval from image and video databases, in order to cover a larger variety of memory access schemes.

With respect to the second issue, many related works propose architectural improvement for multimedia and address specifically cache memory performance. Kuroda and Niscitani in [20] present a complete survey on multimedia processors. They state that current cache architectures are not adequate for multimedia workload and suggest the adoption of software prefetching, given the static predictability of data access. However, in the present work we demonstrate that some classes of multimedia programs are not statically predictable but are data dependent. In these cases only a run-time hardware solution can achieve high performance.

Many of these recent works have focused on hardware prefetching techniques, since recent improvements in hardware technology have rendered more feasible the run-time functionalities required by hardware implementation [2, 25, 27]. The basic technique is known as *One-Block-Lookahead*, [23] (called *always prefetch* by the same author in a more recent work [25]). 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 [16]). OBL prefetching is ideal to catch the locality of a program that refers sequentially the elements of an array, such as an audio application. In this paper, instead, we test OBL performance in image and video applications and we prove that it is not a satisfactory approach since the memory access scheme is bi-dimensional.

Several more complex approaches have been proposed, taking into account different forms of adaptivity in order to achieve a better predictability than that 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 [2] 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). Zucker et al. [27] proves the effectiveness of adaptive hardware prefetching on MPEG algorithms; the authors propose the use of a Stride Prediction Table (SPT) together with a multiple-way stream memory. Some authors [2, 10] propose to filter isolated strides, that are for instance frequent in MPEG decoding (see [24]). Therefore, in the present work we test the performance of both basic and filtered adaptive schemes. Tse and Smith [25] propose a system-level analysis of prefetching impact on system performance by way of an accurate cycle-by-cycle execution simulation, measuring the actual time required by line fill cycles and prefetches. However, time measurements depend on many system parameters, including the number of cache misses, the number of prefetches issued, the amount of overlap between the prefetching activity and execution, and many others. In this paper, the main goal is to show the ability of a specific prefetching algorithm to exploit the form of locality embedded in image processing programs. Therefore, we report results in terms of number of eliminated misses, like in several other recent works [3, 13, 27].

Other works related to our research are those on split caches [12, 22]. 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 [6, 12, 22]. In line with this trend, we propose the adoption of a special cache for image data together with a dedicated prefetching policy.

3. The multimedia image processing benchmark

In the case of multimedia, benchmarks must include massive amounts of video, audio and image processing, since there is general evidence that these three areas dominate multimedia programs. However, a generalized benchmark has not been accepted yet. This may be due to the fact that the computational requirements of video, audio and image processing differ substantially and that the spread of these processing styles varies significantly between applications. In the scope of this work, we aimed to define a benchmark useful for cache performance evaluation over a variety of image and video processing algorithms, since their memory address schemes differ from common scalar or 1-D array computation (typical for instance of audio processing) which consequently results in poor performance on traditional cache organizations. To this end, we focused on a realistic application including video

conferencing and image manipulation, while at the same time covering a reasonably large spectrum of memory address schemes from commonly used algorithms. The algorithms included are:

(a) `MPEG2_dec`: MPEG-2 decoding for image decompression.
(b) `MPEG2_enc`: MPEG-2 encoding for image compression.
(c) `convo`: image convolution for image filtering.
(d) `thresh`: image thresholding for image binarization.
(e) `chain`: chain code computation for extracting contour information.

The video conferencing part involves on-line compression/decompression (`MPEG2_dec`, `MPEG2_enc`), while image manipulation (`convo`, `thresh`, `chain`) is performed on the decompressed frames in order to extract a minimum set of geometrical properties from objects. MPEG-2 decoding is widely used in multimedia applications both as support for film playing and for decompressing videos on line [4, 5, 11, 19, 20]. MPEG frames are of three different types: I (only spatial compression in a JPEG style), P (forward-predicted), and B (forward-backward predicted). Typically MPEG sequences are periodical repetitions of a same type pattern, like for instance IPBBPBBPBBPBB; when the pattern is I I I I I I I, the sequence is usually called MJPEG, representing just a sequence of JPEG compressed images. MPEG processing is based on blocks (8×8 pixels) and macroblocks (16×16 pixels), thus imposing a strong 2D data locality. MPEG-2 decoding typically carries a not negligible number of cache misses [3, 24]: 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 memory access scheme, and therefore the number of misses, depends on the sequence pattern; for this reason in the benchmark we have considered MPEG videos differing in the sequence pattern, whose main characteristics are summarized in Table 1. In MPEG-2 encoding, the major steps are inverse functions of those of MPEG-2 decoding. However, MPEG-2 encoding has a much higher computational load per frame (about one order of magnitude, this being the reason why it is often implemented in hardware on modern acquisition boards).

Convolution is a basic algorithm for image processing [1, 6, 7]; it consists of processing each image pixel by convolving its bidimensional neighborhood with a coefficient mask. Pixels are usually evaluated in raster-scan mode, and the memory access scheme is not data dependent; a substantial amount of strictly 2D locality is embedded. Thresholding is the simplest example of raster-scan image processing, differing from convolution since

Table 1. Features of MPEG videos.

	Frisco	Hulahoop	Pirates	Mj	Flower G.	Genetic	Waterski
NR	1566780	1228860	8601660	4638780	47474400	24629900	47703000
Number of frames	51	40	280	151	61	69	80
Frame size	160×128	160×128	160×128	160×128	352×240	256×192	336×208
Type	IIIII	IIIII	IIIII	IIIII	IPBBPBB	IPBBPBB	IPBBPBB

no pixel reuse is performed. The Chain code computation is included as a typical data-dependent program. 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 by 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 [18]. 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 spatial locality, and its memory access scheme cannot be predicted a priori. Since computation is data dependent, prefetching performance varies with the input image. Yet, 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 [5, 11, 19].

4. The hardware prefetching techniques

Hardware cache prefetching can be suitably adopted for an image multimedia workload if it is able to exploit two main properties of multimedia image processing, namely (i) *locality* of image data access and (ii) *access predictability* of typical algorithms. In this work we have compared different prefetching schemes, including simple schemes like one-block-lookahead, basic adaptive prefetching and other original schemes designed to appropriately mirror these properties, and measured them on a multimedia workload. The compared schemes can be divided into two categories, namely static and adaptive prefetching schemes.

Static prefetching schemes are:

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

while *adaptive prefetching schemes* include:

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

In the following subsections we further analyze these categories.

4.1. Static prefetching schemes

The basic assumption of these schemes is that many algorithms define an a priori-known memory access scheme and exhibit a strong spatial locality. Typical raster-scan algorithms for image pixel-by-pixel manipulation are mainly forward-scan and make access to a limited neighborhood of the current pixel. Moreover, even more complex programs using images and video exhibit a significant raster-scan component in the pixel access. For this reason,

Table 2. Prefetching address.

	Current block address	Prefetch address
No_Pf	$A0$	
OBL	$A0$	$A0 + 1$
OBVL	$A0$	$A0 + NBrow$
neighbor	$A0$	$A0 + 1$, $A0 - 1$
		$A0 + NBrow$, $A0 + NBrow + 1$, $A0 + NBrow - 1$
		$A0 - NBrow$, $A0 - NBrow + 1$, $A0 - NBrow - 1$
adaptive	$A0$	$A0 + S$
ada_2/OBVL	$A0$	$A0 + S$ if S=S′
		$A0 + NBrow$ otherwise
ada_2/OBL	$A0$	$A0 + S$ if S=S′
		$A0 + 1$ otherwise

Ix,Iy row and column image size (in pixel); B block size; b bytes per pixel; N Brow $= bIx/B$ number of blocks covering an image row; S current stride; S' previous stride.

we aim to exploit this feature to increase the overall cache efficiency for these multimedia algorithms. Static prefetching techniques differ in the selection of lines to be prefetched.

One-block-lookahead is the most commonly adopted prefetching scheme: 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 of the basic line; for this same reason it also proves to be very effective on raster-scan image processing algorithms. Furthermore, it is very simple to implement in hardware. With the OBL approach, for every access to the block address named $A0$, the block address $A0 + 1$ (see Table 2) is loaded into the cache (if not already present).

Conversely, *OBVL prefetching* aims to exploit 2D spatial locality [6]. This kind of locality arises since a high probability exists of accessing 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 central memory with a displacement given by the image size scaled by the pixel size in byte [7]. In OBVL prefetching the line just below the one currently referenced (referred to by the $A0 + NBrow$ block address in Table 2) is prefetched if absent from the cache. The rationale is that the unavoidable horizontal spatial locality is partially covered by the (large) block size, while the vertical direction is not explored. Thus, the following vertical block may be more useful than the following horizontal block prefetched by OBL.

We introduce further exploration of 2D spatial locality by proposing a new approach that we call *neighbor prefetching*, where all blocks neighboring the currently referenced block (called 8-connected blocks) are checked for presence in the cache, and prefetched if absent. Thus, in addition to blocks considered by both OBL and OBVL techniques, neighbor prefetching technique verify if another six blocks have to be prefetched, with the block address shown in Table 2. The cost of hardware implementation of neighbor

prefetching is higher than for OBL, since eight addresses have to be computed; however, the image size and the *NBrow* displacement are fixed at declaration time. Thus, given the current referenced block address $A0$, the addresses of the 8-connected blocks can be quickly computed by adding or subtracting a fixed stride of *NBrow* (± 1). 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 obtained by the other techniques.

4.2. Adaptive prefetching schemes

Techniques described in the previous subsection are all static in the prevision of the block(s) to be prefetched, and primarily aim to explore an a-priori probability of access, intrinsic in multimedia image processing. Nevertheless, also adaptive prefetch could just as appropriate in this kind of task. The main justification is that often the memory access scheme is not predictable a priori, but is heavily data dependent. In this case, a dynamic selection of the prefetching address could prove highly profitable.

The basic adaptive prefetching exploits a hardware instruction-based Stride Prediction Table (SPT) [27]: for each instruction making a data reference, the difference between the current and previous data addresses is computed (called *stride* S); the stride is then added to the current address to form the prefetch address. The implementation make use of an SPT with 128 entries and LRU replacement policy, which is tested on some MPEG-based programs [27] with good results. We use a similar SPT in our tests and refer to this approach with the name *adaptive*. The address computed for the block to be prefetched is shown in Table 2.

In addition, similarly to [10], we consider 2-step adaptive techniques (also called delta2 filters) trusting the stride S only if it is equal to the previous one: if a memory instruction is used three times consecutively, say t_i, $t_{(i-1)}$, $t_{(i-2)}$, and $A(t_i)$ is the memory block address dynamically computed at that time, we compute the current stride $S = S(t_i) = A(t_i) - A(t_{(i-1)})$ and the previous stride $S' = S(t_{(i-1)}) = A(t_{(i-1)}) - A(t_{(i-2)})$. If $S = S'$, the computed stride is used for predicting prefetch address and S is updated. This is intended to filter isolated strides from being used for prefetch prediction. As stated in [10], in the case of loops, a single stride algorithm will mispredict the jump and the next address after the jump, while a 2-step adaptive will mispredict only the former and will give a correct prediction for the latter. In some proposals, when the last two strides are not equal, no prefetch is attempted; instead we test the opportunity of providing prediction on a static probability assumption: OBL for the technique called *ada_2/OBL*, and OBVL prefetching for the *ada_2/OBVL* approach (see Table 2). This choice is due to the fact that in image processing prefetching is generally more convenient than non prefetching in any case.

5. Methodology

The reference cache architecture on which the prefetching techniques have been tested is a cache for image data with 2-way set associativity, 32 bytes block length and a 64 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 derived version of the trace-driven ACME simulator [14] 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, i.e. we assume unlimited memory bandwidth and null latency. Although real memory organizations differ from this scheme, these assumptions are not limiting for the scope of this paper since the different hardware prefetching techniques we aim to compare are tested at a parity of conditions. Furthermore, the impact of prefetching costs is analized.

5.1. Performance metrics

Basic performance metrics for cache memories are the number of cache misses and the miss rate, which is a more general measure since it does not depend on the total number of references. In the following, we call *NM* the number of misses without prefetching (*No_Pf*) and, for each *i*th pre-fetching method, NMP_i its related number of misses and MR_i the corresponding miss rate. Starting from the number of misses, we also define the *efficacy* of the *i*th prefetching method as the ratio

$$\eta_i = \frac{NM - NMP_i}{NM} \tag{1}$$

Efficacy ranges between 0 and 1 and tends toward 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. Efficacy is a relative measure that depends neither on the absolute number of references nor on the miss rate. *NMPi*, *MRi*, and η_i values are used in the next section to compare the performance of the considered hardware prefetching techniques. In particular, we use $\eta_i\%$ to express percentual efficacy (varying from 0% to 100%). However, the prefetching activity in general can affect the execution time since it occupies the bus and introduces 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 also discussed in the next section. In the following section, we report only the misses occurring when image data are accessed. This choice is due to the fact that the impact of accesses to image data in multimedia and image processing applications is significantly more critical than that to scalar data or other data. Table 3 reports a comparison of the number of memory accesses to image data as opposed to the ones to scalar and other data (called 2D and scalar references) for the benchmark algorithms. The workload was defined as follows: MPEG algorithms have been performed on videos of Table 1, `convo` and `thresh` on a 512×512 image, and `chain` on a 352×240 frame of the Flower G. video; obviously, values in Table 3 will vary with a different workload consequently. The related miss rates have been measured by using two separate caches like the reference one, without any prefetching approach. Scalar references constitute the majority in many programs, but conversely, the scalar miss rate is nearly zero in any considered case. This means that the

Table 3. Comparison between scalar and 2D miss rate.

	Nr. ref. 2D	Nr. miss 2D	Miss rate 2D (%)	Nr. ref. scalar	Nr. miss scalar	Miss rate scalar (%)
`convo`	19504700	16370	0.0839	6538300	1406	0.0215
`thresh`	520201	8161	1.5688	1855000	1406	0.0215
`chain`	1189740	61159	5.1405	20466400	1819	0.0089
`MPEG2_dec` Frisco	1566780	15926	1.0165	18649400	2190	0.0117
`MPEG2_dec` Genetic	24629900	393978	1.5996	57106000	2214	0.0039
`MPEG2_dec` Hulahoop	1228860	12740	1.0367	17065100	2190	0.0128
`MPEG2_dec` Mj	4638780	44376	0.9566	70214500	2191	0.0031
`MPEG2_dec` Flower G.	47474400	614803	1.2950	85751700	2206	0.0026
`MPEG2_dec` Pirates	8601660	81020	0.9419	117003000	2197	0.0019
`MPEG2_dec` Waterski	47703000	671180	1.4070	98544000	2216	0.0022
`MPEG2_enc` Hulahoop	562546000	375487	0.0667	260050000	297538	0.1144

scalar locality is already perfectly caught by a conventional cache and that no further effort is justified to reduce it anymore (especially if interference with 2D references is avoided, as we grant by using a separate cache for image data). Instead, the 2D miss rate is two orders of magnitude greater than the scalar one on average. Notice that even small miss rates result in a substantial percentage increase in memory latency, since line filling from main memory is accomplished in a time typically an order of magnitude greater than that of a cache hit.

This proves that the locality of 2D references is not caught by conventional caches as accurately as other data. Therefore, it should be possible to use a separate cache for image data, with associated prefetching techniques, in order to achieve better performance while granting less interference with data of different locality.

5.2. Analysis of prefetching costs

Nevertheless, a complete evaluation of prefetching should include 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.
(3) *prefetching memory cycle(s)* (PFMC) for loading the data from next hierarchical memory level to cache memory.

The number of addresses to be generated is the same as the number of blocks that the method attempts to prefetch, which is equal to eight for the neighbor method and is fixed to one for the other prefetching schemes. The costs and times associated with the PFAG

step depend on the prefetching method and the address generation hardware. 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 computation delay is associated with the query on this internal table. Although hardware costs are not null, the time spent for the prefetching address generation can be made negligible with respect to the other two steps.

The second step, PFIC, consists of a lookup in the cache directory with a tag comparison and usually requires one clock cycle. The neighbor prefetching imposes a very high lookup pressure with respect to the other methods, since it issues eight lookups in the cache for each memory reference but it can be substantially limited by a multi-ported tag directory.

However, the longest time which must be accounted for is due to PFMC, the third step of memory access, usually ranging in the order of tens of clock cycles. This time is strongly affected by the memory and bus architecture and the processor-memory interface. Advanced architecture solutions, including victim caches or stream buffers between the cache and the next memory level may reduce the PFMC time [17, 27]. However, this time is mostly proportional to the number of prefetches that are actually issued (called *Npf*), which is normally far less than the number of prefetches attempted, i.e. than the number of cache lookups, since prefetches are issued to memory only when the data is not already in the cache. The *prefetching rate PFR*, that is the ratio between the number of actual prefetches and the total number of references $PFR = Npf/N\text{ref}$ gives a basic idea of the average time cost of an assigned prefetching method, and therefore will be used for comparison in Section 7.

6. Performance results

The goal of this section is to demonstrate the benefits of adopting prefetching in the various video and image processing application included in our benchmark.

6.1. Raster scan programs: Convo *and* Thresh

`Convo` and `Thresh` are representative programs of the basic raster-scan data access with or without near-neighbor computation. Due to their high spatial locality, traditional cache architecture is adequate enough for achieving a relatively low miss rate: 0.08% for `Convo` and 1.56% for `Thresh` (see Table 4) are due to compulsory miss only; convolution has a lower miss rate since it exhibits a substantial amount of temporal locality due to the neighbourhood computation. Although for these programs prefetching is not strictly necessary, it can be suitably exploited since miss rate is practically nullified by whichever prefetching tested (the efficacy is higher than 99.79% in all cases). The strong regularity of the algorithm and its uniform 2D spatial locality makes the number of misses almost *invariant* with the prefetching strategy. Obviously, the classic OBL prefetching performs well since the data access follows the row-by-row data store direction. In general, all static methods show

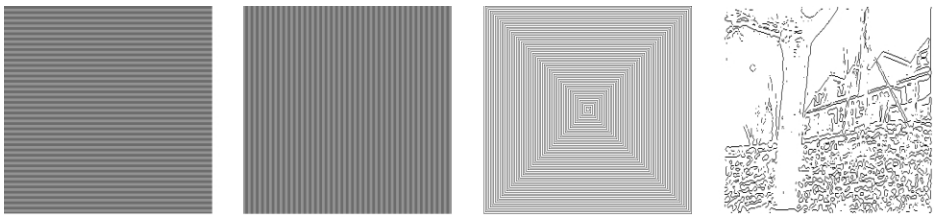
Table 4. Efficacy for `Convo` and `Thresh` (64K × 32 cache size).

Convolution 5 × 5 Nref = 19504949				Thresholding Nref = 520201			
	NMP_i	MR_i %	η_i %		NMP_i	MR_i %	η_i %
No_Pf	16370	0.08393	0	No_Pf	8161	1.5688	0.00
OBL	6	0.00003	99.98	OBL	2	0.0004	99.98
OBVL	32	0.00016	99.80	OBVL	17	0.0033	99.79
neighbor	2	0.00001	99.99	neighbor	2	0.0004	99.98
adaptive	34	0.00017	99.79	adaptive	3	0.0006	99.96
ada_2/OBVL	19	0.00010	99.88	ada_2/OBVL	3	0.0006	99.96
ada_2/OBL	19	0.00010	99.88	ada_2/OBL	2	0.0004	99.98

effective miss elimination, but the neighbor method in particular is able to eliminate all misses (apart from the first image reference) since it mirrors the data access locality. As the data access is easily predictable, adaptive techniques demonstrate high efficacy, too.

6.2. *Propagative programs:* Chain

Chain code is an example of propagative and data dependent programs since the computation follows the edge direction of the visual object in the images. In order to stress the data dependency effects, first we report results for some limit cases of synthetic images with horizontal, vertical, and both horizontal and vertical edges (figure 1(a)–(c)). Tables 5 and 6 report prefetching results for synthetic images; let us first consider static methods. In the image in figure 1(a) data access is mainly in the horizontal direction as in raster-scan programs; thus a basic technique like OBL works well in decreasing the miss rate from 1.666% to 0.026%, achieving a percentual efficacy of 98.41%. Instead, OBVL prefetching, favouring vertical direction, is not able to eliminate a substantial amount of misses. When the image contains many vertical edges like the one in figure 1(b), chain code has a high miss rate of 15.85% (no locality with standard caches is exploited); OBL shows unsatisfactory performance (MR = 13.3%), while OBVL nearly eliminates all misses. Thus, these two



(a) (b) (c) (d)

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

Table 5. Efficacy for Chain on limit cases.

	Chain horizontal.img Nref = 983271			Chain vertical.img Nref = 981744		
	NMP_i	MR_i %	η_i %	NMP_i	MR_i %	η_i %
No_Pf	16385	1.6664	0.00	155629	15.852	0.00
OBL	260	0.0264	98.41	131073	13.351	15.78
OBVL	3874	0.3940	76.36	306	0.0312	99.80
neighbor	4	0.0004	99.98	260	0.0264	99.83
adaptive	275	0.0280	98.32	8694	0.8856	94.41
ada_2/OBVL	275	0.0280	98.32	278	0.0283	99.82
ada_2/OBL	252	0.0256	98.46	8918	0.9084	94.27

Table 6. Efficacy for Chain on one more limit case.

	NMP_i	MR_i %	η_i %
Chain spiral.img Nref = 980471			
No_Pf	82418	8.4060	0.00
OBL	62166	6.3404	24.57
OBVL	32812	3.3466	60.19
neighbor	5	0.0005	99.99
adaptive	5310	0.5416	93.56
ada_2/OBVL	3587	0.3658	95.65
ada_2/OBL	4643	0.4735	94.37

static techniques exhibit their complementary behavior. And as it is easy to foresee, both OBL and OBVL prefetching are disrupted by a pattern like the one in figure 1(c), both demonstrating poor performance. However, in all three cases, neighbor prefetching achieves best performance, with nearly total miss elimination, resulting in a miss rate of 0.0004%, 0.026% and 0.0005% only. The regularity of patterns in the synthetic images (carrying a repetitive address stride) also allows dynamic methods based on adaptivity to achieve very high miss elimination.

Figure 1(d) shows a real image extracted by the *Flower Garden* MPEG video after an edge detection step (which is based on convolution); chain-code results on this image are reported in Table 7. Due to unpredictable access direction following real edges, chain-code has a relatively high miss rate without prefetching (6.8%). The miss rate does not improve satisfactorily with OBL and OBVL, nor with adaptive methods that are useful only if edges are straight (rare in real images). In this case the neighbor method is still the best choice (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.

Table 7. Efficacy for Chain on real image.

	NMP_i	MR_i %	η_i %
FlowerG.img Nref = 1189740			
No_Pf	61159	11.093	0.00
OBL	15231	1.2802	75.10
OBVL	21237	1.7850	65.28
neighbor	232	0.0195	99.62
adaptive	1972	0.1658	96.77
ada_2/OBVL	1446	0.1215	97.63
ada_2/OBL	1277	0.1073	97.91

6.3. MPEG2_dec *and* MPEG2_enc *programs*

In MPEG-2 decoder the access to image data is mainly local (in blocks of 8×8), but in part the access is random between different blocks, as results also from the address traces. The locality of MPEG and the efficiency of standard caches have been evaluated in detail in [24]. Table 8 reports test results. The first four MPEG movies have the same frame size and JPEG-style sequence pattern, while the other three have standard MPEG sequence patterns. The first four movies vary for image content and number of frames; however, in our experiments we verified that there is very little dependency on the number of frames. Therefore, the different miss rates at a parity of prefetching techniques must be attributed to data dependency. The miss rate is not negligible high without prefetching and diminishes with the use of each prefetching strategy. In general, best improvements were obtained by the adaptive methods, and among them, by the 2-step adaptive with OBL. Nevertheless since the near-neighbor nature of computation is dominant overall, the neighbor prefetching, too, is particularly suitable for MPEG decoding; it provides the best results in most cases. A similar performance is shown in Table 9 for an MPEG-2 encoder of one movie only. The encoder program needs a much higher number of memory references (10^9 instead of 10^6) but thanks

Table 8. Miss rate for `MPEG_dec` with 32 bytes/line and 64 KBytes.

	Frisco (%)	Hulahoop (%)	Pirates (%)	Mj (%)	Flower G. (%)	Genetic (%)	Waterski (%)
No_Pf	1.016	1.037	0.942	0.957	1.295	1.600	1.407
OBL	0.252	0.257	0.233	0.237	0.161	0.345	0.211
OBVL	0.692	0.579	0.008	0.009	0.451	0.646	0.539
neighbor	**0.002**	0.137	**0.002**	**0.002**	**0.051**	0.252	**0.106**
adaptive	0.203	0.205	0.191	0.194	0.237	0.341	0.312
ada_2/ OBVL	0.129	**0.119**	0.012	0.012	0.147	0.166	0.208
ada_2/ OBL	0.137	0.139	0.129	0.130	0.057	**0.089**	0.140

Table 9. Efficacy for Encoding on hulahoop.mpg.

	NMP_i	MR_i %	η_i%
Encoder on hulahoop.mpg Nref = 562546000			
No_Pf	375487	0.0667	0.00
OBL	95028	0.0169	74.69
OBVL	73588	0.0131	80.40
neighbor	15313	0.0027	95.92
adaptive	52856	0.0094	85.92
ada_2/OBVL	18492	0.0033	95.08
ada_2/OBL	16476	0.0029	95.61

to temporal locality it has a very low miss rate. Nevertheless, also with encoder, prefetching should be adopted to obtain a significant improvement: the number of misses decreases from 375487 with no_pf to 15313 with neighbour prefetching (95.9% misses eliminated). Further considerations must address performance with different cache parameters, mainly cache size, block size, and degree of associativity. High cache associativity could decrease conflict misses; however, increasing the associativity degree to more than 2-way did not prove to be significant in the experiments; this is also confirmed in [26]. 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 other cases, a larger block size at a parity of cache size, could be a limitation and even decrease performance. In Table 10 we report, as an example, a comparison of the number of misses of `MPEG2_dec` on one of the MPEG videos 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). It proves that the couple cache size/block size strongly influence performance: miss rate decreases more than linearly with the increase of the cache size/block size.

Prefetching techniques demonstrate evident advantages in all cases, but in some cases result in a different ranking of the compared techniques. Table 10 shows that all the prefetching techniques are improved by a large block size, since a substantial amount of horizontal

Table 10. Number of misses for `MPEG2_dec` with variable cache size.

Hulahoop	8 × 16 K	16 × 32 K	32 × 64 K	64 × 128 K	128 × 256 K
No_Pf	153622	69501	12740	967	487
OBL	9416	8651	3158	300	77
OBVL	84458	38800	7110	428	158
neighbor	9359	4206	1678	**15**	**14**
adaptive	17244	6521	2524	405	149
ada_2/OBVL	10836	4286	**1460**	335	149
ada_2/OBL	**1353**	**774**	1703	326	76

spatial locality is embedded in MPEG decoding. However, the 2-step adaptive with OBL has a peak of misses with a 32 × 64 K cache (perhaps due to pollution); with this size and higher ones, neighbor prefetching outperforms all other techniques. In every test performed with large block size neighbor prefetching has the best behavior. Furthermore, since neighbor prefetching also outperforms the other techniques for data-independent algorithms (see convolution) and strongly data-dependent ones (see chain code), it consequently seems to be an attractive strategy for multimedia image processing.

7. Experimental results on prefetching rates

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 some aggressive prefetching schemes, in particular neighbor or 2-step adaptive with OBL, for coping with locality and access predictability of programs working on images.

Tables 11 and 12 report the prefetching rate measured for multimedia image processing programs with real images. In `convo` and `thresh`, *PFR* is similar for all techniques. In `MPEG2_enc`, adaptive techniques show a slightly higher number of issued prefetches. In

Table 11. Prefetching rate.

	Convolution 5 × 5 (%)	Chain code FlowerG (%)	Thresholding (%)	Encoder Hulahoop (%)
OBL	0.0839	0.8617	1.569	0.0505
OBVL	0.0839	0.7889	1.569	0.0548
neighbor	0.0843	1.2964	1.577	0.0667
adaptive	0.0842	1.0425	1.568	0.0794
ada_2/OBVL	0.0840	0.8609	1.572	0.0755
ada_2/OBL	0.0839	0.9387	1.569	0.0758

Table 12. Prefetching rate `MPEG_dec`.

	Mpeg Frisco (%)	Mpeg Hulahoop (%)	Mpeg Pirates (%)	Mpeg MJ (%)	Mpeg Flower G. (%)	Mpeg Genetic (%)	Mpeg Waterski (%)
OBL	0.758	0.774	0.702	0.713	1.15	1.41	1.26
OBVL	0.330	0.447	0.937	0.952	0.855	0.996	0.902
neighbor	1.07	0.945	0.986	1.00	1.30	2.19	1.70
adaptive	1.36	1.37	1.30	1.31	1.42	1.62	1.55
ada_2/OBVL	1.44	1.46	1.47	1.49	1.50	1.81	1.65
ada_2/OBL	1.42	1.44	1.36	1.37	1.61	2.05	1.80

`chain`, neighbor prefetching shows about double of issued prefetches with respect to the other techniques (due to the more complicated memory address scheme, there exists a higher probability that the neighbor blocks are not present in the cache). On average, although the neighbor method attempts a high number of prefetches, it actually issues a number of prefetches comparable with the other techniques.

When evaluating performance, the benefit in the execution time achieved thank to a reduced miss rate must be compared against the potential time overhead carried by prefetching. The most relevant observation is that, whenever the number of actual prefetches is comparable with that of eliminated misses, an overall performance improvement will be obtained, since prefetching time *can be intrinsically better hidden in the execution time* than the fetch on-miss imposed by a miss. This is the case of OBL, OBVL and neighbor prefetching in most MPEG executions (compare Tables 8 and 9 with 11 and 12). Amongst them, the neighbor prefetching is the one that achieves the highest number of eliminated misses.

Another comparison concerns the case in which a specific prefetching technique assesses a lower total number of misses at a comparable number of prefetches: if the prefetching overhead is similar at run-time, the technique will achieve a larger reduction in the execution time. This is the case of the neighbour prefetching in many cases (see again Tables 8, 9, 11 and 12).

Finally, a temporal analysis is currently under development, which preliminary results are presented in [8]. These results report measurements of the actual memory latency carried by both line fills due to misses, and prefetches (called *MADT*, Memory Access Delay Time). Nevertheless, the final impact of the number of cache misses or the MADT on the execution time must take into account also many architectural aspects which cannot be improved or degraded by a cache prefetching strategy, and thus are out of the scope of this paper. These aspects include the instruction set, the compiler, the average CPI, the input data, and many others. For example, a 10% MADT improvement, which was measured in some experiments reported in [8] in the case of MPEG decoding (where the MADT was about 30% of the total execution time) results in an speedup of about 3% in the overall execution time.

8. 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 on a multimedia image processing benchmark, including MPEG-2 decoding and encoding, which are specially relevant applications for multimedia. The reference architecture used is a dedicated cache with parameters typical of a modern L1 cache. In the paper we have shown that prefetching techniques based on 2D locality, and particularly the neighbor prefetching we propose, generally perform better than the other methods (one-block-lookahead and some variations of adaptive schemes) in terms of number of eliminated misses. A preliminary temporal analysis provided in [8] confirms that neighbor prefetching can outperform the other algorithms also in terms of execution time improvement. At the same time, the prefetching overhead is comparable with that of the other techniques: since prefetches can be intrinsically better overlapped with execution than fetches on-miss, this may grant a significant overall performance improvement.

References

1. P. Baglietto, M. Maresca, M. Migliardi, and N. Zingirian, "Image processing on high performance RISC systems," Proceedings of the IEEE, Vol. 84, No. 7, pp. 917–925, 1996.
2. T.F. Chen and J.L. Baer, "Effective hardware-based data prefetching for high-performance processors." IEEE Transactions on Computers, Vol. 44, No. 5, pp. 609–623, 1995.
3. T.F. Chen and J.L. Baer, "A performance study of hardware and software data prefetching schemes," in Proc. of the 21th Intl. Symp. on Computer Architecture (ISCA), pp. 223–232, 1996.
4. L. Chiariglione, "Impact of multimedia standards on multimedia industry," Proceedings of the IEEE, Vol. 86, No. 16, pp. 1222–1227, 1998.
5. Coding of audio-visual objects—Part 2: Visual. ISO/IEC DIS 14496-2 Information technology.
6. R. Cucchiara and M. Piccardi, "Exploiting image processing locality in cache pre-fetching," in Proc. of the 5th Intl. Conf. on High Performance Computing (HiPC), 1998.
7. R. Cucchiara, M. Piccardi, and A. Prati, "Exploiting cache in multimedia," in Proc. of IEEE Intl. Conf. on Multimedia Computing and Systems (ICMCS), Vol. 1, 1999.
8. R. Cucchiara, M. Piccardi, and A. Prati, Temporal Analysis of Cache Prefetching Strategies for Multimedia Applications. Tech. Rep.—Dipartimento di Ingegneria, Università di Ferrara, 2000.
9. K. Diefendoff and P.K. Dubey, "How multimedia workloads will change processor design," IEEE Computer, 1997.
10. R.J. Eickemeyer and S. Vassiliadis, "A load instruction unit for pipelined processor," IBM Journal of Research and Development, Vol. 37, pp. 547–564, 1993.
11. D. Gall, "MPEG: A video compression standard for multimedia application," Communications of the ACM, Vol. 34, No. 4, pp. 46–58, 1991.
12. A. Gonzales, C. Aliagas, and M. Valero, "A data cache with multiple caching strategies tuned to different types of locality," in Proc. of ACM Intl. Conf. on Supercomputing, 1995, pp. 338–347.
13. J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach. Morgan Kaufmann, 1990.
14. ACME Cache Simulator. http://atanasoff.nmsu.edu/~acme/acs.html.
15. G. Irlam, Spa. Personal Communication, 1992.
16. D. Joseph and D. Grunwald, "Prefetching using markov predictors," IEEE Transactions on Computers, Special Issue on Cache Memory and Related Problems, Vol. 48, No. 2, pp. 121–134, 1999.
17. N. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in Proc. of 17th Intl. Symp. on Computer Architecture (ISCA), 1990, pp. 364–373.
18. T. Kanedo and M. Okudaira, "Encoding of arbitrary curves based on the chain code representation," IEEE Transactions on Communications, COM-33, pp. 697–707, 1985.
19. A.K. Katsaggelos, P.K. Lisimachos, F.W. Meier, J. Ostermann, and G.M. Schuster, "Mpeg-4 and rate-distortion-based shape-coding techniques," Proceedings of the IEEE, Vol. 86, No. 6, 1998.
20. I. Kuroda and T. Niscitani, "Multimedia processors," Proceedings of the IEEE, Vol. 86, No. 6, pp. 1203–1221, 1998.
21. C. Lee, M. Potkonjak, and W. Mangione-Smith, "Mediabench: A tool for evaluating multimedia and communications systems," in Proc. of MICRO30, 1997, p. 244.
22. V. Milutinovic, B. Markovic, M. Tomasevic, and M. Tremblay, "The split temporal/spatial cache: Initial performance analysis," in Proc. of the SCIzzL-5, Santa Clara, CA, USA, 1996.
23. A.J. Smith, "Cache memories," ACM Computing Surveys, Vol. 14, No. 3, pp. 473–530, 1982.
24. P. Soderquist and M. Lesser, "Memory traffic and data cache behavior of an mpeg-2 software decoder," preprint for ICCD '97, 1997.
25. J. Tse and A.J. Smith, "Cpu cache prefetching: timing evaluation of hardware implementation," IEEE Computer, Vol. 47, No. 5, pp. 509–526, 1995.
26. Z. Wu and W. Wolf, "Study of cache system in video signal processors," in Proc. IEEE Workshop on Signal Processing Systems (SiPS), 1998.
27. D. Zucker, M.J. Flynn, and R. Lee, "A comparison of hardware prefetching techniques for multimedia benchmark," in Proc. of IEEE Multimedia 96, pp. 236–244, 1996.

Rita Cucchiara

Au: Pls. provide author's biography.

Andrea Prati

Massimo Piccardi