High Efficiency Reconfigurable Cache for Image Processing

Zahir Larabi, Yves Mathieu, Stéphane Mancini

High Efficiency Reconfigurable Cache for Image Processing

Z. Larabi[1], **Y. Mathieu**[1], **and S. Mancini**[2]
[1]Institut TELECOM, TELECOM ParisTech, CNRS LTCI, 46 rue Barrault 75634 Paris , France
[2]GIPSA-Lab, INPG,46 Avenue Félix Viallet, 38031 Grenoble Cedex, France

Abstract— *System On Chip designs commonly use high performance data processing engines able to execute hardwired algorithms. While the performance of these engines heavily relies on the bandwidth of accesses to external memories, traditional cache architectures and algorithms suffer a lack of effectiveness for highly structured data like in 2D or 3D image processing. In a previous work, Mancini and Eveno proposed the nD-AP Cache (n-Dimensional Adaptive and Predictive Cache) in order to target multidimensional data processing. It has been shown that this cache is efficient for applications where data fetches are performed based on the history of data values. Although, the performance depends strongly on the way that the nD-AP Cache running parameters are tuned, no predefined methodology to set these parameters has been proposed before. In this paper, we study the parameters tuning aspect. Then, we compare the efficiency of the nD-AP Cache to three associative caches. Numerical results indicate that 100% improvement in run time performance can be achieved while keeping relatively low hardware cost.*

Keywords: Configurable cache, structured data, nD-AP Cache.

1. Introduction

One of the most challenging problems to design a SoC is to find an adequacy between computing power and data flow. This is becoming more and more complex as the density of digital systems has been increasing faster than the bandwidth with external memory. As a consequence, a common strategy to provide a high data bandwidth to data processing engines (DPE) is to design a hierarchy of embedded memories to cache external data.

A huge effort has been made to optimize memory hierarchies, design efficient prefetching strategies and tune memory architectures to applications, which can be made statically (off-line) and dynamically (on-line). Self-tunable caches are also a promising technology to free the designer from cache management.

S. Mancini and N. Gac in [17], [13], [16] propose the use of the nD-AP Cache (n-Dimensional Adaptive and Predictive Cache) for different signal processing applications using an empirical approach that does not take into account the global analysis of the performance, neither a method to tune the parameters. In this paper, we propose to study these two aspects.

The nD-AP Cache is suitable for a large class of applications that are fetching data from an n-dimensional data structure. It is targeted to be used in the context of applications specific hardware (FPGAs or ASICs) where the nature of the algorithms is clearly identified. The prefetching strategy of the nD-AP Cache is independent from the applications and can be tuned with few parameters.

The outline of the paper is organized as follows. In section 2, we present some of the existing prefetching techniques in structured data and self-tunable cache memories. In section 3, we present a model of the targeted application. The nD-AP Cache architecture is described in section 4. In section 5, we show the tracking mechanism and a methodology for the tuning of the nD-AP Cache memory. Results of hardware complexity and cache performances are discussed in section 6. Section 7 concludes the paper.

2. Related Work

Cache architectures for general purpose processors have been optimized to deal with structured data and especially for multimedia applications [21], [6]. The best performances are reached when the cache architecture, the data structures as well as the application match.

2.1 Cache architectures for structured data management

Performance's gains may come from a suitable static parameterization of the cache (number of lines, size of line, replacement policy), prefetching strategies and dynamic reconfiguration of the cache's parameters [19], [15].

The challenge of a prefetching strategy is to estimate the cache lines to prefetch from an analysis of the past fetches, without the knowledge of the initial data structure.

The One Block Lookahead (OBL) [20] technique fetches consecutive cache lines based on the reference causing a cache miss. Stride Prediction Table (SPT) [12], used in the Intel-Core processor [9], associates to each load instruction the previous fetched address to compute the stride with the new reference address. SPT prefetches the line a stride ahead the current reference. Although SPT is efficient for high-end micro-processors, it is too complex for specific hardware, FPGA targets and embedded systems because it needs an additional associative memory to store the loaded instructions and the associated tags.

The dynamic tuning of the cache memory tries to optimize the efficiency of applications for which memory access patterns may vary in time.

[1] proposes to reconfigure a tunable cache when a phase transition is detected at fixed intervals. The reconfiguration process needs an exhaustive search of the available cache parameters to reduce the miss rate.

Some knowledge about the pattern access may lead to efficient prefetching mechanisms. As an example, texture caching for 3D rendering benefits of some assumptions about the access pattern [18], [5]. Some information about the size of an image can also be used to exploit 2D locality and perform neighbor prefetching [7]. [14] reports satisfying results of a Markov predictor based prefetching but the important memory's need to store the matrix of transition probability makes it impracticable.

Specific caching hardware can be implemented, more or less tighten to the application. The most obvious strategy is to pipeline computations and memory accesses. But it makes little use of the fetch coherency and parallelization is difficult. Similar to pipelining, deterministic caching [8] analyzes a part of the fetch sequence to compute the needed data. It may be of low overhead but some memory is necessary to store the fetch sequence and the corresponding intermediate internal variables. On-line cache accesses with a prefetch mechanism is the most efficient way to reach a high throughput with a low pipeline latency.

2.2 Optimization of applications for cache efficiency

Applications have to be transformed in such a way that they produce fetches in a cache friendly way: the next iteration of a loop has to produce a fetch at an address close to the previous one. The main results we can find in the literature are about the transformation of nested loop when data indexes are affine functions of loop indexes [4]. Tiling is another popular optimization which decomposes a loop into a higher level loop to produce tiles and an inner loop in each tile [10]. Furthermore, the combination of the transformations of an application together with a remapping of the data structure in the memory can lead to a high cache efficiency of a direct mapped cache, which is of low hardware cost [4]. Another way could be using a software controlled prefetch associated to ScratchPad Memory (SPM) [2]. These solutions are shown to be efficient at the expense of a lack of genericity, and long software development time for the SPM management update.

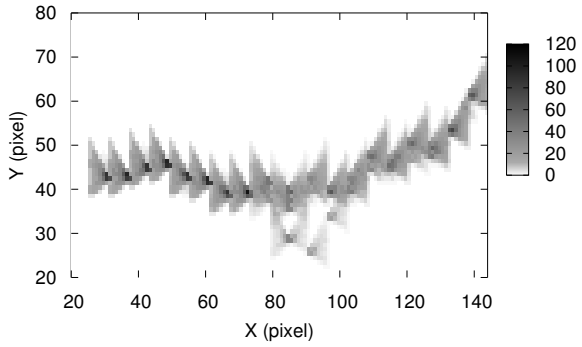
3. A model of targeted data access applications

A typical example of application, for which the nD-AP Cache is intended for, is shown in figure 1. This figure corresponds to the Jumping snake algorithm which is used to find a lip border [11]. Figure 1(a) shows the density of memory references to the 2D image , along the path defined by the algorithm. Figure 1(b) shows the temporal successions of the references over the Y axis. As we can notice in this figure, we can assume that the displacements in that 2D data structure are the sum of:

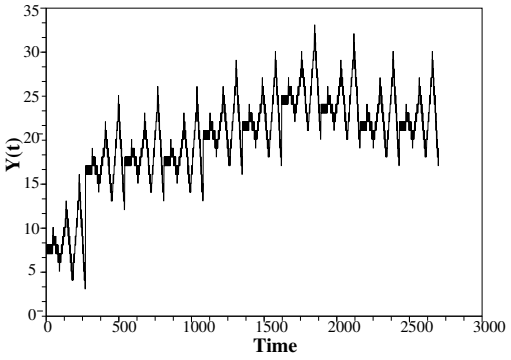
- A low speed global displacement.
- A high speed local displacement around the low speed one.

An ideal cache should be able to contain the data corresponding to local displacements and predict the global displacement in order to update the cached data.

Targetted applications should fit this model or should be transformed to comply with it.



(a) Memory access density in the 2D image



(b) Temporal succession of the position of the image over Y axis

Fig. 1: Jumping snake algorithm displacements

4. The nD-AP Cache architecture

The nD-AP cache's aim is to cache multidimensional data and prefetch zones of data according to the estimation of future references made by the prefetching mechanism, called a *tracker* hereinafter [17]. It also provides a virtual

interface to the computing unit that issues multidimensional indexes in the data structure. The cache performs both the memory mapping between indexes and the external memory addresses and the internal memory.

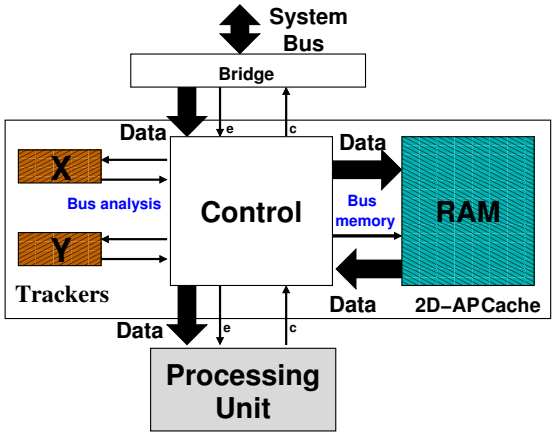


Fig. 2: A nD-AP Cache memory; in a 2D configuration

As illustrated in the figure 2, the cache is made of:

- **Trackers X, Y** : They estimate the center of the zone of data to cache and prefetch according to a model of the fetch sequence.
- **A Cache Control** : It performs the memory mapping of the 2D coordinates into memory addresses and, loads zones of data upon requests of trackers.
- **Embedded memory** : It contains the cached data. It is a double port RAM.
- **External bus interface (Bridge)**: It grabs zones of data from the external memory when requested by the control unit. It should be noted that it can be replaced by a bridge to a higher level cache.

The cache updates are performed concurrently with the cache accesses thanks to a double port memory. Conflicts are avoided thanks to the update mechanism of the cached zone. In opposition to traditional cache architectures, the nD-AP Cache control uses one index per dimension: for a 2D-AP Cache, the data structure is viewed as a rectangle (2D object), like depicted in figure 3, and available through two indexes X and Y. A cached zone is defined by an upper and a lower bound on each dimension.

For each reference, the trackers X and Y estimate the best cache center e_i (see figure 3) using the values of the previous and current references. If this estimated cache center is out of the guard zone, the real cache center C_i is updated to e_i: the cache control unit requests new parts of data to be cached from the bus interface. A cache miss occurs when the index is outside of the zone of data in cache (cached zone).

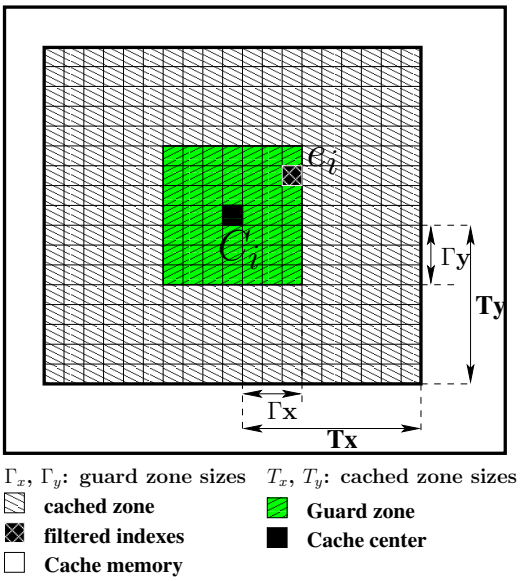


Fig. 3: The 2D-AP Cache zones

5. Tuning an nD-AP Cache memory

In this section, we present an off-line algorithm that computes the running parameters of the Statistical, first order, Constant speed & size (SC) tracker. This algorithm is a starting point to understand how trackers can be dynamically set. The entry of that algorithm is a part of a fetch sequence, called the *reference sequence* hereinafter. We first describe more precisely the SC tracker and the algorithm follows.

5.1 The SC tracker mechanism

Assuming a uniform distribution of the references around the mean, we can expect that most of the references will be included in a bounded area around the mean for a short period of time. A tracking mechanism is used to update the cache position each time the computed mean is too different from the current cache center. We define a guard zone around the current cache center as shown in figure 3. The cache center is updated when the estimated mean crosses the border of that zone. The cached zone size can be estimated from the variations around the mean.

At the i^{th} reference, the state of a SC tracker is described by:

- $s = \{s_i\}$ the fetch sequence,
- c_i the actual center of the cached zone,

In order to compute the sequence of estimated centers e of the cached zone, a low pass IIR filter F_a is used on the fetch sequence s. Multiplications can be avoided when the coefficients of such a filter are of a power of 2. The transfer function of F_a has the form:

$$H_a(z) = \frac{2^{-a}}{1-(1-2^{-a})z^{-1}} \quad (1)$$

where $a \in \mathbb{N}$ is the cut-off frequency parameter of this filter. Then, $e = F_a(s) = \{e_i\}$.

The nD-AP Cache is parameterized by (a, T, Δ, Γ) where:

- T is the size of the cached zone,
- Δ is the tracker speed,
- Γ is the size of the guard zone.

For a short time after the i^{th} fetch, the fetch sequence is supposed to evolve in a range $[c_i - \frac{T}{2}, c_i + \frac{T}{2}]$. If the estimated center e_i gets out of the guard zone $[c_i - \frac{\Gamma}{2}, c_i + \frac{\Gamma}{2}]$, then we make the assumption that the fetch sequence to come will be in the direction of the crossed border. The actual center c_{i+1} then is updated to $c_i + \Delta$ or $c_i - \Delta$.

Figure 4 shows a part of the SC Tracker behaviour for the Snake algorithm along the time. It illustrates the displacements of the cache and shows its zones (Cached range, Guard zone) around the sequence.

5.2 Tuning the SC tracker

A frequential and temporal analysis is performed to compute the set SC(a, T, Δ, Γ) from a reference sequence.

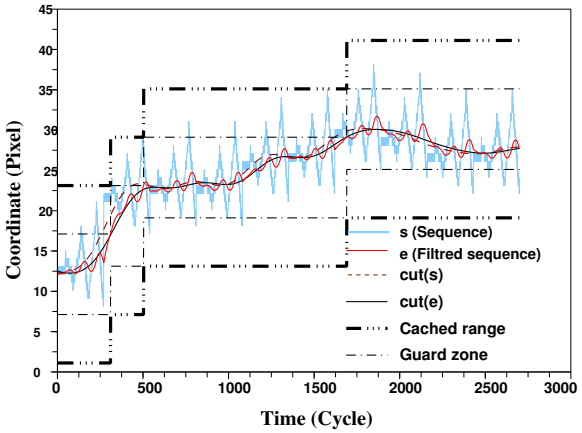


Fig. 4: Reference sequence of fetches along time

From the figure 5, which is the normalized FFT (Fast Fourier Transform) of the reference sequence of figure 4, we can see that:

- The lowest frequencies are the one tracked by the SC tracker (global displacement),
- The intermediate frequencies are the residual oscillations around the cut-off frequency of filter F_a (sub optimal filtering),
- The high frequencies are the fetches around the cache center that need to be inside the cached range (high speed displacement).

A small set of values of a are of interest because the normalised cut-off frequency f_a of the filter F_a is equal to $f_a = \frac{1}{2\pi} \arccos(1 - (2^{2a+1} - 2^{a+1})^{-1}))$. Indeed, $a_{\max}$ is

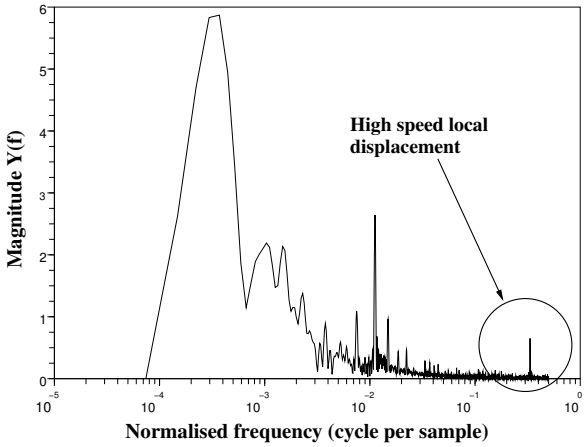


Fig. 5: Normalized spectrum of the reference sequence

the maximum value of a such that $f_{a_{\max}} N > 1$, where N is the length of the reference sequence.

The optimal value of a should verify a minimum cache size T and a good selectivity of the tracked component. For each value of a in the set $\{0, \ldots, a_{\max}\}$:

- $T_a = \max_i(|s_i - e_i|)$ is computed,
- a is chosen as the maximum value of a that gives minimum T_a. This last condition represents a compromise between the size of the cache and the amplitude of the residual oscillations after F_a.
- Γ is such that *osc*, the residual oscillations of the estimated center e_i, are in the guard zone.

$$osc = \{cut_a(e_i) - e_i\} \tag{2}$$

Where cut_a is the ideal filter of cut-off frequency f_a:

$$cut_a(x) = \begin{cases} 0 & for \quad x \in [-\frac{1}{2}, -f_a] \\ \text{FFT}(x) & for \quad x \in [-f_a, f_a] \\ 0 & for \quad x \in [-f_a, -\frac{1}{2}] \end{cases} \tag{3}$$

The optimum value of Γ is the maximum allowed oscillation: $\Gamma = \max_i |osc_i|$.

- The Δ parameter represents the average speed of the fetch sequence and the phase shift of F_a to compensate. This is the most difficult parameter to estimate and, as a first estimation, it is set to the mean phase of the F_a filter:

$$\Delta = E(|cut_a(s) - e|) \tag{4}$$

6. Results

In this section, we present measures of the nD-AP Cache efficiency and complexity. The nD-AP Cache is designed both in SystemC for high speed simulation and VHDL RTL for implementation. It has been successfully implemented

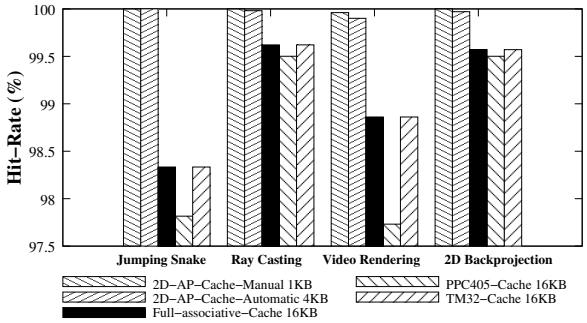


Fig. 6: Measured hit rate (memory latency = 15 cycles)

on a SoPC (System On Programmable Chip) prototype and validated with a Avnet Virtex II Pro Development board. Also, it has been synthesized for an ASIC target.

6.1 Cache efficiency

There are several ways to measure a cache efficiency depending on the target specifications. One can measure the ratio between the number of total memory references and the number of clock cycles to get all the data, the hit rate, the bus occupancy, the power consumption, etc ... In this paper, we focus on the timing performance given by: Efficiency $= \frac{\#\text{references}}{\#\text{cycles}}$, which takes into account the time to initialize the cache as well as the memory latency. After this initialization, the cache can achieve the efficiency of 1 if all the references are found in the cache.

In a standard cache, the memory latency has no impact on the hit rate. In the nD-AP Cache, two kind of miss can occur:

- A miss due to the bad tracking of the global displacement,
- A miss due to the time needed to update the cache.

If we consider only the tracking algorithm, figure 6 shows that the hit rate of the nD-AP Cache is near perfect. The efficiency as defined before is therefore a more objective way of comparing the nD-AP Cache to other architectures.

Performances are measured for several applications such as:

- Lip tracking; Jumping Snake: This algorithm is used to find a lip border on an image [11]. The sequence of fetches depends on the value of the data to optimize a gradient flow of a piecewise line.
- 2D & 3D Backprojection: These algorithms are used in medical imaging [3]. The fetch paths are deterministic but numerous.
- Ray Casting : With this algorithm used for 3D visualization a set of lines propagates in a 3D grid. The sequence depends on the point of view and the iterative behavior of the algorithm prevents deterministic caching.
- 2D tile based video rendering : These algorithms perform image transformations and compositions. Tiles of images are loaded in embedded buffers, the 2D-AP Cache acts as a 2nd level cache.

Figure 7 gives the curves of the cache efficiency depending on the system bus (32 bit bus) latency, for the aforesaid applications. These results are given by the cache parameters computed with the method from section 5. The nD-AP Cache efficiency is compared with an ideal model of the following caches:

- Full Associative, 16K, 256 lines of 16 words.
- TM32 cache, 16K, 2 way set-associative, 256 lines of 16 words.
- PowerPC 405 cache , 16K, 8 way set-associative, 512 lines of 8 words.

The results demonstrate that the nD-AP Cache is better in terms of cache efficiency or cache size than a standard cache in several cases, or sometimes allows a trade-off. The tool presented in section 5 gives as good results as the manual setting while avoiding tedious simulations.

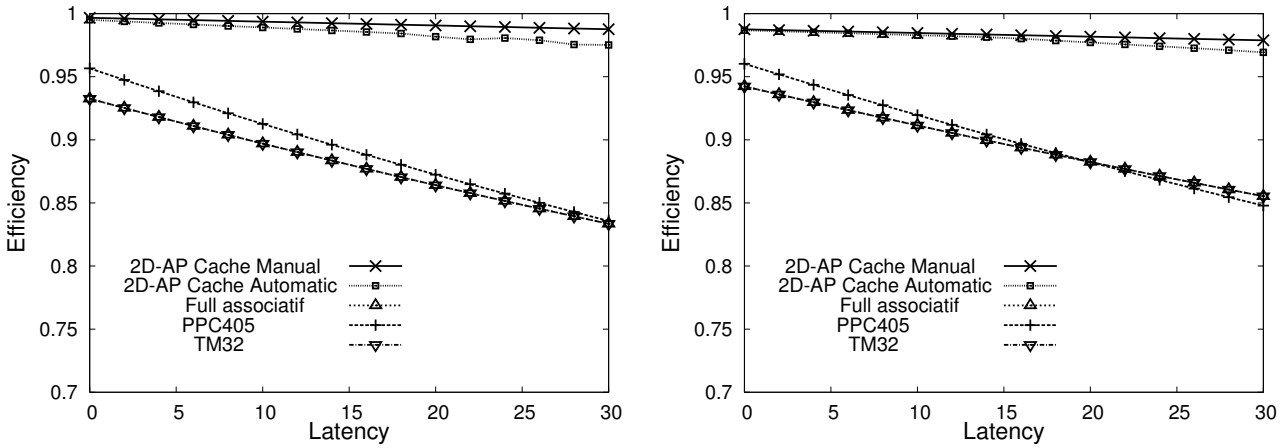
An interesting result is the video rendering that is an IP that was designed previously prior to the nD-AP Cache by an other team. The nD-AP Cache acts as a 2nd level cache and appears to be efficient. The other applications were transformed to exhibit 2D locality but the video rendering example shows that such locality is often naturally present. The reuse of data is relatively low (high speed movement of the cache center) which makes the cache much more sensitive to memory latency. However, the performance remains more efficient than a standard cache(100% improvement).

2D Backprojection and Ray Casting provide almost an ideal performance. For a wide range of memory latencies, the prefetch realized by the cache corresponds exactly to the need of the application. Excellent results are achieved, in part,thanks to the high rates of data reused by these algorithms (automatic parameterization gives easily satisfying results).

Finally, the case of the Snake shows the limitations of the proposed tracking. The residual oscillations of the filter imposes a large guard zone. That limits the prediction's performance and makes it more sensitive to memory latency. This seems to be related to the phase shift of the low pass filter that prevents the tracker to predict the next references on time.

6.2 Complexity

Table 1 gives the complexity results of the nD-AP Cache for a typical applications and an unconstrained logical synthesis. The synthesis tool reports a 170 MHz frequency for the Virtex 4 FX target and 350 MHz for a 65 nm IC (Integrated Circuit) process. These performance can be greatly increased with a suitable pipelining of the trackers and of the control unit.



(a) 2D Backprojection: man and auto 0.5KB

(b) Ray Casting: man and auto 1KB

(c) Snake: man and auto 0.5 KB

(d) Video rendering: man 0.5KB, auto 4KB

Fig. 7: Cache efficiency of a single 2D-AP Cache, together with the cache size, for manual and automatic setting of the parameters

The hardware complexity and timing of the cache control are almost independent of the size of the embedded memory, contrary to a standard cache. The complexity and the timing evolve with the number of bits to code a coordinate. At the opposite, a standard cache complexity evolve with the number of the cache lines.

Unit	Virtex 4	65 nm
Control Unit	853 FG, 280 DFF	7700 μm^2
Tracker	216 FG, 49 DFF	1900 μm^2
RAM	4 KB	

Table 1: The 2D cache complexity

7. Conclusion & perspectives

This paper presents the nD-AP Cache architectures and an associated methodology to compute its running parameters. The nD-AP Cache is a new trade-off between the hardware complexity of the control unit, the size of embedded memory and the cache efficiency. Several prefetching mechanisms and models of fetch sequence are available and the system designer can choose the one that fits its application best. The tracker presented in this paper can be automatically tuned and shown to be efficient for several applications.

The two major drawbacks of the simple filters already used are residual oscillations and prediction delay. In our future research, the introduction of Kalman like filtering will be investigated in order to enhance the performances while keeping relatively low hardware complexity. Auto tunable trackers are also a way of investigating giving the opportunity to dynamically compute the nD-AP cache parameters. This preliminary work is still on-going and the nD-AP cache is gaining new features and is evaluated on other applications.

References

[1] Rajeev Balasubramonian, David Albonesi, Alper Buyuktosunoglu, and Sandhya Dwarkadas. A dynamically tunable memory hierarchy. *IEEE Transactions on Computers*, October 2003.

[2] Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee, M. Balakrishnan, and Peter Marwedel. Scratchpad memory: design alternative for cache on-chip memory in embedded systems. pages 73–78, 2002.

[3] Bernard Bendriem and David W. Townsend. *The Theory & Practice of 3D PET*. Kluwer Academic Publishers, 1998.

[4] Francky Catthoor, Koen Danckart, Chidamber Kulkarn, and al. *Data Access and Storage Management for Embedded Programmable Processors*. Kluwer Academic Publishers, 2002.

[5] Seunghyun Cho, Chang-Hyo Yu, and Lee-Sup Kim. An efficient texture cache for programmable vertex shaders. In *IEEE ISCAS 2006*.

[6] R. Cucchiara, M. Piccardi, and A. Prati. Exploiting cache in multimedia. *Multimedia Computing and Systems, 1999. IEEE International Conference on*, 1:345–350 vol.1, Jul 1999.

[7] R. Cucchiara, M. Piccardi, and A. Prati. Improving data prefetching efficacy in multimedia applications. *Multimedia Tools and Applications*, 20(3):159–178, 2003. Kluwer Academic Press.

[8] C. Cunat, J. Gobert, and Y. Mathieu. A coprocessor for real-time mpeg4 facial animation on mobiles. In *Proc. of ESTIMedia*, 2003.

[9] Jack Doweck. IntelÂő smart memory access: Minimizing latency on intelÂő coretm microarchitecture. *Technology @Intel Magazine*, Sept. 2006.

[10] H. Dutta, F. Hannig, and J. Teich. Hierarchical partitioning for piecewice linear algorithms. In *Proceedings of the International Symposium on PARELEC*, pages 153–160, 2006.

[11] N. Eveno, A. Caplier, and P-Y Coulon. Automatic and accurate lip tracking. *Circuits and Systems for Video technology, IEEE Transactions on*, May 2004.

[12] J.W.C. Fu, J.H. Patel, and B.L. Janssens. Stride directed prefetching in scalar processors. *Microarchitecture, 1992. MICRO 25., Proceedings of the 25th Annual International Symposium on*, pages 102–110, Dec 1992.

[13] N. Gac, S. Mancini, and M. Desvignes. Hardware/software 2D-3D backprojection on a SoPC platform. In *ACM Symposium on Applied Computing*. ACM, April 2006.

[14] D. Joseph and D. Grunwald. Prefetching using markov predictors. *IEEE Transactions on Computers*, 48(2):121 – 133, 1999.

[15] D. Kim, R. Managuli, and Y. Kim. Data cache and direct memory access in programming mediaprocessors. *Micro, IEEE*, 21:33–42, 2001.

[16] S. Mancini and M. Desvignes. Efficient memory management for Ray Casting. In *DASIP'07*, 2007.

[17] S. Mancini and N. Eveno. An IIR based 2D adaptive and predictive cache for image processing. In *DCIS 2004*, page 85, November 2004.

[18] Se-Jeong Park and al. A reconfigurable multilevel parallel texture cache memory with 75-gb/s parallel cache replacement bandwidth. *IEEE Journal of Solid-State Circuits*, May 2002.

[19] D.A. Patterson and J.L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Francisco, 2nd ed. edition, 1996.

[20] Alan Jay Smith. Caches memories. *Computing Surveys*, 14:473–530, September 1982.

[21] S. Wong, S. Cotofana, and S. Vassiliadis. General-purpose processor huffman encoding extension. *Information Technology: Coding and Computing, 2000. Proceedings. International Conference on*, pages 158–163, 2000.