Multimedia Extensions and Sub-word Parallelism in Image Processing: Preliminary Results

Marco Ferretti and Davide Rizzo

Dip. Informatica e Sistemistica, University of Pavia,
Via Ferrata 1, 27100 Pavia, Italy
{ferretti,rizzo}@elzira.unipv.it

Abstract. General purpose microprocessors have long been considered a computing platform unsuited to image processing and vision tasks. The so-called Von-Neuman paradigm and the associated memory bottleneck have motivated the research into various forms of parallel processing and of special processors for vision. The outcome of this long standing effort is negligible, if one considers the computing platforms that became a true product. Recently, the micro-architecture of some general purpose microprocessors has been augmented with extensions to support multimedia processing. It is worthwhile considering how much speed-up can be actually obtained by the limited SIMD processing mode that is embedded in these extensions. This paper presents experimental results obtained on a very simple algorithm, the Haar transform, that has been coded for the HP and the Intel multimedia microengines. Preliminary results reported here show that the system environment (type and dimensions of first and second level caches, and compiler efficiency) affects considerably the theoretical speed-up due to the SIMD microengine.

1 Multimedia Extensions on General Purpose Processors

Modern processors are equipped with extended instruction sets (called *media instruction sets*) designed to cope with the increasing demand of processing power of multimedia applications. Sounds, images and video have become elementary data to many applications and their transmission and processing over Internet has changed the overall framework for evaluating the performance of processors. This process has led to changes in the architectures of the computing platform. We can distinguish two main directions: one approach relies on modifications to existing general purpose processors instruction set architectures such as HP MAX-2 [1], Intel MMX [2,3], UltraSparc VIS [4] and MIPS MDMX [5]; the other approach is represented by specific processors targeted to embed multimedia applications (Multimedia processors MMPs) like Philips TriMedia TM-1[6], Samsung [7]and Mpact [8].

In the past many efforts have been made to implement efficiently image processing and, to some extent, multimedia algorithms on specialised or general purpose parallel architectures. These efforts have brought to the design and construction of many prototypes, and to much fewer commercial systems. Most of these machines adopted a SIMD architecture; some followed the MIMD approach. Specialised processors exhibit systolic, wavefront and similar styles, down to the extremely narrow ASIC paradigm of an integrated- circuit-per-algorithm solution. General purpose parallel processors have not proved flexible enough, and above all have not found a market. Today multimedia application apparently justify the extensions to the micro-architecture of general purpose microprocessors; it is no surprise that they have recovered the SIMD paradigm, with a lower degree of parallelism, but with a much wider diffusion and specially much lower costs than in previous systems. Video and enhanced DSP processors are likely to fill the market share opened up by set-top-boxes, games and high quality audio. We concentrate here in the field of mainstream image and audio processing that is likely to be required by most applications.

Indeed, we can find some characteristics common to the various multimedia extensions, which emerge from the analysis of the of multimedia algorithms. When considering how to extend instruction sets, the starting point for all companies has been an analysis on operations common to simple tasks in image processing, communications and audio processing. The outcome of this analysis is somewhat different, since not all groups involved in the design of multimedia extensions have privileged the same algorithms. Nevertheless, most multimedia applications are characterised by small data types (in general 8, 16 bits width), large quantity of data to be processed in a continuous stream, operations with great data parallelism, real-time processing requirements and large I/O bandwidth requirements.

The paradigm adopted is *sub-word parallelism* (or *sub-word execution model*) [1,9] which exploits the wide paths available nowadays at the various levels of interconnection: at the frontside bus that connects main memory to the second level cache (when available), at the backside bus between the second level cache and primary caches, and at the processor core. At each of this interconnections, a minimum of 64-bit parallelism is available; this allows for parallel data transfer and parallel computations over lower-precision data (for example 8 bytes, 4 words, or 2 double words in the MMX implementation; MAX-2 only supports a 4 word parallelism).

Among the microarchitectures with a support for multimedia processing, we have worked on the Intel MMX and HP MAX-2. They provide two different sets of new instructions that can be subdivided in these classes: arithmetic, compare, shift, conversion/organisation, logic and multiply-accumulate (MMX only).

All the arithmetic operations are implemented in fixed point arithmetic, with support for either *wraparound* (or *modulo*), *signed* or *unsigned saturation*. Saturation arithmetic is one of the really new key features, since parallel fixed point operations need a way to manage overflow/underflow conditions [9] so common in image processing (consider, for example, the sum between two pixels: if the resulting value lies outside the representable range, using wraparound arithmetic, it would be a dark pixel in a linearly growing map of grey values, while using saturation, it is possible to clamp that value at the range's maximum, which makes much more sense).

Intel MMX instruction set is wider (57 new instructions) than MAX-2 (17 new instructions), which was released with the same goal, but rather different implementation costs. The MAX-2 extension is a minimal change to the existing PA-RISC micro-architecture, and it was designed in such a way to cause a minimal increase in the area requirements on the chip core. HP considered the 4-way parallelism the only really useful, and does not support the 8 byte data type available in MMX, nor the 32 bit one. The latter that can be handled with single precision floating point operations.

Another key feature of multimedia extension is the provision for parallel compare operations: data dependent comparison are very common in some image processing tasks. If handled with the usual RISC branch instructions and branch prediction logic, the resulting mispredictions add huge penalties to the execution within the superpipelined microarchitectures. Surprisingly, not all companies have considered the issue of data dependent computation with the same relevance. As an instance, HP MAX-2 has no parallel compare instruction, although the HP PA architecture does provide for a nullification mode that directly handles the negative outcome of a comparison. The discussion of this point is not central to this work, since we have considered initially an algorithm (a digital transform) that has no data dependent behaviour.

The microarchitecture and its support to SIMD processing is just one of factors for improving performance. At the opposite side of the processing chain, the compiler is the first actor, and often the determinant one. Effective, widespread use of an advanced microarchitecture is only possible if the software for application development does exploit the capabilities available. Currently no compiler is capable of automatically inferring and generating multimedia instructions in order to parallelise a high level fragment of C code, for example. At the present, compilers have facilities to embed assembly instructions inside the existing code, to parallelise the particular multimedia task/image processing operation chosen, but this technique relies heavily on the programmer's skills, rather on an optimising compiler technology back-end.

2 Algorithm: Haar Transform

For our preliminary evaluation of multimedia extensions we have chosen the Haar transform algorithm [10]. It's a local algorithm based on convolutions and subsampling. It is often used in image processing applications, when the original image is processed through a decomposition operator, to generate a multiresolution representation. A typical application is image compression based on transform coding, where the Haar transform is used to decorrelate the original image producing a different domain representation. The Haar belongs to the class of Wavelet Transforms [11,12], whose main goal is to obtain a new multiresoluton representation of a signal. Wavelet basis functions used to compute the transform can be chosen so that they have some important features and properties useful in image processing: a limited support and scale-space invariance.

There exist methods to compute a fast wavelet transform in a recursive way. However, the Haar transform has a very simple non separable implementation; it consists of four convolutions based on filters with a small support, that fit perfectly with the required subsampling by 2. There are four of them, to produce four different bands of the decomposition at every transform step. The 2x2 filters being considered are:

Band0

1	1
1	1

Band1

1	1
-1	-1

Band2

1	-1
1	-1

Band3

1	-1
-1	1

The decomposition results from iterating on Band0; it is composed of *logN* levels. At every step, the total number of coefficients to be considered is reduced by a factor of 4. In many practical cases, there is no need to go through the complete decomposition, which can be stopped after four or five levels. This is the case of image compression. To obtain the original signal, the Inverse Haar Transform uses the same filters (normalised with a scaling factor of ¼) to be applied to the corresponding bands of each level in a top-down process.

2.1 Factors Affecting Performance on the Reference Systems

The experiments carried out on the reference systems depend mainly on three factors: i) algorithm style; ii) compilers optimisation; iii) image sizes.

As to the algorithm description, we coded the most straightforward C program derivable from the algorithm definition (sequential version) and then we coded a corresponding parallel version, using the multimedia extension instructions available. We further decided to split the experiments in two cases: computations of first level only (algorithm Haar1), and computations of multiple levels, up to the fourth (algorithm HaarM).

Since there is no automatic generation of the new multimedia instructions directly from C through the available compilers, we had some alternatives for using these new instruction sets. With MAX-2, the options are macro definition or coding of assembly routines to be called within a C program. We followed this second option. With MMX, we used direct assembly inlining in C program to perform core computations.

Each C algorithm has been further optimised using the existing optimising compilers, in order to compare the best code obtainable from deep (and automated) compiler optimisation to what can be directly obtained adopting an assembly hand-coded core module embedded in a C program. We used the *cc* level 3 optimisation (+O3) on the PA-RISC system, which includes, among others, branch optimisation, faster register allocation, instruction scheduling, loop invariant code motion, software pipelining, register reassociation and function inlining. On the Pentium system, we selected Intel C compiler level 2 optimisation, which includes the following: optimised code selection, global register allocation, instruction scheduling, inline expansion, loop invariant code movement and copy propagation. Another option, available only on PA-RISC processors for MAX-2, is the dataprefetch facility, which could produce better results, in situations where the cache acts as the bottleneck of the chain; the available compiler adds dataprefetch instructions on writes only, while MAX-2 hand-coded algorithms use dataprefetch on reads as well.

The performances index is the time required to complete the kernel of the operations, once the image and all the other results have been allocated. We used two different ways to measure time, on the basis of the facilities that were provided on the particular reference system, and we repeated the experiment several times (more than 50), taking the minimum time in the end. For the MAX-2 measures, we used the *gettimeofday()* Unix system call, which provides a time resolution of microseconds. For MMX, we used the Time Stamp Counter facility provided by Pentium II processors, to obtain the clock ticks elapsed since the start of the core algorithm: this allows to obtain more precise results than through the Unix system call.

			HP-PA	INTEL	
			L1 (1 Mb)	L1 (16Kb)	L2 (512 Kb)
Label	Image dim. (pixels)	Image size (Kb)	Cache size/ image size	Cache size/ image size	Cache size/ image size
T	128 x 128	16	n.a.	1	32
XS	256 x 256	64	n.a.	0.25	8
S	512 x 512	256	4	0.062	2
M	768 x 640	480	2.13	0.031	1.06
L	1280 x 1024	1280	0.8	0.012	0.4

Table 1. Ratio of cache size to image size. Labels T (Tiny) through L (Large) denote images of increasing dimensions. Cases T and XS apply only to INTEL MMX experiments.

As to images, we initially chose three 8-bit greyscale PGM images of different dimensions, denoted as Small (S), Medium (M) and Large (L) (see Table 1 for details), for the experiments on the HP-PA MAX-2. When working on MMX, we had to introduce two smaller images, eXtra Small (XS) and Tiny (T) because of the extremely different configurations and dimensions of the caches in the two reference systems.

3.2 Algorithm Coding

Independently of the language, in coding the algorithms we made some choices regarding memory allocation. For Haar1 algorithm, there is nothing particular, apart the need to reorganise the data read in order to fully exploit the parallelism (see MAX-2 experiments).

For algorithm HaarM, the problem regards the memory needed for intermediate computations. In order to exploit the inherent parallelism, the MAX-2 and MMX versions operate on a group of 4 adjacent result coefficients (on the same row) at a time, and this is true in each band, so that each iteration produces 16 coefficients. Each iteration reads in data from two consecutive input rows (initially 8 pixels in each row, then 8 coefficients values in each row), rearranges them and computes in parallel four coefficients in the same band. The parallel strategy can be realised in different ways. For example Intel [13] suggests a strategy based, among others, on the Parallel Multiply Accumulate instructions (PMAC), that handles a group of data and of coefficients stored in two parallel registers. Such a strategy cannot be used with MAX-2 extension, since no parallel multiply-add instruction is provided. PMAC is not needed really, since products factors are 1 or –1. It could also become a bottleneck, because its latency is 3 clock cycles, while all the others are 1-cycle latency instructions (assuming cache hits).

In the MAX-2 strategy the four data corresponding to the same band coefficient computation are placed in the same register word position and then accumulated according to the four Haar filters, using only additions, subtractions and logical operations. We implemented this scheme also in the MMX case. This approach can take advantage of the internal Pentium II microarchitecture. The MMX execution units are connected to two ports of the Reservation Station [14]: Port 0 has MMX ALU Unit and MMX multiplier Unit while Port 1 has MMX ALU Unit and MMX Shifter Unit. So the logical and arithmetical MMX instructions can be dispatched at the same time to two MMX ALU units, while multiplications can only be dispatched one at a time to a single port of the Reservation Station. Moreover these latter have a 3 cycle latency, while the others are one cycle.

A note on the precision of results. We have adopted a 16-bit precision for computation first because input data is 8-bit wide and we need a larger precision to accomplish exact computations; secondly, this allows a parallelism of four. A drawback is that we cannot compute the whole decomposition in all levels. The highest possible level, which still grants perfect reconstruction, is the third, because the growth in range of coefficients over higher levels, can cause a loss of precision that prevents an exact signal reconstruction. So when going to levels higher than third we may have to accept a potential loss of information and a potential reconstruction error (the loss happens in certain worst case images, that can represent however very extreme and not practical images). For MMX this situation could be avoided, reducing the parallelism to two double word (32-bit) computations at a time. Doing so, no loss of information is caused, but the speedup obtainable gets smaller. This however should not be a problem because the proportional relevance of higher level computations is very small with respect to total time needed to perform the first three levels. But with MAX-2 this is not feasible.

The output image is twice as large as the original one, since the transform of a NxN image is composed of NxN wavelet coefficients. As mentioned earlier, in order to perform correctly the filtering operation, the resulting coefficients have a doubled bit depth with respect to the input pixel values. The consequence is that we cannot superimpose the new values directly on the input image, because of data width mismatch. Some buffering is required for intermediate results. Once the first level transform is computed on a separate memory area (twice as large as that of the input image), in order to compute the second level one idea could be to avoid allocating further memory, trying to use the area allocated for Band0 to store both the approximation and the details of the second level. This is not possible without further data reorganization and temporary storage areas, since detail coefficients would replace approximation data not yet used in any computation. The strategy we have adopted is based on a *double buffering scheme*. During each level, the results are written to a different memory area and then, at the end of the transform, a final copy "merges" the results producing the usual wavelet structure. In this way no rearrangement operation, distinct from a copy, is needed, and the additional temporal buffer is equal to one half of the original input image (a quarter of the resulting transformed image).

4 Results

We have used two different hardware architectures to perform our evaluation on the chosen algorithm: a true RISC processor and an evolution from a CISC processor which adopts internal microarchitectural RISC features. The reference systems are an HP C180 series 700 workstation and an HP PC Vectra/233 Series DT.

The HP C180 is a Unix workstation running on HP/UX 10.20, with 128 MB RAM and 1 MB each of instruction and data L1 cache. The PA-RISC 2.0 processor runs at 180 MHz. The system has no L2 cache, a somewhat unique feature within RISC workstations. We used ANSI C with *cc* compiler v. A.10.32.

The HP Vectra system is a Windows NT workstation, equipped with a 233 MHz Pentium II, 96 MB RAM, 16 KB of first level (L1) data cache and the same value for instruction cache, and 512 KB of secondary (L2) cache. The compilers used are both Microsoft Visual C++ 5.0 and Intel C compiler plug-in v2.4. We also used VTune v.2.5, a tool provided by Intel to analyse Intel processor's performances.

While performing the experiments, the user had no particular priority privilege and the systems had all the services of a typical multi-user environment.

4.1 MAX-2 Experiments

At the beginning of the experiments we had to face the data re-organisation problem: since input images are in PGM format, with 8 bits per pixel, in order to fully exploit MAX-2 extensions, data must be rearranged to 16-bit width; this rearrangement operation can be performed either "externally" to the transform algorithm thus producing input data of the correct width or "internally", working directly on 8-bit input data. In the first case, the required memory is larger (*4mn* bytes, for *m x n* image against *3mn*); the algorithm has an additional step and its core still has to manage a partial data re-organisation, functional to the parallelism adopted. In the second case, the advantage is represented by the broader bandwidth in data memory transfer since every read operation can take 8 pixels at the same time, while in the other mode the bandwidth is halved, since only four data pixels are transferred at a time. This alternative regards only the first level computations, since after that results are 16-bit wide.

Experiments showed that the internal conversion is always better (see Table 2), so we have adopted this technique throughout all cases. The evaluations have been made also with a comparison of dataprefetch instructions, available in PA8000 processors.

Haar1MAX-2	S	M	L
External no dp	1.697	11.477	45.671
External dp	1.600	10.205	40.426
Internal no dp	1.404	5.672	36.087
Internal dp	1.291	4.965	33.155

Table 2. Times (msec) for 1 level Haar computation with external vs. internal input data width conversion, with (dp) and without (nodp) dataprefetching.

Table 3 shows the outcome of the execution of assembly coded algorithm for the direct and inverse 1-level Haar: the speed up is computed against the optimised C version.

We can observe that an acceptable speedup can be obtained only in the case of the small image. In fact this image can be completely contained in the L1 cache. With larger images, the parallelism advantage is lost and the main cause is the great number of cache misses. As an extreme case, no gain is obtained with large images.

Data prefetch instructions are useful with larger images, where the hints to data load/store can reduce cache misses. Again cache misses with larger images prevent full parallel ideal speedup.

Algorithm	S		M		L	
	T	*Sp.up*	*T*	*Sp.up*	*T*	*Sp.up*
Haar1MAX-2 no dp	1.404	2.84	5.672	1.97	36.087	1.17
Haar1MAX-2 dp	1.291	2.92	4.965	1.52	33.155	1.00
IHaar1MAX-2 no dp	2.638	3.03	8.157	2.10	32.795	1.68
IHaar1MAX-2 dp	1.828	3.69	5.836	2.23	27.285	1.34

Table 3. Execution times (msec) and speed up relative to C code for 1-level direct (Haar) and inverse (Ihaar) transform with HP-PA MAX-2

Considering the multi-level Haar transform (HaarM), we obtained the results shown in Table 4 (we have reported the computation times up to the fourth level, even if these are related to Haar transform with reconstruction errors).

Algorithm	lev	S		M		L	
		T	*Sp.up*	*T*	*Sp.up*	*T*	*Sp.up*
HaarM MAX-2 no dp	2	1.844	3.22	10.518	1.74	52.246	1.29
	3	1.881	3.41	10.703	1.80	53.910	1.30
	4	1.910	3.46	10.727	1.83	54.158	1.32
HaarM MAX-2 dp	2	1.833	3.21	9.185	1.36	48.947	1.03
	3	1.930	3.15	9.374	1.39	50.279	1.03
	4	1.956	3.15	9.443	1.40	50.755	1.04

Table 4. Execution times (msec) and speed up relative to C code for multi-level (lev) direct Haar transform with HP-PA MAX-2

We can see that the time increase to compute more than two levels is not really relevant, since the number of new coefficients to be computed is relatively small compared to first and second level.

4.2 MMX Experiments

For MMX experiments we used the internal data reorganisation mode, without further checks. We report comparisons between optimised C codes and MMX versions coded with inlining of parallel instructions. As previously mentioned, we were forced to introduce smaller images (T and XS) in order to observe true improvements from MMX parallel version of the algorithm with respect to the serial C version. Considering Table 5 for one level computations, it's clear that the ideal speedup can be obtained only when the images are completely contained in the cache hierarchy. Indeed, when this condition does not hold, (see columns S, M and L), cache penalties become relevant and performance is greatly reduced. The forward and the inverse transform exhibit the same pattern.

Algorithm	T		XS		S		M		L	
	T	*Sp.up*	*T*	*Sp.up*	*T*	*Sp.up*	*T*	*Sp.up*	*T*	*Sp.up*
Haar1 MMX	0.190	3.41	0.737	3.56	10.75	1.45	22.37	1.45	61.51	1.30
IHaar1MMX	0.178	3.54	0.696	3.68	6.652	2.14	18.02	1.55	50.98	1.73

Table 5. Execution times (msec) and speed up relative to C code for 1-level direct (Haar) and inverse (Ihaar) transform with INTEL MMX

Considering multiple levels (algorithm HaarM), Table 6 reports execution times and speedups for computations up to the fourth level. Again cache penalties are heavy when working with large images.

Algorithm	lev	T		XS		S		M		L	
		T	*Sp.up*	*T*	*Sp.up*	*T*	*Sp.up*	*T*	*Sp.up*	*T*	*Sp.up*
HaarM MMX	2	0.266	3.45	1.147	3.22	14.42	1.52	30.75	1.31	92.89	1.26
	3	0.273	3.54	1.151	3.39	14.56	1.55	30.88	1.36	93.48	1.27
	4	0.276	3.58	1.182	3.36	14.87	1.52	30.95	1.39	94.61	1.28
IIaarM MMX	2	0.279	2.94	1.114	2.98	10.47	1.73	29.79	1.27	86.16	1.47
	3	0.283	3.02	1.174	2.97	10.95	1.72	29.16	1.34	88.64	1.49
	4	0.284	3.05	1.175	2.99	10.97	1.73	30.70	1.29	90.47	1.48

Table 6. Execution times (msec) and speed up relative to C code for multi-level direct (Haar) and inverse (Ihaar) transform with INTEL MMX

5 Conclusions

The Haar transform in the formulation used here has some distinct features: it does not require image transposition to compute the bi-dimensional convolution, as is usual with standard implementations of wavelets; the computations are not data dependent, and the output of each step are re-used partially in subsequent steps.

The effect of such characteristics on the performance on superpipelined, RISC microprocessors is distinctly different. On the positive side, the regular structure of the computations (the control structure of the kernel of the transform is a simple nested loop) minimises branch mispredictions.

As already shown, locality in memory references and cache usage is instead a critical point. Since the image need not be transposed in any of the phases of the algorithm, a major cause of cache miss is eliminated. The end effect is therefore the ratio between image and caches' dimensions. The pattern of the speed-up against the dimension of the image shows this effect clearly. At this stage of the work, it is uncertain whether tuning memory references can bring substantial benefits.

On the side of software development, the exploitation of the small effects of sub-word parallelism is still a matter of good assembly programming skill. Improvements in compiler technology could make the effects of the limited SIMD parallel processing visible only if cache usage is optimised first.

References

1. Lee, R.: Subword Parallelism with MAX-2, IEEE Micro, 16, n.4, (1996), 51-59
2. Peleg, A., Weiser, U., MMX Technology Extension to the Intel Architecture, IEEE Micro, July/Aug. (1996), 42-50
3. The complete Guide to MMX Technology, McGraw-Hill, (1997)
4. Tremblay et al.: VIS Speeds New Media Processing, IEEE Micro, July/Aug., (1996), 10-20
5. MIPS Digital Media Extension, Instruction Set Architecture Specification, http://www.mips.com/Documentation/isa5_tech_brf.pdf
6. Slavenburg, G.A., Rathnam, S., Dijkstra, H.: The Trimedia TM-1 PCI VLIW Media processor, Proc. Hot Chips VIII Symp., IEEE CS Press, Los Alamitos, Calif., (1996)
7. L.T. Nguyen et al: MSP: Multi-Media Signal Processor, Proc. Hot Chips VIII Symp., IEEE CS Press, Los Alamitos, Calif., (1996)
8. Kalapathy, P.: Hardware/Software Interactions on the Mpact, IEEE Micro, Mar./Apr. (1997), 20-26
9. Lee, R.: Accelerating Multimedia with Enhanced Microprocessors, IEEE Micro, 15, n. 2, (1995), 22-32
10. Haar, A.: Zur Theorie der Orthogonalen Functionensysteme, Math. Annal. 69, (1910), 331-371
11. Rioul, O., Vetterli, M.: Wavelets and Signal Processing, IEEE SP Magazine, October (1991), 14-37
12. Strang, G., Nguyen, T.: Wavelets and Filter Banks, Wellesley-Cambridge Press, (1996)
13. Using MMX Instructions to Compute the 2x2 Haar Transform, Intel Application Note AP-531, available at Intel Web site
14. Shanley, T.: Pentium Pro and Pentium II System Architecture, MindShare Inc., (1997)