Performance Impact of Unaligned Memory Operations in SIMD Extensions for Video Codec Applications

Mauricio Alvarez*, Esther Salamí*, Alex Ramírez*† and Mateo Valero*†
*Department of Computer Architecture. Universitat Politècnica de Catalunya (UPC), Spain
European Network of Excellence on High-Performance Embedded Architecture and Compilation
Email: {alvarez,esalami}@ac.upc.edu
†Barcelona Supercomputing Center-CNS, Spain
Email: {alex.ramirez,mateo.valero}@bsc.es

***Abstract*—Although SIMD extensions are a cost effective way to exploit the data level parallelism present in most media applications, we will show that they had have a very limited memory architecture with a weak support for unaligned memory accesses. In video codec, and other applications, the overhead for accessing unaligned positions without an efficient architecture support has a big performance penalty and in some cases makes vectorization counter-productive. In this paper we analyze the performance impact of extending the Altivec SIMD ISA with unaligned memory operations. Results show that for several kernels in the H.264/AVC media codec, unaligned access support provides a speedup up to 3.8X compared to the plain SIMD version, translating into an average of 1.2X in the entire application. In addition to providing a significant performance advantage, the use of unaligned memory instructions makes programming SIMD code much easier both for the manual developer and the autovectorizing compiler.**

I. Introduction

Multimedia applications have become a very important workload, both in general purpose and embedded computing application domains [1]. In microprocessors for desktop computers and in most embedded media processors, the common approach for dealing with the requirements of digital video processing, and other multimedia applications, has been the extension of the Instruction Set Architecture (ISA) with SIMD instructions tailored to the operations and data types present in multimedia kernels [2].

However, most SIMD media extensions have a limited memory architecture which provides access to only contiguous data in memory, with strong alignment restrictions and a weak support for partial load and stores [3], [4]. These architectures, either do not provide any hardware support for unaligned accesses or provide it but at the expense of a big performance penalty. Therefore, the programmer usually ends up taking care of the alignment in software which, in turn, implies an extra-overhead that reduces or inhibits the performance gains due to vectorization. Moreover, software optimizations such as data reorganization become unsuccessful in video codec applications, where motion estimation (ME) and motion compensation (MC) algorithms and variable block sizes entail unpredictable alignments.

In this paper we analyze the performance impact of providing hardware support for unaligned access in current SIMD extensions for emerging video applications like H.264/AVC decoding and encoding [5]. We evaluate an efficient hardware architecture that can deliver high bandwidth and low latency for unaligned accesses. Software support includes new instructions on top of the Altivec SIMD extension of the PowerPC architecture. Our results show that the availability of instructions for unaligned access has an important speed-up in some kernels, and in some cases they allow the vectorization of other kernels that otherwise have to be implemented with scalar instructions.

This paper is organized as follows. In section II, we discuss the problem of alignment in video applications and we overview the existing support for unaligned accesses in current SIMD extensions. In section III, we describe the process of adding support for unaligned memory access to the Altivec extension both from hardware and software perspectives. In section IV, we depict the methodology used for the experimental evaluation and, in section V, we present some results in terms of speed-up and reduction in the number of instructions. Finally, in section VI, we present our main conclusions for this paper.

II. Support for Unaligned Access in SIMD Extensions

A memory reference is called misaligned (or unaligned) when it accesses a position that does not match with the memory access granularity of the processor. In most SIMD architectures, it is not possible or it has a big performance penalty to access an unaligned memory position. When there is an attempt to access an unaligned position, it is necessary to perform a realignment process that consist in, first, to read the aligned memory word that is located before the unaligned position and shift out the unnecessary bytes; second, to read the aligned word that is located next to the unaligned position and discard the unnecessary bytes; and finally, to merge the two parts that were extracted previously. The realignment

1-4244-1081-9/07/$25.00 ©2007 IEEE

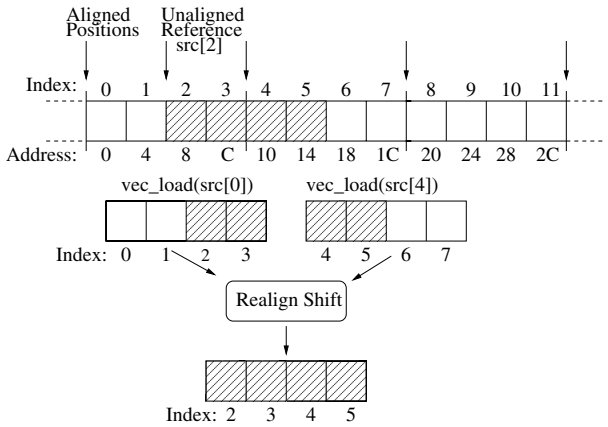


Fig. 1. Vector load from an unaligned address

process for an unaligned vector load of four elements is shown in figure 1.

The level of support for unaligned accesses in current SIMD extensions includes variations from hardware mechanisms that transparently perform the memory accesses, system exceptions that generates a call to the operating system, and instructions to do the re-alignment in software. In the domain of high performance general purpose processors (GPPs) the Intel's SSE extension is the only one that includes both hardware support and unaligned exceptions. The initial design of the SSE extension only provides support for aligned accesses, the instruction MOVDQA (Move Aligned Double Quadword) requires that the effective address have to be aligned, and in the opposite case a general protection fault is generated [6]. The SSE2 extension includes support for non-aligned accesses by providing the instruction MOVDQU (Move Unaligned Double Quadword) that allows to load and to store non-aligned 128 bit words. This instruction was implemented using two 64-bit loads (or stores) and was based on microcode; this kind of implementation results in big latencies and big performance penalties for unaligned accesses that cross cache boundaries. In the SSE3 extension another instruction was introduced in order to resolve the above mentioned problems [7]. The LDDQU (Load Unaligned Integer 128 bits) instruction performs a 32-byte load and then performs a shift to extract the corresponding 16 bytes of unaligned data. However this instruction may reduce performance if the load requires store to load forwarding and it only applies for loads [8].

In other SIMD extensions like Altivec, MIPS and Alpha, the hardware always returns aligned positions by automatically clearing the lower bits of the effective address. In these extensions, it is necessary to load the two adjacent aligned positions and to shift them in order to extract the unaligned data elements. SIMD extensions differ in the way they can generate the data necessary for the shift. Naishlos and Henderson call this value the "realignment token" [9]. The realignment token can be an address, a bit mask or any other value that is a function of the unalignment of the original address. The Altivec extension uses an approach in which

```
  src_ptr   = InputArray;
LOOP:
  alignmask = vec_lvsl(0,src_ptr);
  aligned_a = vec_ld(0, src_ptr);
  aligned_b = vec_ld(15, src_ptr);
  unaligned = vec_perm(aligned_a,
                aligned_b, alignmask);
  src_ptr  += srcStride;
END_LOOP
```

(a) Non unitary-stride ((srcStride%16)!=0)

```
  src_ptr   = InputArray;
  alignmask = vec_lvsl(0,src_ptr);
  aligned_a = vec_ld(0, src_ptr);
LOOP:
  aligned_b = vec_ld(15, src_ptr);
  unaligned = vec_perm(aligned_a,
                aligned_b, alignmask);
  aligned_a = aligned_b;
  src_ptr  += 16;
END_LOOP
```

(b) Stride-one vectors ((srcStride%16)=0)

Fig. 2. Altivec alignment code for a vector load

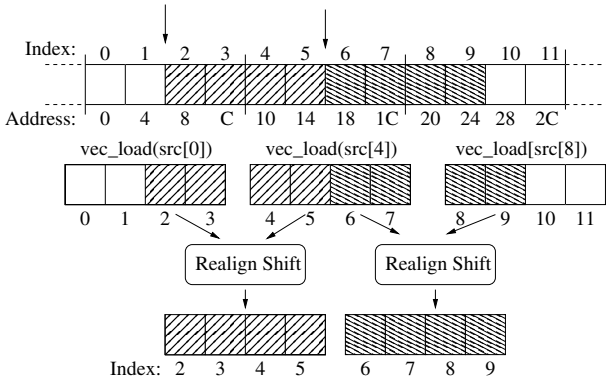


Fig. 3. Vector load from an unaligned address with stride one

the realignment token is a vector mask generated with the LVSL (Load Vector for Shift Left) instruction which is used in conjunction with the VPERM (Vector Permute) instruction to merge two aligned vectors and to produce the desired unaligned data [10]. Figure 2(a) shows the necessary code for re-alignment of a vector load, using Altivec C intrinsics.

On the other hand, most Digital Signal Processors (DSP) architectures traditionally do not provide support for unaligned accesses. The proliferation of video applications in multimedia devices have prompted the designers to enhance the memory architecture of DSPs with misaligned accesses support. The recent Trimedia TM3270 processor has included support for 32 bit non-aligned loads and stores with no-stall cycles [11], previous processors in the Trimedia series produce exceptions when trying to access a misaligned position. Due to the fact that the TM3270 has only one load/store unit if the unaligned access crosses a cache line boundary the access may result in two sequential cache misses. In the TMS320C

Texas instruments family of DSPs, the recent TMS320C64X set of processors includes support for unaligned loads and stores of 32 and 64 bit values. But when there is an unaligned memory access one of the two memory ports can not be used for memory operations and the memory system does not assure that these memory accesses will be atomic [12]. Other DSP architectures for embedded systems, like the TigerSharc, support accesses to misaligned positions by using specialized hardware units (like the Data Alignment Buffer) which performs the required aligned loads and shifts [13]. The Cell Broadband Engine has added two instructions for unaligned load and stores into the PowerPC Processor Element (PPE). Using the load instruction an unaligned load requires three instructions (one less than plain Altivec) but still two more than a single unaligned load [14]. These instructions belong to the critical path of the loop body representing a significant execution delay. The unaligned store instructions are useful for the leading and trailing edges of misaligned arrays, but not for unaligned 2-dimensional data structures like in video codec applications.

Table I summarizes the unalignment support provided by different architectures based on the scheme proposed by Nuzman and Henderson and with the addition of some media processors.

The first designs of SIMD ISAs did not include support for unaligned accesses because it has been taken as an unnecessary addition of complexity, specially for the load/store pipeline. As a way to overcome the problem of unaligned accesses some architectures, like PowerPC, included powerful permutation instructions and units that help with the problems of data reorganization within a vector register. But still there are some applications, with video processing being one of the most remarkable one, for those not having an efficient support for unaligned accesses degrade the performance significantly. For this kind of applications the extra cost in hardware complexity is more than justified.

Although currently there are processors that include some extent of support for non-aligned accesses and there are wide consensus about their importance for video applications, most of the current SIMD architectures that support unaligned accesses have restrictions and limitations that do not allow an efficient use of the unaligned instructions in all the cases. These restrictions include: microcode-based operations, short buses in internal datapaths, short bandwidth to the L1 data cache, partial support for unaligned instructions that requires several instructions for each memory access, not-supporting unaligned stores, not being thread safe, causing extra latency for crossing cache boundaries, requiring a sequential handling of more than one cache miss and having restrictions in the use of the load store units. Additionally there is not in the literature a complete evaluation of the impact of unaligned instructions in SIMD extensions using contemporary multimedia applications. We have addressed this issues by providing a high performance and efficient support for both non-aligned loads and stores and by evaluating their performance impact with a state-of-the-art video codec like H.264/AVC.

Architecture & SIMD extension	unaligned load	aligned load	realign operation	realign token
IA32 SSE1,2,3,4	movdqu, lddqu	movdqa		
PowerPC - Altivec		lvx	vperm	lvsl
Cell (PPE) - Altivec	lvlx, lvrx			
MIPS-rev2	ldl, ldr			
MIPS - MDMX		luxc1	alnv.ps	address
ALPHA		ldq_u	extql, extqh, or	address
Trimedia TM3270	ld32r			
TI TMS320C64X	ldnw			

TABLE I
SUPPORT FOR UNALIGNED LOADS IN DIFFERENT PLATFORMS

A. Compiler Optimizations Related to Memory Alignment

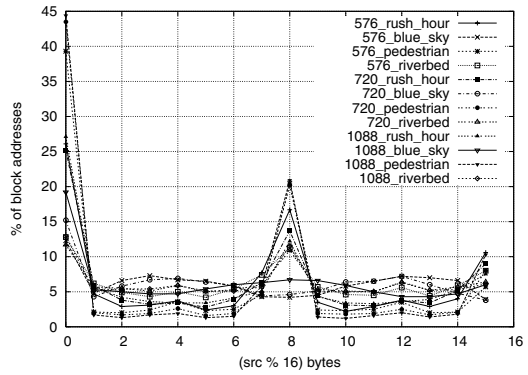
As unaligned accesses usually have more latency than aligned ones, the programmer/compiler tries to avoid them as much as possible. In some algorithms in which unaligned accesses cannot be avoided, compile-time optimizations such as loop peeling and static and dynamic detection of unalignment can still be applied [15], [16]. Additional optimizations exist for stride-one references [17]. Figure 2(b) shows a version of the loop in figure 2(a) optimized for stride-one streams. In this case, the mask for doing the permutation remains constant and has to be calculated only once. Similarly it is possible to reuse one of the aligned loads from one iteration to the next one (see Figure 3). As a result the mask generation instructions and one aligned load can be moved out the loop. (Note that in Altivec a stride-one vector means that the stride is equal to 16).

B. Unaligned Access in Video Applications

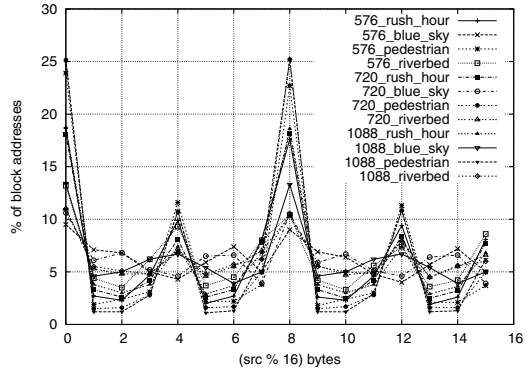
In applications like video coding and decoding that use ME and MC algorithms, it is not possible to avoid unpredictable unaligned memory references. This is due to the fact that ME performs a search and a comparison of pixel blocks within a search window. The displacements in the search window can be arbitrary and that results in a lot of unpredictable unaligned accesses [18]. Additionally, in video standards like H.264/AVC that supports variable block size for ME, it is necessary to perform unaligned stores in order to save those blocks whose size is not equal to the SIMD register width. In Altivec, 16-bytes of a 16x16 block can be stored simultaneously to an aligned memory address but for other blocks sizes, like 8x8 or 4x4, the data is naturally aligned to 8 or 4 bytes but not to 16-bytes requiring to perform partial stores of unaligned data.

Figures 4(a) and 4(b) show the distribution of unalignment offsets for Altivec loads for two kernels (luma and chroma interpolation) of the MC stage of the H.264/AVC decoder using different input videos at different resolutions. The unalignment offsets are distributed across the full range from 0 (aligned) to 15. These offsets can not be determined at compile time, and the use of optimizations like loop peeling is not well suited. Moreover, these accesses are made on a 2-dimensional pattern preventing the use of the compile time optimizations developed for linear streams.

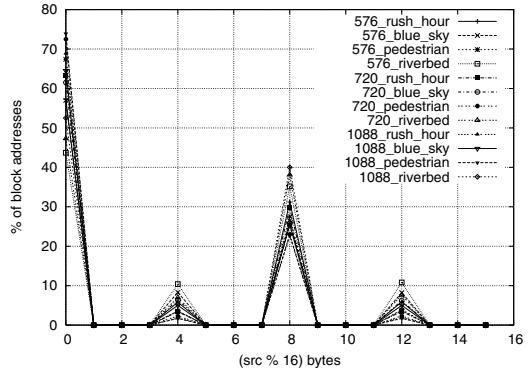
Figures 4(c) and 4(d) show the distribution of unalignment offsets for the Altivec stores for the same kernels and input sets. Here, the unalignment depends only on the block size.



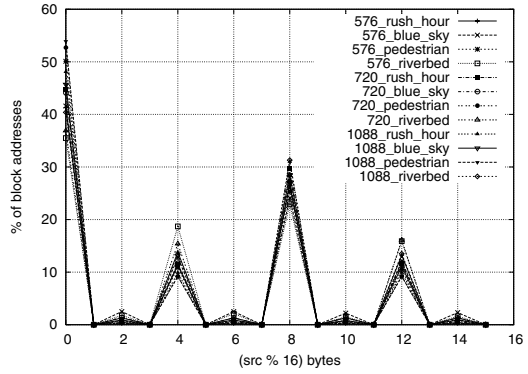
(a) luma load pointers



(b) chroma load pointers



(c) luma store pointers



(d) chroma store pointers

Fig. 4. Alignment offsets in H.264/AVC luma and chroma interpolation kernels

```
dst1 = vec_ld(0, dst);
dst2 = vec_ld(16, dst);
dstperm = vec_lvsr(0, dst);
dstmask = vec_perm(vzero_u8, neg1, dstperm);
rsum = vec_perm(sum, sum, dstperm);
fdst1 = vec_sel(dst1, rsum, dstmask);
fdst2 = vec_sel(rsum, dst2, dstmask);
vec_st(fdst1, 0, dst);
vec_st(fdst2, 16, dst);
```

Fig. 5. Altivec alignment code for a vector store

The offsets are predictable, and loop peeling can be applied efficiently. The main concern with unaligned stores is that they require a load-store sequence that can take more than 10 assembly instructions for each store and they are not atomic which implies that they are not thread safe. Figure 5 shows the Altivec intrinsics for performing an unaligned store. First, there is a sequence of instructions that performs an unaligned load, then the desired data is inserted into the loaded values (with VEC_SEL instructions), and finally, the two modified words are stored back to memory.

III. Adding Support for Unaligned Loads and Stores

In order to evaluate the impact of unaligned access support we have added SIMD instructions for unaligned loads and stores. Complete support for unaligned instructions requires modifications both in the load/store pipeline of the processor and in the compiler tool chain.

The new instructions, called LVXU (load vector unaligned indexed) and STVXU (store vector unaligned indexed), have been added to the Altivec SIMD extension. The instructions are similar to the aligned ones in Altivec; they use indexed addressing but do not impose any alignment restriction on the effective address.

The new instructions can be used directly in assembler programs and, additionally, we have added support for them as intrinsics into the GCC 4.0 compiler allowing to use them in C/C++ programs and leaving the compiler to do the register allocation, instruction scheduling and other optimizations. Direct support for unaligned accesses can be very useful for autovectorizing compilers because it simplifies the alignment detection and correction optimizations [9], [17].

From the processor hardware perspective, it is necessary to adapt the Load Store Unit (LSU) to include a realignment subsystem and to modify the interconnection between the processor and the L1 data cache (D-L1). Figure 6 shows the structure of the LSU with the addition of an alignment network unit. The architecture support for unaligned memory accesses must be added so that it does not severely impact the latency of aligned accesses, and has the minimum possible penalty for unaligned ones. Taken that into account, long latency mechanisms like microcode expansion must be avoided and the bandwidth of the D-L1 must be adapted to the vector accesses. Most of current SIMD implementations use ports to the L1 that have half of the vector width, thus having

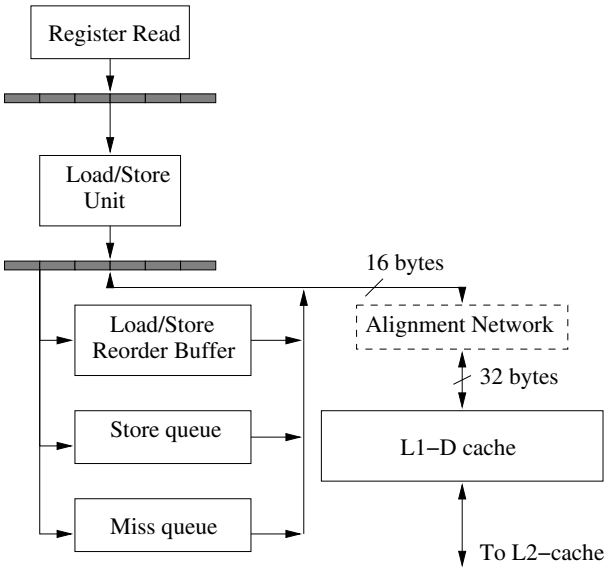


Fig. 6. Load store pipeline of the modeled superscalar processor

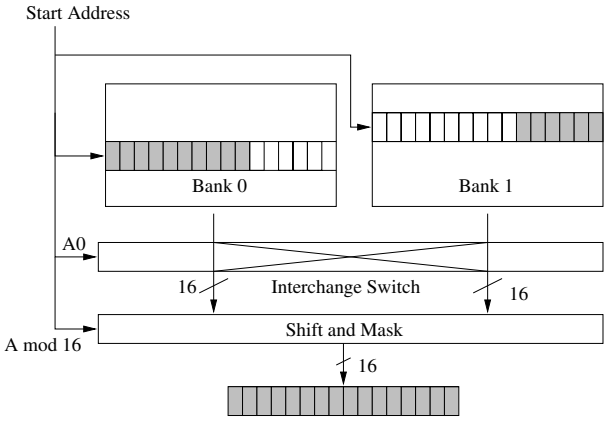


Fig. 7. Realignment unit using a two-bank interleaved cache

to perform two or more access to the L1 for each vector reference [19], [20].

The alignment network unit can be designed using a two-bank interleaved cache, so that two consecutive cache lines can be accessed simultaneously, and therefore a whole stride-one vector access, overlapped over two different cache lines, can be performed. This scheme requires three building blocks: an interchange switch, since it may be needed to swap the two cache lines, a shifter to align the lines accessed to the initial address, and a logic to mask the unused data based on the unalignment offset [21] (see figure 7). Using this scheme, the unaligned load can be performed in one cycle and the store requires an additional cycle because it first needs to shift and mask the data from the vector register and then to swap the partition for the two cache banks [22], [23]. Using such a scheme does not impose any of the restrictions that most of the current processors that support non-aligned access have. First there is not a cache line boundary penalty because there are two parallel accesses to the multi-bank cache. Neither are there forwarding restrictions with respect to the other memory operations, and the unaligned accesses are atomic from the processor perspective.

IV. Methodology

For our experiments we have selected the Altivec/VMX extension of the PowerPC architecture. Altivec is very representative of the current SIMD extensions and the results presented here can be extended to other SIMD extensions as well.

To conduct the experiments we are using a trace-driven simulation methodology using the IBM MET tools, that include an instruction emulator and trace generator based on the Aria dynamic instrumentation tool, and a cycle accurate processor simulator based on the Turandot simulator [24].

The applications and kernels are programmed using Altivec intrinsics and compiled with the GCC-4.0.2 compiler [25]. The GCC compiler and GNU assembler have been modified to include intrinsics and opcodes for the new instructions under study. Traces are collected by running the applications on the AIX-5.2 operating system using an Aria based instruction emulator. The execution trace contains PowerPC, Altivec and the new instructions added for unaligned accesses.

We have defined three different processor configurations for the simulations. The first one is a 2-way in-order processor that is somewhat similar to some current embedded media processors like the Cell SPE. The other two configurations are a 4-way and an 8-way out-of-order superscalar processors with a microarchitecture similar to the IBM Power-4 processor with the addition of the Altivec pipeline [26]. The three configurations have the same number of pipeline stages and the same configuration of the branch predictor and memory hierarchy. The basic parameters of the modeled processors are described in table II.

For our experiments we have selected the FFMPEG H264 decoder, an open source implementation of the H.264/AVC standard that is optimized for high performance including extensive use of SIMD instructions [27]. The high profile of the H.264 was selected, with CABAC, B frames, multiple reference frames, weighted prediction, and a I-P-B-B sequence of pictures. Since we are interested in analyzing the performance of H264/AVC at HD resolutions, we have selected input sequences for 720x576, 1280x720 and 1920x1088 pixels. The set of input sequences has different motion characteristics in order to cover a broad range of video content.

V. Performance Evaluation

In order to isolate the effects of unaligned instructions we have extracted some of the most important kernels of the H.264/AVC decoder and encoder [28]. These kernels are: luma interpolation, chroma interpolation, inverse transform (IDCT) and sum of absolute differences (SAD). These kernels account for 45% and 34% of the execution time of the complete H.264/AVC decoder application for the scalar and Altivec versions respectively (see section V-D). We have implemented all these kernels for three different block sizes including

Instr. x 1000	Total	Int.	Loads	Stores	Bran-ches	Altivec Load	Altivec Store	Altivec Simple	Altivec Compl.	Altivec Perm.
LUMA										
16x16_scalar	9,926	6,437	2,693	676	120	0	0	0	0	0
16x16_altivec	1,999	269	9	10	85	244	106	564	128	584
16x16_unaligned	1,438	155	2	3	51	135	50	564	128	350
CHROMA										
8x8_scalar	2,110	1,439	514	132	25	0	0	0	0	0
8x8_altivec	489	108	6	12	34	63	16	64	64	122
8x8_unaligned	388	79	6	12	23	40	16	48	64	100
IDCT										
4x4_scalar	4,984	3,074	1,121	608	181	0	0	0	0	0
4x4_altivec	2,090	532	49	48	105	112	64	448	0	732
4x4_unaligned	1,960	530	49	48	53	112	64	448	0	656
4x4_altivec_mat	1,980	486	177	32	105	256	64	128	256	476
4x4_mat_unaligned	1,832	450	177	32	53	256	64	128	256	416
SAD										
16x16_scalar	2,198	1,141	512	0	545	0	0	0	0	0
16x16_altivec	266	52	1	0	17	64	1	49	17	65
16x16_unaligned	170	52	1	0	17	32	1	49	17	1

TABLE III
DYNAMIC INSTRUCTION COUNT FOR H.264/AVC KERNELS (THOUSANDS OF INSTRUCTIONS)

Configuration	Param	2-way	4-way	8-way
Issue Policy		In-order	Out-of Order	Out-of Order
Width	Fetch-Rename-Dispatch	2	4	8
	Retire	4	6	12
	Inflight	80	160	255
Units	FX	2	3	6
	FP, LS, BR, VI	1	2	4
	VPERM, VCMPLX	1	1	2
PhysRegs	GPR, FPR, VPR	60	80	128
Queues	BR Issue	5	12	40
	Issue:	10	20	40
	Retire	80	128	160
	Ibuffer	12	24	48
D-cache	Read Ports	1	2	4
	Write Ports	1	1	2
	Mis Max	2	4	8
L1-D	Size	32KB		
	Line Size	128B		
	Associativity	2		
L1-I	Size	32KB		
	Line Size	128B		
	Associativity	1		
L2-(I+D)	Size	1MB		
	Line Size	128B		
	Associativity	8		
	Latency	12 cycles		
Main Memory	Latency	250 cycles		

TABLE II
PROCESSOR CONFIGURATIONS USED IN SIMULATION ANALYSIS

16x16, 8x8 and 4x4 pixels. Additionally we have evaluated three different implementations of each of these kernels. The first one is a scalar implementation using integer instructions, the second one is the SIMD implementation using Altivec instructions, and the third one is the Altivec implementation extended with unaligned accesses.

A. Dynamic Instruction Count

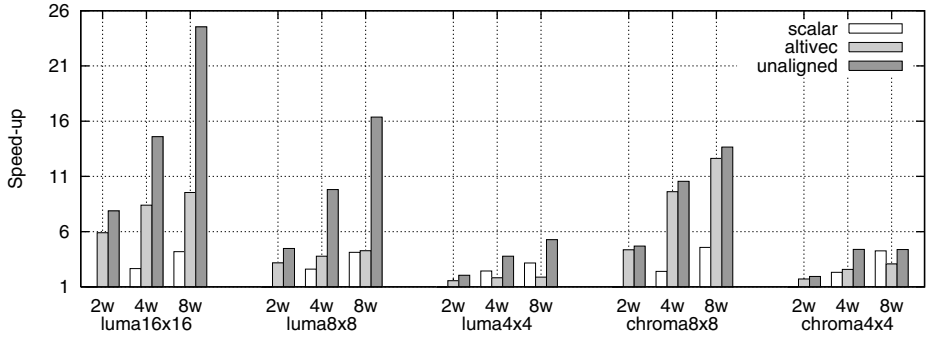
Table III shows the dynamic instruction count for 1000 executions of each kernel for one block size per kernel (the results for other block sizes are not shown due to space constraints). When comparing the scalar and plain Altivec versions, we appreciate a large reduction in the total number of executed instructions due to vectorization. When comparing the Altivec and the extended Altivec versions, we observe that the use of the new unaligned instructions adds an additional reduction (on average for all the block sizes) of 33.4%, 22.6%, 1.8% and 33.7% for the luma, chroma, IDCT and SAD kernels respectively.

The most important instruction reduction comes from the elimination of memory and permutation instructions. The permutation instructions represent an overhead of the unaligned memory operations and its reduction increases the ratio of real computation. In kernels like SAD there is an average reduction, for all the block sizes (not shown in the table) of 95% of the permutation instructions.

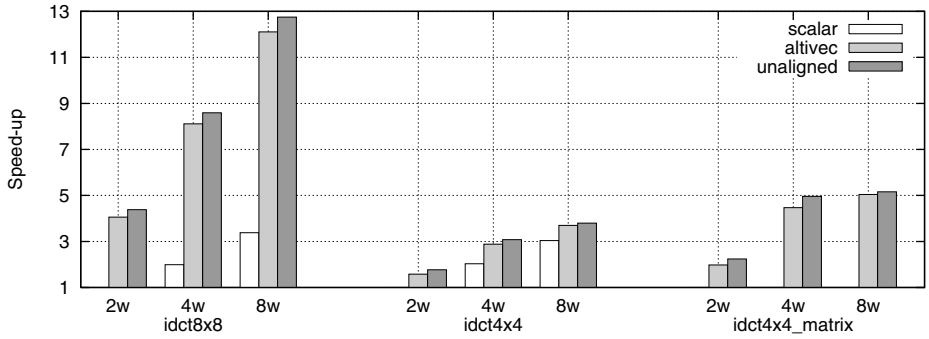
The use of unaligned instructions not only reduces the Altivec memory operations, but also the integer arithmetic and integer load and store instructions, due to the elimination of some pointer arithmetic necessary in the realignment code. Additionally, a number of branches are also eliminated from the Altivec version, because in kernels like chroma there are branches that depend on the unalignment offset of the address. On the other hand, in kernels like IDCT, in which all the input data is properly aligned by rearrangements in the source code, the impact of unaligned instructions only contributes to a small reduction of permutation instructions that are used in the final load-add-store sequence. Additionally in some kernels there is an additional elimination of branches that were used for peeling the loops in the final store sequence and which are possible to replace with a single unaligned load-store sequence.

B. Speed-up

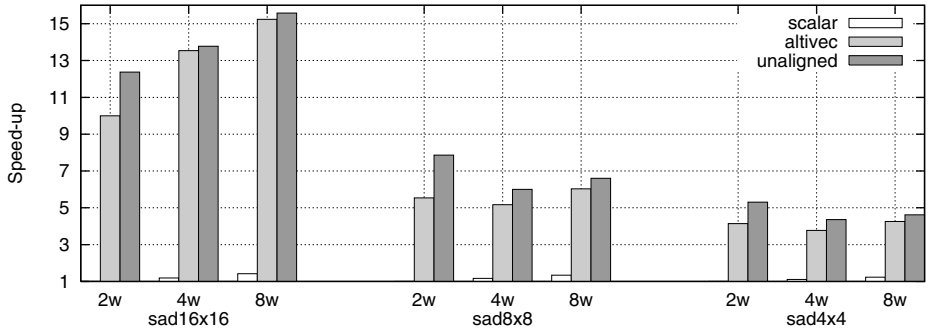
In order to analyze the potential and upper-bound speedups before dealing with implementation issues we have made an experiment in which the unaligned accesses have the same latency than the aligned ones. For our architecture that means that aligned and unaligned accesses will have a latency of 4 cycles for a D-L1 hit. In the next section we are going to analyze the effects of latency increase in the unaligned accesses due to the realignment hardware. These results can be taken



(a) Luma and chroma



(b) IDCT



(c) SAD

Fig. 8. Speed-up in kernels with support for unaligned load and stores

as an upper-bound of the speed-up achievable with unaligned instructions. The actual values for our suggested realignment network are shown in the next section, but it is important to note that more aggressive implementations are possible in embedded systems (such as the one in the Trimedia-TM3270) in which unaligned accesses are implemented with no stall cycles.

Figure 8 shows the speed-up in the execution time for all the kernels under study. All the values are normalized to the 2-way scalar version. For the Luma interpolation kernels (fig 8(a)), the Altivec version with unaligned instructions exhibits a 1.9X, 2.6X and 2.1X speed-up for the 16x16, 8x8 and 4x4 block sizes respectively. It is worth to remark that as the block size decreases the overhead of the realignment code in the Altivec version increases. In the 4x4 case, the scalar implementation has better performance than the Altivec one. In this case, the use of unaligned instructions helps to eliminate a lot of overhead inside the main loop of the interpolation routine, thus allowing to obtain a important speed-up for a kernel that otherwise would not be vectorized.

The chroma interpolation kernel has an average speed-up of 1.1X and 1.25X for the 8x8 and 4x4 sizes respectively. It should be noted that due to the YUV 4:2:0 chroma sub-sampling scheme, used by most current video codecs (included H.264/AVC), the size of the chroma blocks is 8x8, 4x4 and 2x2 pixels. The 2x2 block is not included because the available DLP is very limited.

We have evaluated three versions of the IDCT. The first one is the factorized algorithm for the 4x4 block size, in which the speed-up is 1.07X. The second one is based on a matrix product algorithm and has a speed-up of 1.09X [29]. Finally there is the 8x8 factorized transform in which the speed-up is 1.06X. The impact of unaligned instructions in the IDCT is minimal because, as noted before, the input data structures

can be properly aligned in software. The unaligned store instruction is useful here for performing partial load and stores required in the output sequence. An additional instruction for unaligned partial load and stores would be very useful for reducing the overhead for loading small data structures like 4x4 blocks. In the SAD kernel, there is an average speed-up of 1.16X for all block sizes and processor configurations. The performance gains are bigger in the 2-way configuration (going up to 1.4X speed-up for the 8x8 block version). The speed-up is saturated for the 4-way and 8-way processor because the processor does not find enough instructions to issue in parallel and has bigger branch misprediction ratios.

From the obtained results, it is clear that supporting unaligned memory access is crucial in any SIMD extension targeting multimedia workloads, where unaligned accesses can not be avoided. In those kernels suffering from unaligned addressing, the unaligned load/store support more than doubles performance. Furthermore, kernels which resulted in performance drops are now efficient at exploiting DLP, and the resulting codes are shorter and easier to write and debug, even for autovectorizing compilers.

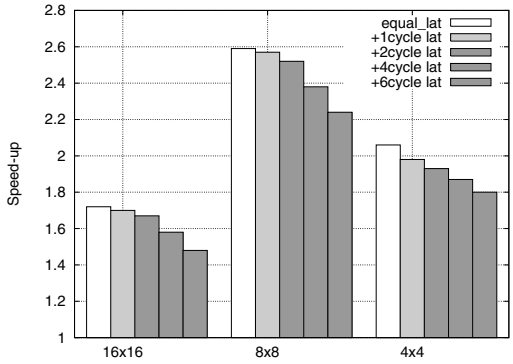
C. Impact of the Latency of Unaligned Load and Stores

The evaluations in the previous section were done assuming that unaligned instructions had the same latency as the aligned ones, that is 4 cycles in all the processor configurations. In order to analyze the effects of the latency of the hardware realignment network, we have performed an experiment in which the latency of the unaligned loads and stores is increased by 1, 2, 4 and 6 extra cycles with respect to the original ones. The resultant speed-ups compared to the original Altivec implementation are shown in figure 9. Due to space constraints, we only present results for the 4-way processor configuration. Results for 2-wide and 8-wide configurations follow the same pattern.

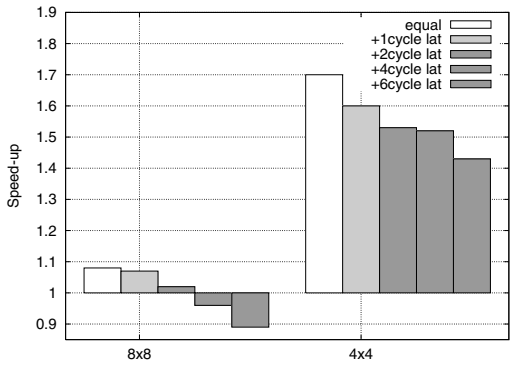
Luma interpolation is the more insensitive kernel to the latency increase. With one extra cycle, the speed-up decreases only 1.9% on average for the three block sizes. With six extra cycles of latency, the speed-up decreases by 13.2% but still achieves a 1.8X speed-up over the Altivec version.

Chroma interpolation, on the other hand, is more sensitive to the latency of the unaligned load and stores, mainly in the 8x8 block size. The speed-up reduction is very similar to that of the Luma interpolation kernel, ranging from 3.8% for one extra cycle to 17% for 6 extra cycles. But, for the 8x8 block size, when the latency increase in 8 cycles or more, the execution time becomes worse than the original Altivec version. For the 4x4 case, although there is a performance penalty with the latency increase, the use of unaligned instructions results in speed-ups even with high latencies.

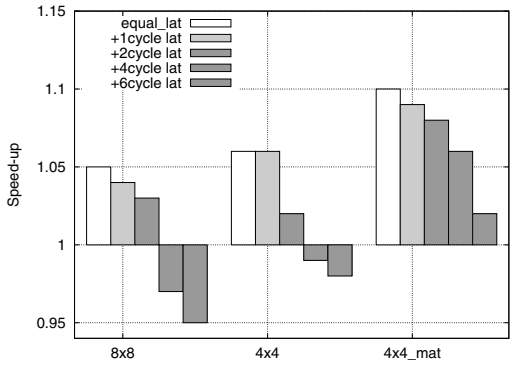
In the IDCT kernel the increase of latency affects only a few unaligned memory operations used in the final load-add-store sequence. The latency increase has a bigger impact in the 8x8 version because it has more dependent load and stores in the output sequence. It is important to note that the matrix algorithm not only has a bigger speed-up than the factorized



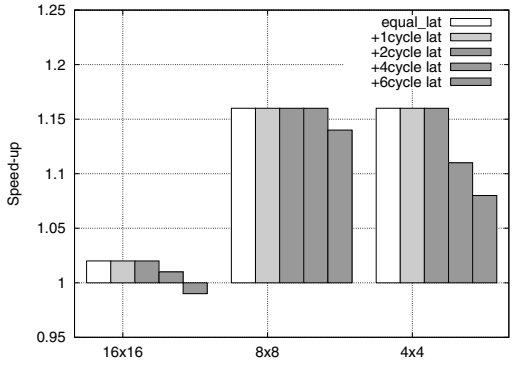
(a) Luma kernel



(b) chroma kernel



(c) idct kernel



(d) sad kernel

Fig. 9. Performance Impact of latency of unaligned load and stores

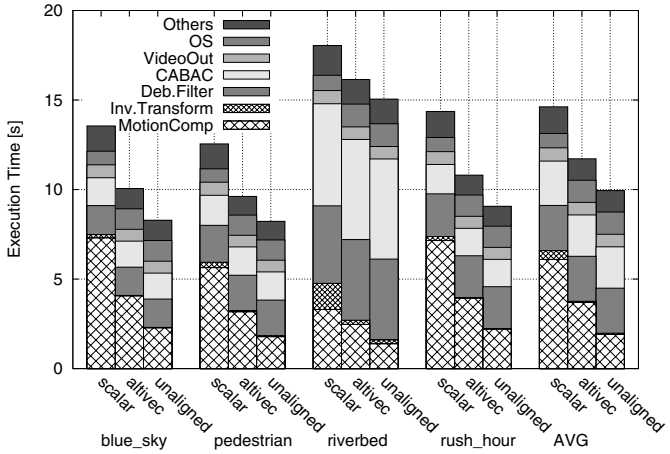


Fig. 10. Profiling of scalar, altivec and altivec_unaligned H.264/AVC decoder

version but additionally tolerates better the latency increase; the speed-up decreases 0.64% (1.09X) and 7.62% (1.02X) with one and 6 more cycles of latency respectively.

In the SAD kernel, speed-ups are obtained in all the cases with the exception of 6 extra cycles in the 16x16 size. That is because this kernel is dominated by the memory access and the latency of the loads is very significant for the performance of the SIMD vectorization.

Summarizing, most of the kernels that use unaligned memory instructions exhibit considerable speed-ups when compared to the original Altivec version until the point where we double the memory latency. Using the proposed hardware design, it is possible to perform a load with just one extra cycle of latency and a store with two cycles (that means 5 and 6 cycles respectively). In such a case, most of the kernels under study achieve a significant speed-up with respect to the original Altivec version.

D. Impact of Unaligned SIMD Instructions in a Complete H.264/AVC Decoder Application

Based on a profiling analysis of the H.264/AVC decoder, we have estimated the impact of the unaligned memory instructions on the complete application. Figure 10 shows a profiling of the H.264 decoder for the scalar, Altivec, and Altivec with unaligned instructions implementations. The profiling was conducted on a PowerPC 970 machine using the Apple Chud tools. The profiling shows that the most time consuming parts of the H.264/AVC decoder are the MC routines, the deblocking filter, the entropy coding (CABAC) and the inverse transform (IDCT). The kernels that have been optimized using Altivec SIMD instructions are the MC stage, which includes the Luma and Chroma interpolation, and the IDCT. The CABAC entropy coding is a kernel with a strong serial behavior that it is not amenable for SIMD optimization. The deblocking filter, although is an excellent candidate to benefit from unaligned memory access support, contains a high number of conditional branches that complicate the SIMD vectorization. A SIMD optimized version for the deblocking filter is currently under development.

Given the performance improvements obtained in the SIMD optimized kernels, the Altivec version of the code is 1.2X faster than the scalar version. The code optimized using unaligned instructions ranges from 1.16X to 1.23X faster. The average speedup is 1.20X compared to the plain Altivec version, and 1.49X compared to the scalar version.

The input content has a very big impact on the performance of the video decoder. For example, the riverbed sequence includes highly complicated motion of fluids in which the encoder can not use MC effectively, resulting in a few macroblocks coded as inter-macroblocks. That, in turn, reduces the relevance of the MC stage and reduces the potential for optimization with unaligned instructions.

VI. Conclusion

We have evaluated the performance impact of extending current SIMD ISAs with instructions for unaligned memory accesses in the context of emerging video codec applications like H.264/AVC. We have shown that for these kind of applications (but not limited to them) there is a big overhead for the SIMD memory accesses due to presence of unpredictable unaligned memory references. We have shown that this overhead comes from the additional instructions that are necessary for doing the data realignment in software.

The main impact of supporting non-aligned memory instructions is a significant reduction of the number of dynamically executed instructions. Unaligned instructions allow the programmer to remove code overhead necessary for data realignment. In addition, control flow instructions used to tailor code to the precise data unalignment offset can also be avoided. This reduction of overhead instructions make SIMD vectorization much easier, both for the manual developer and the autovectorizing compiler. Whenever the alignment of data can not be verified at compile time or fixed in the application, the unaligned instruction can be used safely.

Our results show that for some kernels the presence of overhead instructions prevents successful vectorizations, while unaligned instructions allow successful exploitation of DLP. Such is the case of the 4x4 pixel blocks that are common in new video codecs that uses variable block size ME. In general the unaligned memory operations increase the opportunities for SIMD vectorization in latency dominated code and in codes with highly unpredictable alignments. Video codec workloads are only one example of such case of applications.

As a result of the significant reductions in the amount of executed code, our results show that key kernels in the H.264/AVC decoder and encoder benefit from important speed-ups due to unaligned memory instructions. The speed-ups range from 1.06X in the IDCT, where data accesses are mostly aligned, to 2.1X in the Luma interpolation kernel, where unaligned accesses are the norm.

We also measure the impact of the added latency of the realignment network in the obtained results, and find that most of the kernels obtain sensible performance improvements if

the hardware does not introduce large latency increases (up to $4+4$ cycles in our simulations). These results help explain why long latency mechanisms used in some current processors, like microcode expansion or half-size datapaths do not result in high speed-ups in current applications.

We conclude that instructions for unaligned memory accesses are extremely useful for media SIMD extensions because they allow a significant reduction of the memory access overhead, allow vectorization of codes in which it is necessary to access small amount of data with low latency, and because they also lead to an easier SIMD software development. We also support the approach of having both types of instructions, aligned like in the original Altivec, and unaligned like those evaluated in this study. The original aligned instructions can be used when the alignment is predictable or known at compile time and, in turn, they can be used as a hint to the processor in order to optimize the memory accesses when all the data is aligned.

ACKNOWLEDGMENT

This work has been supported by the Ministry of Science and Technology of Spain, the European Union (FEDER funds) under contract TIC2004-07739-C02-01, the European Project SARC (FET-27648) and HiPEAC (The European Network of Excellence on High-Performance Embedded Architecture and Compilation). We acknowledge the Barcelona Supercomputing Center (BSC) for supplying the computing resources for our research.

REFERENCES

[1] K. Diefendorff and P. Dubey, "How Multimedia Workloads Will Change Processor Design," *IEEE Micro*, vol. 30, no. 9, pp. 43–45, Sept 1997.

[2] N. T. Slingerland and A. J. Smith, "Multimedia Instruction Sets for General Pupose Microprocessors: A Survey," UCB, Tech. Rep. CSD-00-1124, Dec. 1999.

[3] T. M. Conte, P. K. Dubey, M. D. Jennings, R. B. Lee, A. Peleg, S. Rathnam, M. Schlansker, P. Song, and A. Wolfe, "Challenges to Combining General Purpose and Multimedia Processors," *IEEE Computer*, vol. 30, no. 12, pp. 33–37, Dec. 1997.

[4] N. Slingerland and A. J. Smith, "Measuring the Performance of Multimedia Instruction Sets," *IEEE Transactions on Computers*, vol. 51, no. 11, pp. 1317–1332, Nov 2002.

[5] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video Coding with H.264/AVC: Tools, Performance, and Complexity," *IEEE Circuits and Systems Magazine*, vol. 4, no. 1, pp. 7–28, Jan 2004.

[6] S. Thakkar and T. Huff, "The Internet Streaming SIMD extensions," *IEEE Computer*, vol. 32, no. 12, pp. 26–34, Dec. 1999.

[7] D. Boggs, A. Baktha, J. Hawkins, D. Marr, J. Miller, P. Roussel, R. Singhal, B. Toll, and K. S. Venkatraman, "The Microarchitecture of the Intel Pentium 4 Processor on 90nm Techonology," *Intel Technology Journal*, vol. 08, no. 01, pp. 7–23, February 2004.

[8] Intel Corporation, "Block-Matching In Motion Estimation Algorithms Using Streaming SIMD Extensions 3," Intel Corporation," Application Note, 2003.

[9] D. Nuzman and R. Henderson, "Multi-platform auto-vectorization," in *International Symposium on Code Generation and Optimization. CGO'06*, March 2006, pp. 281–294.

[10] K. Diefendorff, P. Dubey, R. Hochsprung, and H. Scales, "AltiVec Extension to PowerPC Accelerates Media Processing," *IEEE Micro*, vol. 20, no. 2, pp. 85–95, April 2000.

[11] J.-W. van de Waerdt, S. Vassiliadis, S. Das, S. Mirolo, C. Yen, B. Zhong, C. Basto, J.-P. van Itegem, D. Amirtharaj, K. Kalra, P. Rodriguez, and H. van Antwerpen, "The TM3270 Media-Processor," in *MICRO 38: Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, Nov 2005, pp. 331–342.

[12] Texas Instruments, "TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide," Texas Instruments, User manual SPRU732C, 2005.

[13] J. Fridman, "Data Alignment for Sub-word Parallelism in DSP," in *IEEE Workshop on Signal Processing Systems SiPS 99*, Oct 1999, pp. pages 251–260.

[14] IBM, "PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual," IBM Corp.," User's manual, 2005.

[15] A. Krall and S. Lelait, "Compilation Techniques for Multimedia Processors," *International Journal of Parallel Programming*, vol. 28, no. 4, pp. 347–361, Aug 2000.

[16] S. Larsen and E. Witchel and S. Amarasinghe, "Techniques for Increasing and Detecting Memory Alignment," MIT Laboratory for Computer Science, Research Report MIT-LCS-TM-621, Nov. 2001.

[17] A. E. Eichenberger, P. Wu, and K. O'Brien, "Vectorization for SIMD Architectures with Alignment Constraints," in *PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, June 2004, pp. 82–93.

[18] I. E. G. Richardson, *Video Codec Design: Developing Image and Video Compression Systems*. John Wiley and Sons, 2002.

[19] IBM, "IBM PowerPC 970FX RISC Microprocessor User's Manual," IBM Corp., User's manual v1.6, Feb 2006.

[20] S. K. Raman, V. Pentkovski, and J. Keshav, "Implementing Streaming SIMD Extensions on the Pentium III Processor," *IEEE Micro*, vol. 20, no. 4, pp. 47–57, Aug. 2000.

[21] T. Conte, K. Menezes, P. Mills, and B. Patel, "Optimization of Instruction Fetch Mechanisms for High Issue Rates," in *22nd Annual International Symposium on Computer Architecture*, June 1995, pp. 333–344.

[22] E. Rotenberg, J. Smith, and S. Bennett, "Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching," in *29th Annual International Symposium on Computer Architecture*, 1996, p. 24.

[23] F. Quintana, J. Corbal, R. Espasa, and M. Valero, "Adding a Vector Unit on a Superscalar Processor," in *International Conference on Supercomputing*, June 1999, pp. 1–10.

[24] M. Moudgill, J.-D. Wellman, and J. Moreno, "Environment for PowerPC Microarchitecture Exploration," *IEEE Micro*, vol. 19, no. 3, pp. 15–25, May-Jun 1999.

[25] Freescale Semiconductor, "AltiVec Technology Programming Interface Manual," Freescale Semiconductor, User manual Altivecpim/D 6/1999, 1999.

[26] J. M. Tendler, J. S. Dodson, J. S. F. Jr., H. Le, and B. Sinharoy, "POWER4 System Microarchitecture," *IBM Journal of Research and Development*, vol. 46, no. 1, pp. 5–26, 2002.

[27] "FFmpeg Multimedia System. A Solution to Record, Convert and Stream Audio and Video," 2005, http://ffmpeg.sourceforge.net.

[28] M. Horowitz, A. Joch, and F. Kossentini, "H.264/AVC Baseline Profile Decoder Complexity Analyis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 704–716, July 2003.

[29] X. Zhou, E. Q. Li, and Y.-K. Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions," in *Proceedings of SPIE Conference on Image and Video Communications and Processing*, 2003.