

© 2004 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Neighbor Cache Prefetching for Multimedia Image and Video Processing

Rita Cucchiara*, Massimo Piccardi and Andrea Prati

Abstract

Cache performance is strongly influenced by the type of locality embodied in programs. In particular, multimedia programs handling images and videos are characterized by a bidimensional spatial locality, which is not adequately exploited by standard caches. In this paper we propose novel cache prefetching techniques for image data, called neighbor prefetching, able to improve exploitation of bidimensional spatial locality. A performance comparison is provided against other assessed prefetching techniques on a multimedia workload (with MPEG-2 and MPEG-4 decoding, image processing, and visual object segmentation), including a detailed evaluation of both the miss rate and the memory access time. Results prove that neighbor prefetching achieves a significant reduction in the time due to delayed memory cycles (more than 97% on MPEG-4 with respect to 75% of the second performing technique). This reduction leads to a substantial speedup on the overall memory access time (up to 140% for MPEG-4). Performance has been measured with the PRIMA trace-driven simulator, specifically devised to support cache prefetching.

Keywords

Cache memories, prefetching, multimedia, image processing, neighbor prefetching. EDICS: 2-EXTN

Rita Cucchiara is with Dipartimento di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia, Via Vignolese, 905/b - Modena - Italy - phone: +39-059-2056136 - fax: +39-059-2056129 - e-mail: cucchiara.rita@unimore.it

Massimo Piccardi is with Department of Computer Systems, Faculty of IT, University of Technology, Sydney - Broadway NSW 2007 - Australia - phone: +61-2-9514-7942 - fax: +61-2-9514-1807 - e-mail: massimo@it.uts.edu.au

Andrea Prati is with Dipartimento di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia, Via Vignolese, 905/b - Modena - Italy - phone: +39-059-2056142 - fax: +39-059-2056129 - e-mail: prati.andrea@unimore.it

I. INTRODUCTION

MULTIMEDIA workloads are leading the design of modern computers, since high performance in executing multimedia applications is mandatory. To this aim, memory access performance proves particularly critical, since the speed gap between processors and memories still tends to increase.

Cache memories improve memory access performance by allowing the exploitation of temporal and spatial data locality. However, spatial locality can actually be exploited only if the storage organization in the cache mirrors the memory access schemes embodied in programs. This requirement is often not satisfied for multimedia data, such as images or videos. In fact, image programs are characterized by a peculiar access locality, that we call *2D spatial locality*, since images are structured as bidimensional arrays and whenever the CPU accesses a single data item, a high probability exists to access logically adjacent data items in both vertical and horizontal directions.

Most algorithms for image compression, noise filtering, and image analysis process two-dimensional pixel blocks (e.g., MPEG typically uses 8 x 8 or 16 x 16 pixels blocks), thus exhibiting both horizontal and vertical spatial locality. Since standard cache architectures are based on blocks preserving horizontal spatial locality only, in multimedia programs a large amount of compulsory misses in accessing image data has been reported [1][2][3]. Enlarging the cache size is not the right solution, since it decreases only conflict or capacity misses, and even increasing the block size improves exploitation of horizontal locality only. It is possible to alter also the order of memory references in the algorithms in order to achieve a decreased number of misses. However, we aim to describe a general technique that does not require instruction re-ordering.

For enhancing cache performance, prefetching techniques have been deeply explored. Cache data prefetching consists of inserting data into the cache before they are requested by the current instruction, so as to limit or eliminate the waiting time due to compulsory misses. Prefetching can be implemented mainly in either hardware or software. The main advantage of hardware implementations is that prefetching can exploit program information available only to the hardware, such as the fact that a memory reference results in a hit or a miss. The main drawback, instead, is that hardware implementations are more limited than

software implementations in terms of timing and computational resources. However, the recent increase of on-chip resources has made hardware prefetching more appealing, and therefore in the scope of this paper we will concentrate on hardware techniques [4][5][6]. A detailed analysis of hardware implementations of cache prefetching is found in the paper from Tse and Smith [4]. For hardware techniques, two main directions have been explored: *static* and *adaptive* (or *stride-based*). Static techniques are directed towards prefetching one or more blocks, based on a static assumption of probability; adaptive techniques analyze history of strides (the stride is the difference in address between two consecutive accesses made by a same instruction) to predict the most useful blocks to be prefetched. The former approach achieves high performance on vector data, and this results in a good performance improvement also on image data, since at least the horizontal locality is used. The latter is good for programs working on data with regular memory access schemes. For this reason they have been proposed for multimedia programs, too [2][7][8][9].

In fact, many multimedia image and video processing programs are regular in memory access: think for instance of the basic convolution-based filtering, where all the image points are raster scanned and processed by a coefficient mask. Nevertheless, some multimedia programs are not regular, since the computation is data-dependent. Typical examples are contour-tracing algorithms, region-growing segmenters, or labeling operators, used to extract visual objects from images (visual objects are supported by MPEG-4 and MPEG-7). In these cases, even stride-based techniques are not adequate: as we will show in this paper, their performance can be outperformed by some specific techniques optimized for prefetching image data.

For these reasons, this work aims to analyze performance of prefetching techniques on a significant multimedia benchmark with different access schemes to image and video data. The goal is to prove that a significant speedup can be achieved by using prefetching techniques, and especially by some innovative and image-oriented prefetching methods that we present in this paper. In particular, we propose two new static techniques, called *neighbor* and *8-step neighbor* prefetching that outperform other prefetching techniques both in terms of miss rate and memory access time improvement. In [10], we introduced the neighbor prefetching and compared it with other well-assessed prefetching techniques in terms of eliminated cache misses only. In this paper we extend the performance evaluation to temporal analysis, since the most relevant performance

figure in this context is the achievable reduction in terms of memory access time.

The paper is structured as follows: the next section presents related works on cache prefetching, including the main proposals for image and multimedia workloads. Section 3 describes the prefetching techniques compared in this paper, including the novel neighbor prefetching techniques. Section 4 outlines the multimedia benchmark used for performance evaluation. Section 5 describes the temporal model used for timing evaluation and the simulation environment. Section 6 presents the performance results in terms of cache misses and improvement in memory access time on the benchmark's programs; the impact of line size, cache size and miss penalty is shown, too. In the conclusions, the relevant aspects of our work are briefly sketched and the main results achieved commented.

II. RELATED WORKS

In this section, we aim to present a brief overview of the most relevant works addressing *cache prefetching*. Most papers in the literature address general-purpose workloads, but cache prefetching techniques have been evaluated also for some multimedia programs; for instance, results are proposed by Zucker *et al.* in [7][9] for MPEG-2 decoder and encoder programs and by Pimentel *et al.* in [8] for the median algorithm, used to produce non-interlaced frames from videos with interlaced frames. In this work we extend the analysis to a larger and more complete (in the sense that it covers different image data access schemes) multimedia benchmark oriented to image and video processing.

In cache prefetching, the two main directions of *static* and *adaptive* techniques have been investigated. The basic static prefetching method is known as One-Block-Lookahead (OBL) [11] (also called *always prefetch* in a more recent work [4]). With OBL, 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. Many other OBL-based solutions have been proposed; amongst them we cite those based on stream buffers [7][12][13], that prefetch one or more blocks following block $i + 1$ in another level of the memory hierarchy, i.e. the stream buffer. The implicit assumption of all static techniques is that, given the spatial locality, 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.

These assumptions are correct, and testified by many simulations, especially when arrays are accessed. In this paper we show that they are justifiable for image data too, when pixels of an image or a video frame are evaluated and processed one by one in *raster-scan* mode (i.e. row-by-row). Nevertheless, whenever programs exhibit 2D spatial locality, we will show that prefetching performance can be improved with respect to that of OBL, since it does not exploit locality in the vertical direction.

The other main direction for prefetching is that of adaptive techniques. Adaptive techniques take into account some form of *adaptivity* in order to achieve a better prevision than that represented by block $i + 1$. The idea is to induce block reference probability by recent references history, and is promising for programs with regular memory access schemes. 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. Fu and Patel in [14] proposed the use of an associative memory, called the Stride Prediction Table (SPT), to store stride information. Chen and Baer in [6] proposed the use of a Reference Prediction Table (RPT), similar to the SPT, but with an added state machine to assert if the prediction can be trusted (correct) or not (incorrect). Selecting strides useful for prefetching is often referred to as *stride filtering*. Other proposals combine static and adaptive techniques in order to receive benefits from both the approaches: in [15], a stride predictor is combined with a stream buffer and a voting scheme is used to decide whether the stream buffer or the stride predictor must be used. Finally, other authors propose more complicated forms of adaptivity (as in [16]), but we expect them to be too demanding in hardware resources to be feasible in hardware, and therefore didn't use them for comparison in this paper.

In this work we consider image data only, i.e. the regular array-based structure used for referencing pixels in an image or in a video frame. Hence, we evaluate the performance of a possible data cache for image data only. Several studies refer to the performance improvement that can be achieved by further splitting the data cache for scalar and vector data types [17] or for temporal locality and spatial locality [18][19]. In [9], a stream cache separated from the main cache is used to store data accessed with regular strides. Just as many authors suggest the use of a separate cache for vector data, we suggest the adoption of a special cache for image data since this choice, together with a specific cache replacement strategy, reduces cache misses in

the very frequent accesses to image data [20]. Although the addition of a further dedicated cache in modern general-purpose processors is not very likely, this may not be the case for multimedia processors which could significantly benefit from the results reported in this paper.

The primary metrics typically used in the literature for cache performance evaluation are the *miss rate* and the *efficiency* expressed as the fraction of eliminated misses. These metrics [7][9][10][6] are useful for an initial evaluation of prefetching techniques because they focus on cache performance independently of system parameters. Nevertheless, as clearly stated in [4] and [5], performance analysis must address the execution time. Tse and Smith in [4] proposed a system-level analysis of the impact of prefetching on system performance by way of an accurate cycle-by-cycle execution simulation. The main metric used is the MCPI, defined as the total memory access penalty due to delayed memory cycles divided by the total number of instructions executed. This figure includes all aspects which can be influenced by prefetching, while at the same time excluding all those which cannot. To the purpose of comparison, the metric used in [4] is actually the *relative* MCPI, expressed as the ratio between the MCPI with and without cache prefetching. Hennessy and Patterson in [5] use instead the average memory access time (AMAT), which includes also the time due to non-delayed memory cycles, and is divided by the number of memory references. The AMAT is expressive of the overall impact of memory access. In this work, we, report results with metrics equivalent to the relative MCPI and AMAT for a wide range of system parameters such as the prefetching technique, cache size, miss penalty, and others.

III. PREFETCHING TECHNIQUES

As previously stated, we divide prefetching techniques in the two main categories of static and adaptive prefetching. To the purpose of comparison, in this paper we use OBL as reference for the static techniques [4][6], and the stride-based prediction technique [9] as the reference for the adaptive class (called SPT for short in the following). In [10], we explored also more sophisticated adaptive techniques based on some form of stride filtering (2-delta filters, [21]), but performance achieved didn't prove significantly different from that of the basic scheme, and therefore are not reported in this paper.

In OBL prefetching, every time a memory reference is made, a lookup in the cache is performed for the block that follows the currently referenced one in memory. If we call A_0 the current address and $B(A_0)$ the current *block address*, i.e. the memory address without the byte offset field, the block address of the looked-up block will be $B(A_0) + 1$ (see Table I). If this block is absent from the cache, its prefetch is issued. Thanks to the lookup, the actual prefetching of data is performed only if it is not already in the cache, thus avoiding useless bus occupation.

In adaptive prefetching, every time a memory reference is made, a lookup in the SPT is performed. If there is a hit, a lookup in the cache is performed at the address $B(A_0 + S)$ (by calling S the computed stride). If this block is absent from the cache, its prefetch is issued.

In this paper, we propose to explicitly explore the 2D spatial locality in images by a new approach called *neighbor* prefetching. To this aim, we give definitions of three algorithms, called *basic*, *first-reference*, and *8-step* neighbor prefetching, respectively.

Def. 1: The *basic neighbor prefetching*, at each memory reference, attempts to prefetch all blocks containing data in the nearest-neighborhood of that currently referenced; we assume a neighborhood of 3 x 3 blocks that are called 8-connected blocks; for each block belonging to the 8-connected set that is not in the cache already, a prefetch is issued.

For the sake of clarity, see the sketch of Fig. 1. Let us assume image pixels are stored row-by-row, and divided in blocks of the same size of cache blocks; the 8-connected blocks contain pixels adjacent in both vertical and horizontal directions.

In basic neighbor prefetching, every prefetching stage consists of eight lookups and M issued prefetches, with $M \in [0,8]$ (M is equal to 0 if all the 8-connected blocks are already present in the cache). The block addresses of the looked-up blocks are shown in Table I, with reference to the A_0 address and where NB_{row} is the number of bytes in an image row. Neighbor prefetching mirrors the way data are accessed by many image processing algorithms, including both raster-scan and data-dependent ones, and thus promises a general improvement in 2D spatial locality exploitation. At the same time, potential drawbacks of this approach are evident: i) it carries a substantial increase of the lookup pressure, and ii) potentially issues a higher number

of prefetches. The first effect can be limited by an adequate cache architecture. We assume a highly efficient lookup mechanism, with a multiple-port tag directory and pipelining with prefetch issuing, so as to consider the lookup time negligible. We also introduce two modifications to the basic neighbor prefetching, reducing both the lookup pressure and the number of issued prefetches.

We consider the sequences of memory references made all to a same block. A sequence starts with the first reference to a block and ends with the first reference to a different block (not included). Therefore, a generic sequence can range from one reference only up to an unlimited number of references. However, in the tested applications sequences are typically in the order of a few units. The idea is to reduce the amount of prefetching by scheduling the prefetching activity not on *all* the references but only on the *first* reference of each sequence. The rationale of first-reference neighbor prefetching is that if a block is prefetched at this stage, it will not be substituted in the cache in the sequence access, making it unnecessary to check for its presence during next references to the block in the same sequence.

Def. 2: In *first-reference neighbor prefetching*, the prefetching activities are scheduled at the first reference only of each sequence of references.

First-reference neighbor prefetching minimizes lookup costs, accounting for them just once for each block referenced. Nevertheless, drawback ii) remains on the first reference, when a high M number of prefetches could be issued all together, with a potentially long completion time. In order to improve this aspect we define a modification for distributing the M prefetches over time along the sequence of references to the same memory block.

Def. 3: In the *8-step neighbor prefetching*, at each memory reference of a sequence of references to a same block, at most one prefetch is actually issued.

We can more precisely describe this method with a simple algorithm. Let us call block i one of the 8-connected blocks of the currently referenced one, with $i = 1...8$ as in Fig. 1. At each memory reference to block A_0 the following algorithm is executed:

```

{ if first_reference ( $A_0$  block) = TRUE then  $i \leftarrow 1$ ;
  else  $i \leftarrow \textit{lastdirection} + 1$ ;
   $hit \leftarrow \text{TRUE}$ ;
  while ( $i \leq 8$  AND  $hit = \text{TRUE}$ ) do
    {  $hit \leftarrow \textit{lookup}$  (block  $i$ );
      if ( $hit = \text{TRUE}$ ) then  $i \leftarrow i+1$ ;
      else  $\textit{prefetch}$  ( $i$ ); }
   $\textit{lastdirection} \leftarrow i$ ; }

```

Every time the program changes the referenced block, the direction #1 is looked-up. If the data is already present in the cache the following block within the neighborhood (in the clockwise direction) is checked. This process continues until either the data is not present in the cache ($hit = \text{FALSE}$) or the whole neighborhood has been checked. The next reference to the same block will restart the process from the last direction checked. The idea is to more effectively distribute the prefetching activity over time.

Block #1 is used for the first lookup (like in OBL) since it is that with the highest static probability of access (experimental evidence is given in the next section). The clockwise direction is a heuristic rule tuned in raster scan processing, since data belonging to block #2 should be absent from the cache and thus its prefetching ought to be scheduled before the others. Therefore, for each reference of a sequence of references to a same block, the 8-step neighbor prefetching performs a number N of lookups (with N ranging between 0 and 8) until a miss occurs; the number of lookups in the same block cannot exceed the number of 8, like in first-reference neighbor prefetching. Moreover, the 8-step algorithm performs at most one prefetch for each reference, thus avoiding long prefetching latencies.

In neighbor prefetching, in order to enable the prefetching mechanism, the memory address A_0 must be associated with the correct image row size in bytes, NB_{row} , by way of a mapping function. For implementation of the mapping function, we refer to modern programming languages such as C, C++ and Java and we assume that image data are declared as either static or dynamic 2D arrays (or objects derived from 2D arrays). For static variables, the NB_{row} information can be extracted by the compiler from variable declarations and stored in a special symbol table; the mapping function can then be initialised at run time from the special symbol table. For dynamic variables, the NB_{row} information can be stored in the mapping function at allocation time.

In either cases, the extracted information must then be made available to the prefetching unit. In our implementation, the prefetching unit is enabled to query the mapping function to retrieve NB_{row} from A_0 ; in order to improve the efficiency of this mechanism, the most recent A_0 - NB_{row} associations are assumed to be cached in the prefetching unit.

IV. MULTIMEDIA BENCHMARK

Multimedia programs exhibit a different locality in data access depending on the data type: numerical data and text are accessed with standard spatial/temporal locality schemes, while audio is often processed with a stream access showing a strong 1D spatial locality. Instead, in image and video processing a wide variety of memory access schemes is used, often depending both on programs and data themselves. Unfortunately, no generally accepted benchmark exists for multimedia image and video processing yet. Therefore, we selected a kernel of basic programs that are commonly adopted in image multimedia tools, as in other works on multimedia performance evaluation [7][22]: the set includes image and video decompression programs and common programs for image manipulation, characterized by different data access schemes.

A. The benchmark's programs

Table II summarizes the benchmark's programs used in this work.

Image convolution (**Convo** for short) is the basic algorithm for image processing; it consists of processing each image pixel by convoluting pixels of its bidimensional neighborhood with a coefficient mask. 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 data access are not data-dependent; a substantial amount of strictly 2D locality is embedded in processing each pixel and its neighborhood. In many papers addressing performance evaluation in image processing [2][7][23], convolution is included in the basic benchmark.

Thresholding (**Thresh**) is the pure raster-scan data access, since pixels are loaded and processed one by one, row by row. In particular, it consists of a simple comparison between the pixel value and an assigned threshold.

Chain code (**Chain**) is included as a typical data-dependent image processing program characterized by an *unpredictable* 2D spatial locality. It is a propagative algorithm, since the computational flow is propagated along the image, in an a-priori unknown direction depending on the data themselves [24]. The program we used is a standard edge tracing algorithm that scans objects' contours and can be used for encoding objects' geometric properties [24][25][26], including shape encoding for MPEG-4 [27][28]. In edge tracing, the direction of the next pixel to be used depends on the image and its edges, and therefore the address of the next memory reference is data-dependent.

The benchmark includes also standard video decoders. *MPEG-2 decode* is the typical benchmark for multimedia processors, since it is currently the most spread multimedia program, both for video streams and files [1][6][27][28][29][30]. MPEG frames are of three different types: I (only spatial compression in JPEG style), P (forward-predicted), and B (forward-backward predicted). Typically MPEG sequences are periodical repetitions of a same type pattern, like for instance the standard IPBBPBBPBBPBB (MPEG videos used in the tests); when the pattern is IIIIII, the decompression (called JPEG in Table II) consists only of JPEG decoding of a sequence of frames (sometime called also *MJPEG* or *M-JPEG* type). MPEG-2 performs a variable-length decoding and inverse quantization of Discrete Cosine Transform (DCT) coefficients, Inverse DCT (IDCT) computation, and motion compensation, mostly operating on 8x8 and 16x16 pixel blocks. For the tests we have used different MPEG data, selected from standard movies with different type pattern, image size and number of frames (files and address traces are available at <http://enki.ing.unimo.it/ImageLab/research.html>). In particular, the first two (Frisco and Pirates) are MJPEG and are described in Table III , while the others (Waterski and FlowerG) are MPEG movies with sequence pattern IPBBPBBPBBPBB and are described in Table III.

Finally MPEG-4 decode (**MPEG4**) has been included as an example of the recent audio and video compression standard, ISO/IEC 14496 [27]. MPEG-4 standard defines the basic coding structure for managing media objects, and among them Visual Objects (VOs) as meaningful individual parts of images that are coded separately. VO decoding requires more image planes to be decoded for each VO, one for the shape (or *alpha plane*) and one for texture information. In this work we use the MPEG-4 decoder from CSELT,

developed according to standard specifications [27]. Tests are reported for a video example, whose features are summarized in Table III; note that the trace from only 4 frames results as being so large as to contain more than 23.4 millions memory references and, for this reason, we limit our study to few frames. Nevertheless, the behaviour of the program and the data access pattern do not change significantly when considering more frames.

The source code of the programs selected are the following:

- as JPEG/MPEG2 decoder, the free codec developed by the MPEG Software Simulation Group has been used. The complete reference is: MPEG-2 Encoder / Decoder, Version 1.2, July 19, 1996, Web site <http://www.mpeg.org/MSSG/>;
- as MPEG4 decoder, the Version 1.0 (August 1999) of the program developed for the MPEG-4 Video (ISO/IEC 14496-2) standard within the framework of the European ACTS project MoMuSys was selected. Further information on the project can be found at <http://www.tnt.uni-hannover.de/project/eu/momusys/> and the software can be downloaded for example from <http://www.ganesh.org/~pmeerw/>.

Instead, the **Thresh**, **Convo** and **Chain** programs are self-written in standard C language.

B. *The impact of the workload on cache architecture*

In this work, we analyze the impact of cache architecture for accessing image and video data type (that we call *2D data*) and we propose new cache prefetching techniques for a dedicated 2D cache, containing only 2D data. Instead, no improvement is needed for the conventional cache containing other data types (that we call for short, although improperly, *scalar data*) such as text, audio, numerical data, and so on. This is due to the fact that the impact of memory accesses to image data is significantly more critical than that to other data in multimedia image processing applications. Table IV reports a comparison of the number of memory accesses to 2D and scalar data ($N_{REF} 2D$ and $N_{REF} Scalar$ columns respectively) and the number of misses and miss rate without any prefetching approach for a 2D and conventional caches of 32 KB each, 2-way set-associative, with 32 bytes/line. Table IV refers to the following algorithms: **Convo** and **Thresh** on a 512x512 image, **Chain** on a 352x240 frame of the FlowerG Video reported in Table III, JPEG and MPEG2 programs for videos of Table

III and MPEG4 decode.

Table IV shows that the number of scalar references is normally higher than that of 2D references (apart from **Convo**) but, conversely, the scalar miss rate is nearly zero in all cases. This means that the standard locality is already perfectly caught by a conventional cache and that no further effort is justified to reduce it any more (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. This proves that the locality of 2D references is not caught by conventional caches as accurately as other data. Therefore, it could be convenient to explore different prefetching techniques based on a separate cache, in order to achieve better performance while granting less interference with data exhibiting different locality.

Raster-scan algorithms, such as **Convo** and **Thresh**, allow for static prediction of cache misses, that are mainly of the compulsory type (if the convolution mask radius is limited). Thus, cache performance in terms of miss rate could be strongly improved with data prefetching techniques. Moreover, convolution programs suitably match standard cache architecture in terms of temporal locality, which arises from the large re-use of pixels in a neighborhood (in fact, Table IV shows that **Convo** has the lowest 2D miss rate).

Conversely, **Chain** has been included as an example of image computation that potentially does not benefit from standard cache techniques since it exhibits unusual and non-predictable spatial locality. From a data access point of view, propagative algorithms show the same bidimensional spatial locality of the raster-scan ones and at the same time the impossibility of predicting the data access. Moreover, the temporal locality is difficult to emphasize, because points involved in the neighbor computation may be used in the future, but perhaps only after long computation and thus capacity misses probably occur. For instance, Fig. 2 shows that when the P pixel is referenced, the other pixels loaded in cache (of the same blocks) could be unused or, as in this case, used after a long processing time; therefore spatial locality is not exploited and the data loaded in cache do not have adequate timeliness.

Nevertheless, **Chain** has an embedded 2D data locality that can be measured: we profile memory accesses to image data structures based on a simple measure of spatial locality given by the difference between the current and the following memory address made by a same instruction (i.e. the stride). In the histogram of

Fig. 3(a), a large amount of strides equal to +1 and -1 testifies the standard 1D spatial locality (i.e. the pixel accessed is adjacent to that previously accessed in the forward or backward direction, respectively), but a large number of strides equal to -352 and 352 indicates that the block of pixels belonging to the previous and following rows (image is 352x240) is accessed with a 2D spatial locality.

The JPEG and MPEG2 programs are dominated by operations on image blocks (8 x 8 pixels) or macroblocks (16 x 16 pixels), thus exhibiting a strong 2D data locality. MPEG2 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 2D miss rate depends on the image format and compression but is the same order of magnitude for the same MPEG format: in our case is about 2.8% for JPEG videos and 1.4% for MPEG2 (see Table IV). Contrary to what expected, MPEG2 has a lower miss rate than JPEG. This has been confirmed by our former experiments in which the locality of the MPEG2 decoder has been studied: the results of these experiments are not reported in this paper, but they demonstrate that MPEG2 shows higher re-use of the pixel's neighborhood and thus achieve lower miss rates.

The performance improvement achievable by reducing the 2D miss ratio depends on the amount of the 2D references with respect to the overall amount of memory references; in our workload, for the MPEG applications, for instance, the percentage of 2D references vary between about 3% and 30%.

Fig. 3(b) reports the stride histogram for the Waterski video. The dominant stride value is 1, which means that the greater part of memory accesses are performed in raster-scan order. However, some other non negligible stride values are concentrated around values corresponding to pixels of a same 8x8 (16x16) block belonging to the previous and following rows, thus providing the 2D data locality.

Finally, the MPEG4 program is more complex since many image planes for each VO are processed with a large number of dynamic image data allocations. It computes the same Inverse DCT as the MPEG-2 programs but it must also decode alpha planes for the VOs' shape information. MPEG4 decoding exhibits a very high 2D miss rate that is of 5% in our test (see Table IV). By assuming that most of the misses are compulsory, all the benchmark programs should benefit greatly from adopting aggressive prefetching techniques.

Miss rates of multimedia applications cannot be considered negligible in general: in our workload the 2D miss rates vary in a range of 1-5%, but strongly depending on the cache size and configuration; miss rates of up to 20% have been reported by other authors for some cache configurations [3].

V. ARCHITECTURAL AND TIMING MODEL

In this section, we describe the architectural and timing model used to evaluate results by means of temporal analysis. We simulate a RISC processor with an ideal single pipeline allowing a standard one clock-per-instruction execution ($T_{EXEC} = 1 \cdot T_{CK}$) and a reference memory architecture with a specific cache for image data (2D cache) as well as a standard data cache for other data (scalar cache). Simulations are based on the following assumptions [31]:

- The 2D and scalar caches can be accessed separately, without interference.
- The size of the 2D cache is assumed 32 KB (1 K lines of 32 bytes each, 2-way set-associative), that is a typical size for an L1 split cache. In addition, we report performance also for different cache sizes (both smaller and larger).
- The 2D cache is not multiple-ported, i.e. only either one miss or one prefetch can be sustained at a time. The hit penalty is assumed null for simplicity, i.e. $T_{HIT} = 1 \cdot T_{CK}$, included in the instruction execution, while miss and prefetch accesses have the same penalty $T_{MISS} = T_{PF} = 8 \cdot T_{CK}$ for the line fill. In addition, we report performance also assuming higher miss penalties.
- The lookup time, i.e., the time to access the cache tag directory, is assumed null; this is a simplifying assumption which does not affect the significance of simulation results. In particular, for neighbor prefetching we assume a multiple-ported tag directory which can sustain multiple lookups at a time.
- *Misses are blocking (or synchronizing)* events for execution, meaning that when a miss happens, execution must wait the miss line fill completion before starting again. Moreover, any prefetching activity scheduled is completed before serving the miss. Hence, if the prefetching completion loads in the cache the data required by the miss, no miss line fill is then issued. This mechanism is called *lookahead miss inquire*, i.e. when a miss occurs, the current prefetching queue is tested and a line fill is performed only if queued prefetches are not

already loading the data which caused the miss, thanks to the lookahead miss inquire.

- Instead, *prefetches* are obviously *non-blocking* events for execution, i.e. can be accomplished in parallel with instruction execution and their penalty could be hidden by execution of other instructions.
- In our architectural model, we assume in-order execution, since this is the typical execution model of several processors used for multimedia applications such as for instance the Philips Trimedia and the Texas C6000. Modern general-purpose processors are based on out-of-order engines which can more effectively tolerate the effect of cache misses.

The total execution time is computed according to this model. We account a time T_{EXEC} (the basic one-clock execution time) for completing each instruction with no memory reference or with a memory hit access. If an instruction causes a miss, we account an additional execution time of T_{MISS} . If an instruction causes a prefetch, prefetching is performed in parallel with instruction execution, and no time is accounted for prefetching in addition to that of instruction execution. In case instruction execution causes a miss while a prefetch is performed, the prefetching activity is completed before serving the miss; for completion of prefetching we account an additional execution time $(1 - OV)T_{PF}$, where OV is the fraction of T_{PF} covered by overlapping. If further prefetches were already scheduled in the prefetching queue, they are completed and a time $T_{PF} = T_{MISS}$ is added for each of them to the total execution time. The time T_{MISS} for the miss line fill is counted only if the prefetches in the prefetching queue have not already loaded in the cache the data required by the reference which caused the miss.

In this work, performance were measured by using trace-driven simulation.

Traces are collected by using the tracer Spy [32] and processed using our simulator, called PRIMA (PRefetching IMproves Multimedia Applications), evolved from ACME [33] simulator: we integrated a support for dealing with prefetching and 2D caches; then, we extended it to perform temporal simulation ([34]).

VI. PERFORMANCE RESULTS

Using the architectural model and the benchmark defined, we measured the performance achieved using caches without prefetching, comparing well-known prefetching methods (OBL and SPT) with the new

neighbor-based ones.

A. Number of misses

The number of misses is an important parameter for estimating the efficacy of a prefetching method, since an effective prefetching method will assess a high miss reduction. In addition to the miss number, an expressive measure to the aim of comparison between the different prefetching techniques is the fraction of misses eliminated. We call it *miss-based efficiency* of the i -th prefetching method, η_i , defined as:

$$\eta_i = \frac{N_{MISS} - N_{MISS_PF_i}}{N_{MISS}} \quad (1)$$

where N_{MISS} is the number of misses without prefetching. This miss-based efficiency ranges between 0 and 1 (but could be even negative in case of ineffective prefetching) and tends to 1 when prefetching achieves the highest performance, that is its $N_{MISS_PF_i}$ (i.e., the number of non-eliminated misses) tends to 0.

Table V and Fig. 4 show, respectively, the number of misses $N_{MISS_PF_i}$ and the efficiency η_i (in percentage) for the four considered prefetching techniques on the test programs, for a 2-way set-associative cache of 32 KB with 32 bytes per line.

Results confirm that exploiting horizontal spatial locality only (as OBL does) is not an effective solution: in fact the number of misses with OBL prefetching remains non negligible and the OBL's efficiency is only 75% for JPEG, 83-87% for MPEG2 and is 61.9% only for MPEG4 (see Fig. 4). Better results are achieved by SPT prefetching (η is greater than 90% for MJPEG video, is 86-87% for MPEG2 videos and in particular is 97.5% for MPEG4). However, the best performance is obtained by both the neighbor-based techniques, with efficiency very close to 100% for most of the programs. If the prefetching mechanism did not introduce any time cost, these results would support the feasibility of including prefetching techniques, especially neighbor-based, in the design of a processor oriented to multimedia image and video processing.

B. Time analysis

However, the miss number is not sufficient to define execution performance, since many bus transfers are due to prefetches, and if the prefetch number is too high or if prefetch is not enough overlapped with instruction

execution, according with the model previously presented, the gain achieved by the miss reduction would be lost due to the prefetching cost [4].

As a matter of fact, the prefetching techniques issue many prefetching operations, whose number is reported in Table VI; note however that this number is less than or equal to the original number of misses. For each method the second row of Table VI denotes the number of prefetch cycles that are not completely hidden by instruction executions. In many cases this number is 0: this mean that all prefetches issued are hidden and do not cause any time penalty. Conversely, in other cases (especially for MPEG2 and MPEG4) the number of not completed prefetches introduces a non negligible delay, that depends on the fraction of prefetch cycle that is not completely overlapped ($1 - OV$).

In order to take into account the prefetch cost, we compute the *Memory Access Delay Time MADT*, as adopted in [4][35]. This figure accounts for the delay caused by the access to the lower level of memory hierarchy. Thus, we define MADT as the ratio between the total amount of memory delay and the number of memory reference instructions executed (N_{REF}).

In absence of prefetching, this quantity is:

$$MADT_{NO_PF} = \frac{N_{MISS}T_{MISS}}{N_{REF}} \quad (2)$$

In the case of the i -th prefetching (by assuming $T_{PF_i} = T_{MISS}$), MADT becomes:

$$MADT_{PF_i} = \frac{N_{MISS_PF_i}T_{MISS} + N_{PF_i}(1 - OV_{PF_i})T_{MISS}}{N_{REF}} \quad (3)$$

In Eq. 3, $N_{MISS_PF_i}$ accounts for the actual number of misses (i.e. excluding those attempted but not issued thanks to the lookahead miss inquire mechanism), as shown in Table V. N_{PF_i} is the total number of issued prefetches (see Table VI), while $N_{PF_i}OV_{PF_i}T_{MISS}$ is the total amount of time saved thank to the overlap between prefetches and instruction execution (OV_{PF_i} being the average of all OV contributions over the whole execution). Therefore, MADT is a more significant performance measure than only the number of misses, since it takes into account also the number of not completed prefetches, weighted by the fraction of not overlapping with other executed instructions.

The N_{PF_i} quantity strongly depends on the prefetching technique. As summarized in Table I, the OBL and SPT techniques schedule a maximum of one prefetch for each reference: therefore the prefetches issued are less than (or at most equal to) the number of references N_{REF} (note that we are considering prefetching *on reference*). Actually, by comparing Tables IV and VI, it is evident that the number of issued prefetches is just a few percents of the number of references. Instead, in first-reference neighbor prefetching, as discussed in Section 3, a maximum of eight prefetches could be issued on the first reference of each sequence of references to the same block; in the 8-step version, eight prefetches (at most) are distributed along the sequence of references to the same block. Since the length of a sequence of references is not known a priori, the neighbor-based techniques could cause a higher number of prefetches and potentially a higher MADT. Instead, by comparing Table IV and Table VI, we measured that even for neighbor techniques the number of prefetches actually issued is much less than the number of references. This proves that the locality embedded in the program is suitably exploited by the cache prefetching techniques.

Table VII reports the total memory access delay time ($MADT * N_{REF}$) for each tested program, or in other words the number of clocks spent for accessing the lower level of memory hierarchy; without prefetching (first row), it measures the miss penalty only and in the case of a prefetch technique, the sum of both miss and prefetch penalties.

Just as in the previous section we defined the miss-based efficiency (Eq. 1), we can define the *MADT efficiency* in terms of MADT as:

$$\eta_i^T = \frac{MADT_{NO_PF} - MADT_{PF_i}}{MADT_{NO_PF}} \quad (4)$$

This measure has the characteristics of an efficiency, but at the same time is a relative measure equivalent to the relative MCPI of [4] ($\eta_i^T = 1 - relative\ MCPI$). The measures of η_i^T , reported in the histogram of Fig. 5, are similar to the η_i ones of Fig. 4, but differ for the cost of the non completely-overlapped prefetching. In fact, by substituting the definition of Eq. 2 and 3 in Eq. 4, we have that

$$\eta_i^T = \eta_i - \frac{N_{PF_i}(1 - OV_{PF_i})}{N_{MISS}} \quad (5)$$

The second term is the cost of prefetching for the non completely overlapped memory accesses. By comparing

the graphs of Figg. 4 and 5 we can note that prefetches are totally hidden (and thus $\eta_i^T = \eta_i$), for **Convo**, while in other cases (**JPEG**, **MPEG2**, **MPEG4**) this cost is not negligible. The prefetching costs particularly affect the OBL and SPT methods and far less the neighbor methods. On average, the 8-step neighbor prefetching exhibits the best performance: the explicit exploitation of the 2D data locality allows a better prediction of the data to be prefetched, together with a good distribution over time of the prefetching activity.

An important consideration is that, observing the time measures for **MPEG4**, not only OBL but also SPT prefetching exhibits a relatively low efficiency (η_i^T is 61.8% and 75.6% respectively). Instead the adoption of neighbor prefetching reveals itself to be extremely beneficial for **MPEG-4**, proving an efficiency of 97%.

Prefetching actually improves performance in terms of memory access time. This result is not obvious, since prefetching could even worsen the memory access time, in the case where the prefetching activity cannot be completely accomplished in overlap with execution and the number of prefetches is high. The *amount of execution time actually saved* can be easily deduced from Table VII, if the number of spent clock cycles with a given prefetching method is compared with that of the first row, assessed without prefetching.

Another relevant measure takes into account how much the improvement obtained by prefetching influences the memory access time: this can be obtained by adding up the MADT (which is the average memory access delay time) with the normal memory access time without delay (equal to $T_{HIT} = T_{EXEC} = 1 \cdot T_{CK}$). The result is the *average Memory Access Time*, **MAT** [5], defined as:

$$MAT_{NO_PF} = MADT_{NO_PF} + \frac{N_{HIT}T_{HIT}}{N_{REF}} \quad (6)$$

and

$$MAT_{PF_i} = MADT_{PF_i} + \frac{N_{HIT_PF_i}T_{HIT}}{N_{REF}} \quad (7)$$

We use the MAT measure to compute *the relative percentage speedup of the i -th technique* with respect to the case without prefetching as

$$\xi_i^T = \left(\frac{MAT_{NO_PF}}{MAT_{PF_i}} - 1 \right) * 100 = \frac{MAT_{NO_PF} - MAT_{PF_i}}{MAT_{PF_i}} * 100 \quad (8)$$

The measures in the graph of Fig. 6 show that prefetching achieves a significant speedup even when considering the total memory access time and not the memory delay time only. We can note that the speedup

is low with all prefetching techniques for the **Convo** program since its initial miss rate was very low (0.08%). The speedup is instead about 10% with whichever prefetching technique for a simple raster-scan program as **Thresh**. The speedup becomes relevant for **JPEG**, whose initial miss rate is about 2.8%, since neighbor prefetching reaches the 19.7% of speedup (versus the 14.1% and 18.8% of **OBL** and **SPT**). Speedup assesses up to 8% in the best case in our experiments on **MPEG2** with 8-step neighbor. More importantly, the 8-step neighbor speedup achieves up to 35% for **MPEG4**, which has a considerable 5% of initial miss rate. Moreover, this result has been computed taking into account a relatively low $T_{MISS} = T_{PF} = 8 \cdot T_{CK}$; in subsection D we will show that this speedup is significantly greater for higher values of the miss penalty, exceeding values of 140%.

In conclusion, in the case of algorithms with significant miss rates, such as **JPEG** and, in particular, **MPEG4**, the 2D memory access time improvement is highly relevant. In the case of algorithms with still non negligible miss rates, such as **Thresh**, **Chain** and **MPEG2**, the time improvement is about 10% with the given cache configuration. Finally, in the case of a simple and regular algorithm with high temporal locality such as **Convo**, the time improvement achievable by prefetching is heavily limited by the low initial miss rate.

C. Impact of cache size

In this section we discuss the impact of cache parameters, mainly cache size, block size, and degree of associativity, on performance of the multimedia programs.

Varying the block size does not particularly affect performance. This is due to the fact that enlarging block size means improving 1D spatial locality and therefore we should find similar performance improvements as in **OBL** techniques. An example, confirming this assumption, is in Fig. 7(a), showing the MADT efficiency on **MPEG4** decoder, measured on a cache of 32KB two-way associative (with the time model discussed in section 5) with block size varying from 8 to 32 bytes.

We do not report similar results for the associativity degree, since we proved that this not a critical parameter, since most of the misses were compulsory and not due to conflicts.

From our experiments, the cache size is the most relevant parameter. Fig. 7(b) shows the MADT efficiency

measured with different cache sizes (two-way associative cache with 32 bytes per block) for the MPEG4 program. Enlarging the cache size, the performance improves from very small cache (8KB) to medium size cache (32K); then, a further cache enlargement does not improve performance anymore. However, a significant result is that 8-step neighbor prefetching is able to completely eliminate misses even with caches of small size. This measure can be very useful in cache design for low-cost, multimedia processors, which cannot be endowed with large data caches (for instance, like the Philips TriMedia TM1100 processor, with 16 KB data cache).

D. Impact of the miss penalty

The following test analyzes the impact of the miss penalty time on the total memory access time: the higher the penalty, the larger the impact of misses and non completely overlapped prefetches. Therefore, the benefit achievable with a prefetching strategy able to nullify almost all misses while at the same time overlapping prefetches will accordingly be higher. Since the ratio between memory and cache access times is increasing with current technologies, large miss penalties are expected to become common.

In all the previous tests the miss penalty considered was $T_{MISS} = T_{PF} = 8 \cdot T_{CK}$. Now, let us suppose we modify the architecture, by incrementing the number of clocks needed to access to the lower level of hierarchy. What could happen if the memory miss penalty increases? Considering the memory access delay time, not only the miss costs but also prefetching costs will increase. Moreover, this increase is not proportional, since the number of non completely overlapped prefetches changes, together with the amount of fraction overlapped. We report the simulations provided for the MPEG4 decoder with different miss penalties. Results with other programs of the benchmark are similar and are omitted for brevity. Fig. 8(a) shows the MADT efficiency η_i^T for MPEG4 on a 32KB x 32B cache with $T_{MISS} = T_{PF} = 8, 16, 24$ and $32 \cdot T_{CK}$.

With the assumption previously introduced of $T_{MISS} = 8 \cdot T_{CK}$, the efficiency of neighbor methods was higher than that of all standard techniques (see Fig. 5). Increasing the time latency, the prefetching efficiency varies differently for each method: in particular the SPT technique is very sensitive to this change, while OBL is almost insensitive: when $T_{MISS} = 24 \cdot T_{CK}$ SPT exhibits an efficiency of 59.7% only with respect to 62.14% of OBL. Nevertheless, efficiency of neighbor techniques remains close to 100%; in particular, the

8-step neighbor prefetching has an efficiency of 97.42% with $8 \cdot T_{CK}$ and 97.3% with $24 \cdot T_{CK}$.

Results for the speedup on the overall memory access time (MAT) are even more significant. Fig. 8(b) reports the MAT speedup ξ_i^T for MPEG4 for the same 32KB x 32B cache with $T_{MISS} = T_{PF} = 8, 16, 24$ and $32 \cdot T_{CK}$. The speedup achievable with OBL and SPT prefetching grows modestly with the miss penalty, proving that the non eliminated misses and non overlapped prefetches heavily limit prefetching performance. Instead, the speedup obtained with neighbor-based prefetching techniques grows linearly with the miss penalty, exceeding 110% for a miss penalty of $24 \cdot T_{CK}$ and 140% for a miss penalty of $32 \cdot T_{CK}$. This means that the memory access time can be more than halved by neighbor prefetching for these configurations.

VII. CONCLUSION

This paper has presented a detailed performance analysis of different cache prefetching techniques on a multimedia benchmark. The benchmark consists of a set of standard image and video decompression programs (JPEG, MPEG-2, MPEG-4) and some common image processing programs characterized by different memory access schemes to 2D data, including both raster-scan and data-dependent accesses.

We proposed two new techniques of cache prefetching, namely first-reference and 8-step neighbor prefetching, whose results outperform standard techniques both in terms of miss rate and memory access delay time. The most important results can be summarized as:

- 1) Prefetching should be adopted by multimedia oriented processors since all programs working on images and video have a high initial miss rate, and misses are essentially of compulsory type. All prefetching methods improve performance considerably.
- 2) The standard One-Block Lookahead prefetching technique that exploits classic (1D) spatial locality is interesting for raster-scan programs but is not particularly efficient for programs working on macroblocks such as MPEG-2 and MPEG-4.
- 3) Stride-based techniques (such as the Stride Prediction Table) achieve better results in terms of eliminated misses, but if a time analysis is performed by taking into account also the cost of prefetching, it shows limited efficiency (81-83% for MPEG-2 and 75% for MPEG-4 with the reference cache architecture). Moreover, the

SPT technique decreases its performance as the latency time increases (see previous section).

4) The new prefetching techniques proposed in this paper, namely the *neighbor* and the *8-step neighbor* prefetching, show high efficiency both in terms of miss rate and memory access delay time, and an impressive speedup of the overall memory access time. They outperform the other techniques in the test provided on the multimedia workload. On average, the 8-step neighbor performs slightly better than first-reference neighbor, thank to a better distribution of the prefetching activity.

5) A specific consideration must be addressed regarding the MPEG-4 program. MPEG-4 decoding exhibits a significant miss rate (about 5% in the reference 2D cache) since it is characterized by a data access sequence which is not well caught by standard caches. With this workload, neighbor techniques are very interesting: for instance, with a two-way set-associative 32 KB cache with 32 bytes/line, the miss rate becomes about 0.11-0.14% and the speedup in the average memory access time ranges from 35% to 140% assuming increasing miss penalty.

6) A detailed analysis of implementation costs was out of the scope of the present paper. However, the area and power costs of a split data cache can be considered in first approximation proportional to its size. In this paper, results were reported with a reference cache size of 32 KB which is fairly large and correspondingly costly, but results seem to be very good also for smaller sizes such as for instance 8 KB, as shown in Fig. 7(b). Therefore, the technique could be used even for small caches which imply far more limited implementation costs.

These results qualify neighbor prefetching as an effective solution for prefetching of 2D data such as images and videos processed in the most common multimedia applications.

REFERENCES

- [1] I. Kuroda and T. Niscitani, "Multimedia processors," *Proceedings of the IEEE*, vol. 86, no. 6, pp. 1203–1221, 1998.
- [2] R. Cucchiara, M. Piccardi, and A. Prati, "Exploiting cache in multimedia," in *Proc. of IEEE Intl. Conf. on Multimedia Computing and Systems (ICMCS)*, 1999, vol. 1, pp. 345–350.
- [3] Z. Wu and W. Wolf, "Study of cache system in video signal processors," in *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 1998, pp. 23–32.
- [4] J. Tse and A.J. Smith, "CPU cache prefetching: timing evaluation of hardware implementation," *IEEE Transactions on Computers*, vol. 47, no. 5, pp. 509–526, 1998.
- [5] J. Hennessy and D. Patterson, *Computer architecture: a quantitative approach*, Morgan Kaufmann Publisher, 2 edition, 1996.
- [6] 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)*, 1996, pp. 223–232.

- [7] D. Zucker, M.J. Flynn, and R. Lee, "A comparison of hardware prefetching techniques for multimedia benchmark," in *Proc. of IEEE Multimedia*, 1996, pp. 236–244.
- [8] A.D. Pimentel, L.O. Hertberger, P. Struik, and P. Van Der Wolf, "Hardware versus hybrid data prefetching in multimedia processors," in *Proc. of IEEE Intl. Performance, Computing and Communications Conf. (IPCCC)*, 1999, pp. 525–531.
- [9] D.B. Zucker, R.B. Lee, and M.J. Flynn, "Hardware and software cache prefetching techniques for MPEG benchmarks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 5, pp. 782–796, Aug. 1995.
- [10] R. Cucchiara, M. Piccardi, and A. Prati, "Improving cache performance for multimedia applications," *Tech. Rep. n. 91, Dept. of Engineering, University of Ferrara, Italy, accepted for publication on Multimedia Tools and Applications, Kluwer Academic Publishers*, 2001.
- [11] A.J. Smith, "Cache memories," *ACM Computing Surveys*, vol. 14, no. 3, pp. 473–530, 1982.
- [12] N.P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Proc. of IEEE/ACM Intl. Symp. on Computer Architecture (ISCA)*, May 1990, pp. 364–373.
- [13] P. Ranganathan, S. Adve, and N.P. Jouppi, "Performance of image and video processing with general-purpose processors and media ISA extensions," in *Proc. of IEEE/ACM Intl. Symp. on Computer Architecture (ISCA)*, 1999, pp. 124–135.
- [14] J.W.C. Fu and J.H. Patel, "Data prefetching in multiprocessor vector cache memories," in *Proc. of IEEE/ACM Intl. Symp. on Computer Architecture (ISCA)*, May 1991, pp. 54–63.
- [15] G. Singh Manku, M.R. Prasad, and D.A. Patterson, "A new voting based hardware data prefetch scheme," in *Proc. of IEEE Intl. Conf. on High-Performance Computing*, 1997, pp. 100–105.
- [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, Feb. 1999.
- [17] M. Tomasko, S. Hadjiyiannis, and W.A. Najjar, "Experimental evaluation of array caches," in *IEEE TCCA Newsletter*, Mar. 1997, pp. 11–16.
- [18] A. Gonzalez, 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.
- [19] 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.
- [20] R. Cucchiara and M. Piccardi, "Exploiting image processing locality in cache pre-fetching," in *Proc. of IEEE Intl. Conf. on High-Performance Computing*, Dec. 1998, pp. 466–472.
- [21] R.J. Eickemeyer and S. Vassiliadis, "A load instruction unit for pipelined processor," *IBM Journal of Research and Development*, vol. 37, pp. 547–564, July 1993.
- [22] C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench: a tool for evaluating multimedia and communications systems," in *Proc. of IEEE/ACM Intl. Symp. on Microarchitecture (MICRO)*, 1997, pp. 330–335.
- [23] 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.
- [24] H. Freeman, "On the encoding of arbitrary geometric configurations," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 260–268, 1961.
- [25] J. Koplowitz, "On the performance of chain codes for quantization of line drawings," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 3, pp. 180–185, 1981.
- [26] T. Kanedo and M. Okudaira, "Encoding of arbitrary curves based on the chain code representation," *IEEE Transactions on Communications*, vol. COM-33, pp. 697–707, 1985.
- [27] ISO/IEC DIS 14496-2 Information technology, *Coding of audio-visual objects – Part 2: Visual*.
- [28] 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, pp. 1126–1154, June 1998.
- [29] D. Gall, "MPEG: a video compression standard for multimedia application," *Communications of the ACM*, vol. 34, no. 4, pp. 46–58, 1991.
- [30] 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.
- [31] R. Cucchiara, M. Piccardi, and A. Prati, "Temporal analysis of cache prefetching strategies for multimedia applications," in *Proc. of IEEE Intl. Performance, Computing and Communications Conf. (IPCCC)*, accepted for publication, in press, 2001.
- [32] G. Irlam, *SPA – SPARC analyzer tool set*, <http://www.base.com/gordoni/spa/spa-1.0.tar.Z>, 1991.
- [33] <http://atanasoff.nmsu.edu/~acme/acs.html>. ACME Cache Simulator.
- [34] <http://enki.ing.unimo.it/ImageLab/prima.html> PRIMA Web Site.
- [35] G. Park, O. Kwon, T. Han, A. Kim, and S. Yang, "An improved lookahead instruction prefetching," in *Proc. of the High-Performance Computing on the Information Superhighway (HPC-Asia)*, 1997, pp. 712–715.

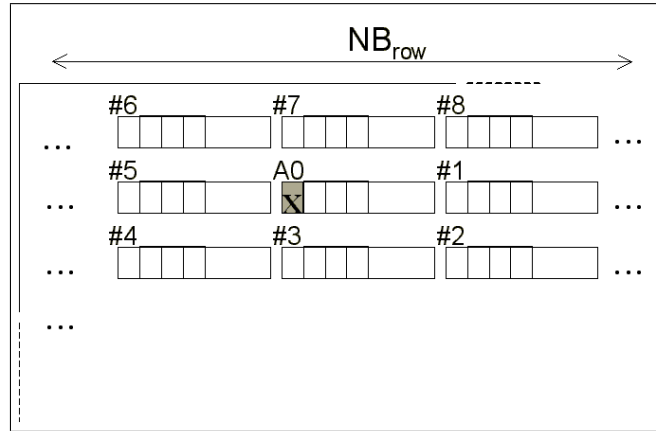


Fig. 1. 8-connected blocks of an image pixel

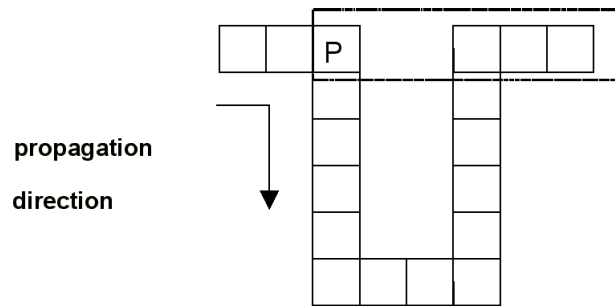
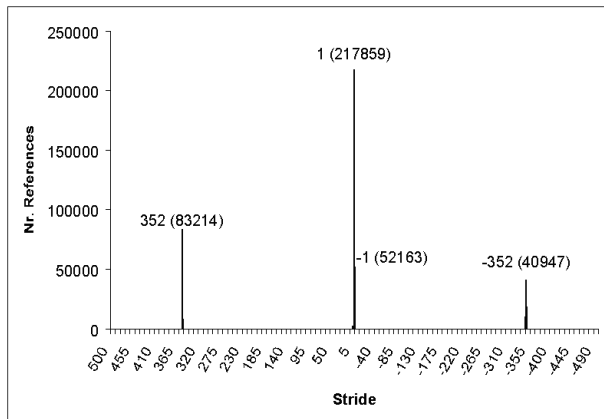
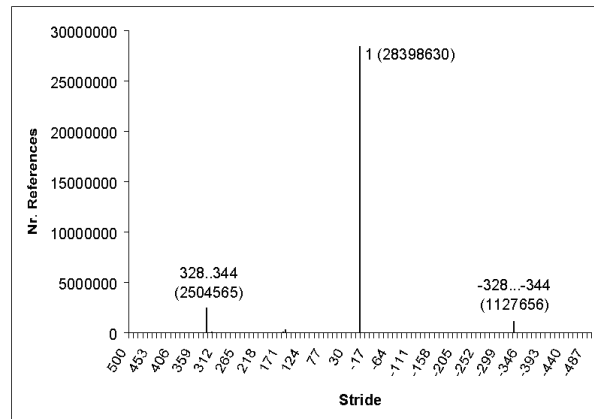


Fig. 2. A propagative algorithm

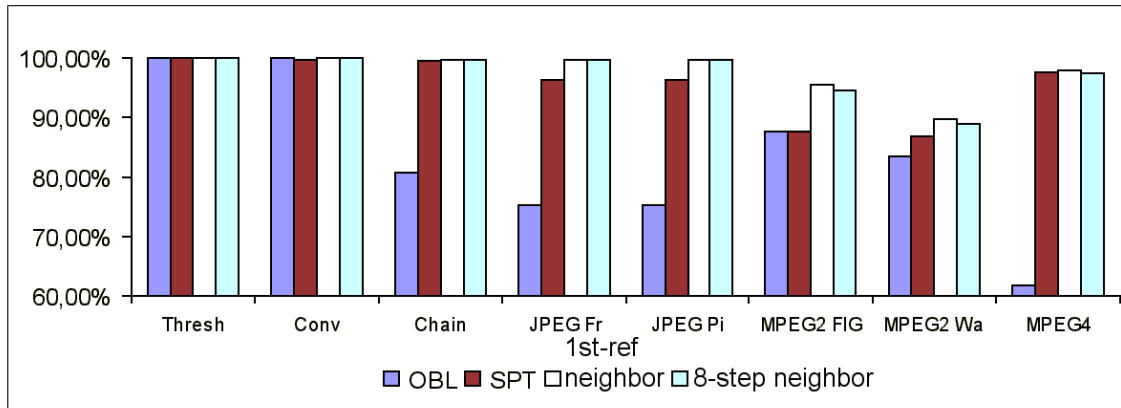
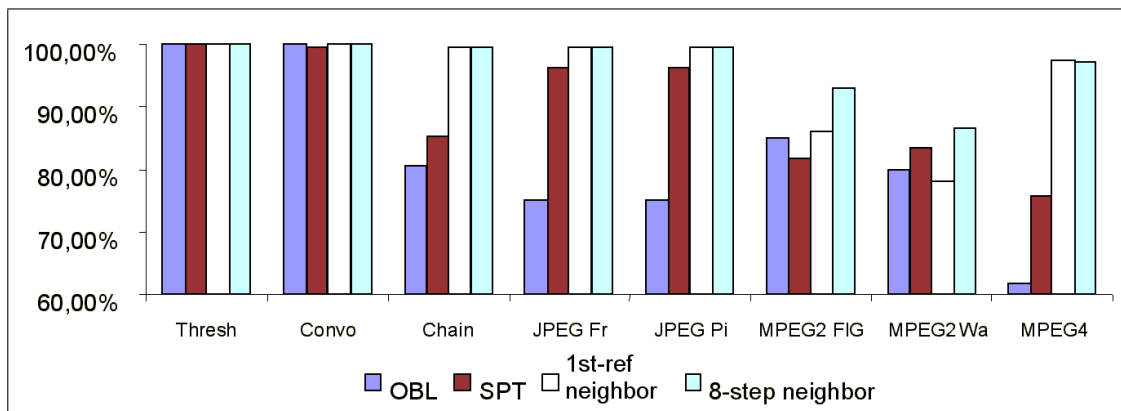
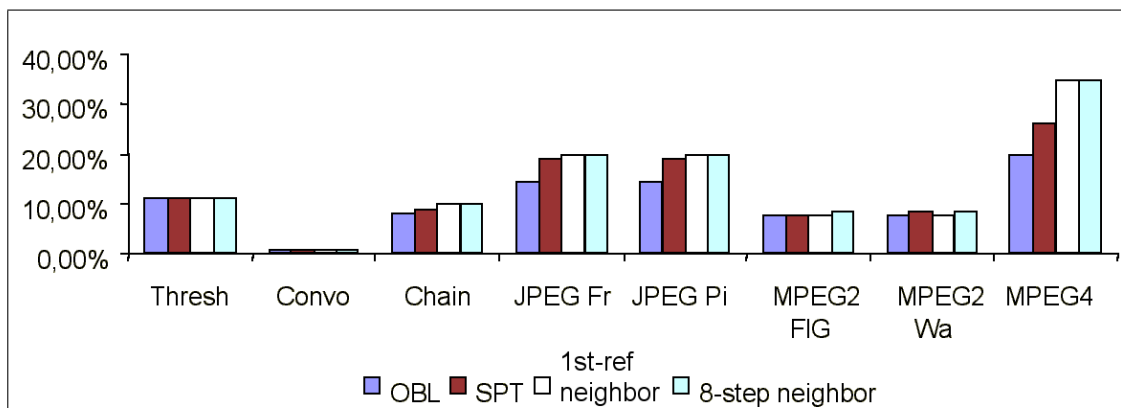


(a) Locality of Chain



(b) Locality of MPEG2 on Waterski video

Fig. 3. Demonstration of 2D spatial locality

Fig. 4. Miss-based efficiency η_i (in percentage)Fig. 5. MADT efficiency η_i^T (in percentage)Fig. 6. MAT speedup ξ_i^T (in percentage)

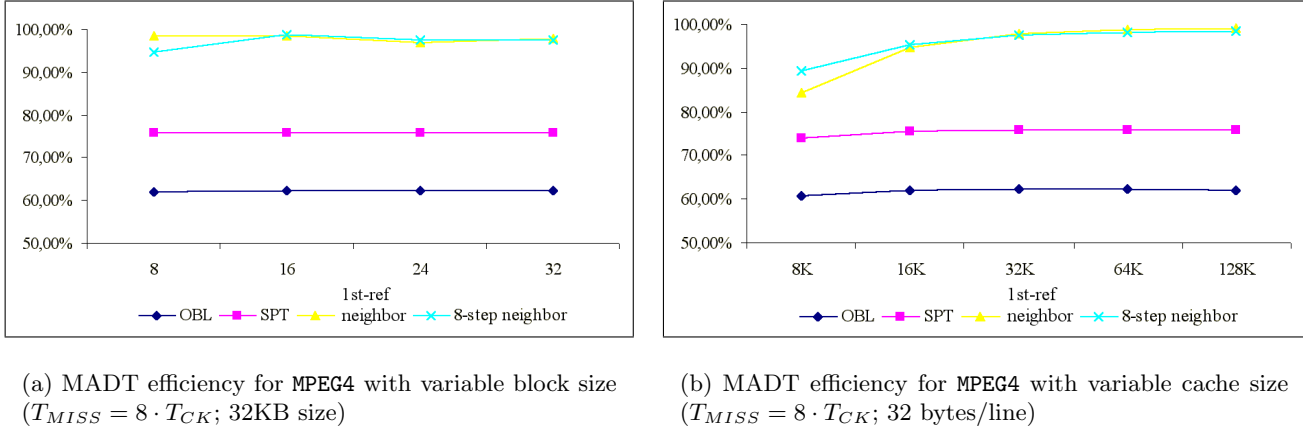


Fig. 7.

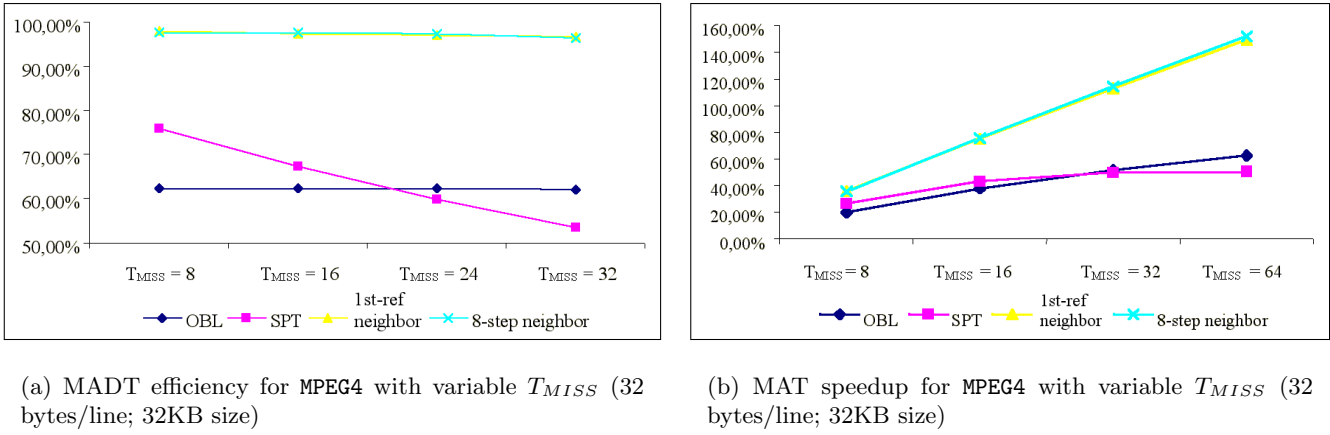




Fig. 8.

	Possible prefetching address (A_0 is the current block address)	Nr. of prefetches attempted	Nr. of prefetches issued
No-prefetch	-	0	0
OBL	$B(A_0)+1$	1	0-1
adaptive	$B(A_0 + S)$	1	0-1
1st-ref neighbor	$B(A_0)+1, B(A_0)-1,$ $B(A_0+NB_{row}), B(A_0-NB_{row}),$ $B(A_0+NB_{row})+1, B(A_0+NB_{row})-1,$ $B(A_0-NB_{row})+1, B(A_0-NB_{row})-1$	8	0-8
8-step neighbor	$B(A_0)+1, B(A_0)-1,$ $B(A_0+NB_{row}), B(A_0-NB_{row}),$ $B(A_0+NB_{row})+1, B(A_0+NB_{row})-1,$ $B(A_0-NB_{row})+1, B(A_0-NB_{row})-1$	0-8	0-1

TABLE I
COMPARISON OF PREFETCHING TECHNIQUES

Name	Algorithm	Application	Data access
Thresh	Pixel thresholding: for each pixel of the image, executes an action (i.e., a threshold) in function of the pixel value	binarization, color change, luminance enhancement, motion detection...	Raster-scan, data-independent
Convo	5x5 convolution: Executes an image convolution in a window of 5x5 pixels	filtering, noise cleaning, image enhancement, edge detection...	Raster-scan, local in a pixel neighborhood, data-independent
Chain	Chain code: starting from a initial pixel (e.g. an edge pixel), follows the adjacent pixels exhibiting the same property	Edge coding, information extraction from image; encoding of visual objects in MPEG-4; contour tracing...	Local in a pixel neighborhood, data-dependent
JPEG	JPEG decode: processing of 8x8 or 16x16 pixel blocks; only spatial compression is used	Decompressing JPEG still images or MJPEG videos (MPEG with I-frames only)	Partially raster-scan, partially block-based processing; data-dependent inside the pixel blocks
MPEG2	MPEG-2 decode: processing of 8x8 (16x16) pixel blocks, both spatial and temporal compressions are used	Decompressing MPEG-2 videos, for video visualization or frame processing	Partially raster-scan, partially block-based processing; data-dependent inside the pixel blocks
MPEG4	MPEG-4 decode: each visual object can be treated differently	Decompressing MPEG-4 videos, for visualization, animation of visual objects, video processing	Partially raster-scan, partially block-based processing; data-dependent inside the pixel blocks

TABLE II
THE BENCHMARK'S PROGRAMS

	FRISCO	PIRATES
		
Nr. of frames	51	280
Frame Size	160x128	160x128
Type	MJPEG	MJPEG




	WATERSKI	FLOWERG	DANCE
			
Nr. of frames	80	61	4
Frame Size	336x208	352x288	325x288
Type	MPEG2	MPEG2	MPEG2

TABLE III
JPEG, MPEG2 AND MPEG4 WORKLOAD

	N_{REF} 2D	N_{REF} Scalar	Nr. Miss 2D	Nr. Miss Scalar	Miss Rate 2D %	Miss Rate Scalar %
Thresh	524,288	1,874,661	8,192	1,622	1.5625%	0.0865%
Convo	19,504,949	6,538,099	16,370	1,518	0.0839%	0.0232%
Chain	484,491	15,023,718	6,987	2,012	1.4421%	0.0134%
JPEG Frisco	1,566,730	18,649,423	44,288	14,638	2.8268%	0.0785%
JPEG Pirates	8,601,610	117,003,333	242,259	69,609	2.8164%	0.0595%
MPEG2 FlowerG	47,474,320	85,751,703	631,784	62,809	1.3308%	0.0732%
MPEG2 Waterski	47,702,928	98,544,033	684,189	69,311	1.4343%	0.0703%
MPEG4 Dance	23,439,182	689,270,094	1,218,222	1,337,120	5.1974%	0.1940%

TABLE IV
WORKLOAD FEATURES

	Thresh	Convo	Chain	JPEG Frisco	JPEG Pirates	MPEG2 FlowerG	MPEG2 Waterski	MPEG4 Dance
No pf	8,192	16,370	6,987	44,288	242,259	631,784	684,189	609,111
OBL	1	6	1,351	11,008	60,245	78,601	113,966	231,661
SPT	1	32	40	1,590	8,688	78,282	90,166	14,826
1st-ref neighbor	1	2	12	83	428	28,385	70,734	12,968
8-step neighbor	1	2	23	83	428	35,362	75,753	16,815

TABLE V
NUMBER OF MISSES $N_{MISS_PF_i}$

	Thresh	Convo	Chain	JPEG Frisco	JPEG Pirates	MPEG2 FlowerG	MPEG2 Waterski	MPEG4 Dance
OBL	8,192	16,366	6,653	33,410	182,718	582,638	630,747	390,102
	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>25,892</i>	<i>30,714</i>	<i>652</i>
SPT	8,192	16,418	7,606	49,749	272,576	797,478	910,582	614,588
	<i>0</i>	<i>0</i>	<i>2,615</i>	<i>0</i>	<i>0</i>	<i>104,234</i>	<i>51,871</i>	<i>213,780</i>
1st-ref neighbor	8,225	16,436	11,320	45,417	248,439	777,279	1,119,519	711,234
	<i>0</i>	<i>0</i>	<i>6</i>	<i>0</i>	<i>0</i>	<i>67,154</i>	<i>84,191</i>	<i>3,715</i>
8-step neighbor	8,224	16,436	11,117	45,373	248,280	766,309	1,103,399	692,890
	<i>0</i>	<i>0</i>	<i>6</i>	<i>0</i>	<i>0</i>	<i>17,043</i>	<i>22,227</i>	<i>758</i>

TABLE VI

THE NUMBER OF ISSUED PREFETCHES N_{PF_i} AND THE NUMBER OF PREFETCHES NOT COMPLETELY OVERLAPPED

	Thresh	Convo	Chain	JPEG Frisco	JPEG Pirates	MPEG2 FlowerG	MPEG2 Waterski	MPEG4 Dance
No pf	65,536	130,960	55,896	354,304	1,938,072	5,054,272	5,473,512	4,872,888
OBL	8	48	10,808	88,064	481,960	748,250	1,095,475	1,857,230
SPT	8	256	8,165	12,720	69,504	931,044	899,561	1,188,205
1st-ref neighbor	8	16	137	664	3,424	705,661	1,182,970	130,392
8-step neighbor	8	16	184	664	3,424	359,468	728,157	138,596

TABLE VII

THE TOTAL MEMORY ACCESS DELAY TIME ($MADT * N_{REF}$)